Scilab manual

Table of Contents

Scilab

Name

abort — interrupt evaluation.

Description

`abort` interrupts current evaluation and gives the prompt. Within a `pause` level `abort` return to level 0 prompt.

See Also

quit , pause , break

Name

add_demo — Add an entry in the demos list

```
add_demo(title,path)
```

Parameters

title
: a character string, the demo title

path
: a character string, the path of the scilab script associated with the demo

Description

This function adds a new entry in the demos list. The demo should be executed by a Scilab script file. If the given `title` already exists in the demo list associated with the same file nothing is done. The function checks if the file exist.

Examples

```
//create a simple demo script
path=TMPDIR+'/foo.sce';
mputl('disp Hello',path)
add_demo('My first demo',path)
//the demo can now be run using the "Demos" menu.
```

See Also

add_help_chapter

Authors

Serge Steer , INRIA

Name

ans — answer

Description

`ans` means "answer". Variable `ans` is created automatically when expressions are not assigned. `ans` contains the last unassigned evaluated expression.

Name

argn — Returns the number of input/output arguments in a function call

```
[lhs [,rhs] ]=argn()
lhs=argn(1)
rhs=argn(2)
```

Description

This function is used inside a function definition. It gives the number of actual inputs arguments (`rhs`) and output arguments (`lhs`) passed to the function when the function is called. It is usually used in function definitions to deal with optional arguments.

Examples

```
function concat=myOwnFunction(name,optional)
  [lhs,rhs]=argn(0)
  if rhs <= 1 then
    optional="my Optional value"
  end
  if rhs == 0 then
    error("Expect at least one argument")
  end
  concat=name+" "+optional
endfunction
```

See Also

function , varargin

Name

backslash (\) — left matrix division.

```
x=A\b
```

Description

Backslash denotes left matrix division. `x=A\b` is a solution to `A*x=b`.

If `A` is square and nonsingular `x=A\b` (uniquely defined) is equivalent to `x=inv(A)*b` (but the computations are much cheaper).

If `A` is not square, `x` is a least square solution. i.e. `norm(A*x-b)` is minimal (euclidian norm). If `A` is full column rank, the least square solution, `x=A\b`, is uniquely defined (there is a unique `x` which minimizes `norm(A*x-b)`). If `A` is not full column rank, then the least square solution is not unique, and `x=A\b`, in general, is not the solution with minimum norm (the minimum norm solution is `x=pinv(A)*b`).

`A.\B` is the matrix with `(i,j)` entry `A(i,j)\B(i,j)`. If `A` (or `B`) is a scalar `A.\B` is equivalent to `A*ones(B).\B` (or `A.\(B*ones(A))`

`A\.B` is an operator with no predefined meaning. It may be used to define a new operator (see overloading) with the same precedence as * or /.

Examples

```
A=rand(3,2);b=[1;1;1]; x=A\b; y=pinv(A)*b;  x-y
A=rand(2,3);b=[1;1]; x=A\b; y=pinv(A)*b; x-y, A*x-b, A*y-b
A=rand(3,1)*rand(1,2); b=[1;1;1]; x=A\b; y=pinv(A)*b; A*x-b, A*y-b
A=rand(2,1)*rand(1,3); b=[1;1]; x=A\b; y=pinv(A)*b; A*x-b, A*y-b
```

See Also

slash , inv , pinv , percent , ieee

Name

banner — show scilab banner (Windows)

```
banner()
```

Description

show scilab banner.

Examples

```
clc();banner()
```

Authors

Allan CORNET

Name

boolean — Scilab Objects, boolean variables and operators & | ~

Description

A boolean variable is `%T` (for "true") or `%F` (for "false"). These variables can be used to define matrices of booleans, with the usual syntax. Boolean matrices can be manipulated as ordinary matrices for elements extraction/insertion and concatenation. Note that other usual operations (`+, *, -, ^,` etc) are undefined for booleans matrices, three special operators are defined for boolean matrices:

~b
: is the element wise negation of boolean `b` (matrix).

b1&b2
: is the element wise logical `and` of `b1` and `b2` (matrices).

b1|b2
: is the element wise logical `or` of `b1` and `b2` (matrices).

Boolean variables can be used for indexing matrices or vectors.
: For instance `a([%T,%F,%T],:)` returns the submatrix made of rows `1` and `3` of `a`. Boolean sparse matrices are supported.

Examples

```
[1,2]==[1,3]
[1,2]==1
a=1:5; a(a>2)
```

See Also

matrices , or , and , not

Name

brackets — ([,]) left and right brackets

```
[a11,a12,...;a21,a22,...;...]
[s1,s2,...]=func(...)
```

Parameters

a11,a12,...
: any matrix (real, polynomial, rational,`syslin` list ...) with appropriate dimensions

s1,s2,...
: any possible variable name

Description

Left and right brackets are used to note vector and matrix concatenation. These symbols are also used to denote a multiple left-hand-side for a function call

Inside concatenation brackets, blank or comma characters mean "column concatenation", semicolon and carriage-return mean "row concatenation".

Note : to avoid confusions it is safer to use commas instead of blank to separate columns.

Within multiple lhs brackets variable names must be separated by comma.

Examples

```
[6.9,9.64; sqrt(-1) 0]
[1 +%i  2 -%i  3]
[]
['this is';'a string';'vector']
s=poly(0,'s');[1/s,2/s]
[tf2ss(1/s),tf2ss(2/s)]

[u,s]=schur(rand(3,3))
```

See Also

comma , semicolon

Name

break — keyword to interrupt loops

Description

Inside a `for` or `while` loop, the command `break` forces the end of the loop.

Examples

```
k=0; while 1==1, k=k+1; if k > 100 then  break,end; end
```

See Also

while , if , for , abort , return

Name

case — keyword used in select

Description

Keyword used in `select ... case`

Use it in the following way:

```
select expr0,
 case expr1 then instructions1,
 case expr2 then instructions2,
 ...
 case exprn then instructionsn,
 [else instructions],
end
```

See Also

select , while , end , for

Name

chdir — changes Scilab current directory
cd — changes Scilab current directory

```
b=chdir(path)
realpath=cd(path)
cd path
```

Parameters

b
: a boolean %t if chdir operation is ok.

path
: a character string

realpath
: a character string, the real path name after pathname conversion (see below)

Description

Change the current Scilab directory to those given by `path`. Note that path conversion is performed and for example `SCI/modules/core/macros` is a valid pattern for both unix and windows. If path is empty change to "home" directory.

Examples

```
chdir(TMPDIR);
pwd
cd
cd SCI
```

See Also

getcwd

Name

clear — kills variables

```
clear a
```

Description

This command kills variables which are not protected. It removes the named variables from the environment. By itself `clear` kills all the variables except the variables protected by `predef`. Thus the two commands `predef(0)` and `clear` remove all the variables.

Normally, protected variables are standard libraries and variables with the percent prefix.

Note the particular syntax `clear a` and not `clear(a)`. Note also that `a=[]` does not kill `a` but sets `a` to an empty matrix.

See Also

predef , who

Name

clearfun — remove primitive.

```
ret=clearfun('name')
```

Description

`clearfun('name')` removes the primitive `'name'` from the set of primitives (built-in functions). clearfun returns %t or %f. This function allows to rename a primitive : a Scilab primitive can be replaced by a user-defined function. For experts...

See Also

newfun , funptr

Name

clearglobal — kills global variables

```
clearglobal()
clearglobal nam1 .. namn
clearglobal('nam1', ..,'namn')
```

Parameters

nam1,..., namn
: valid variable names

Description

`clearglobal()` kills all the global variables.

`clearglobal nam1 .. namn` kills the global variables given by their names

Note that `clearglobal()` only clears the global variables, the local copies of these global variables are not destroyed.

Examples

```
global a b c
a=1;b=2;c=3;
who('global')
clearglobal b
who('global')
```

See Also

global , who

Name

colon — (:) colon operator

Description

Colon symbol : can be used to form implicit vectors. (see also linspace, logspace)

j:k
: is the vector `[j, j+1,...,k]` (empty if `j>k`).

j:d:k
: is the vector `[j, j+d, ..., j+m*d]`

The colon notation can also be used to pick out selected rows, columns and elements of vectors and matrices (see also extraction,insertion)

A(:)
: is the vector of all the elements of `A` regarded as a single column.

A(:,j)
: ys the `j`-th column of `A`

A(j:k)
: is `[A(j),A(j+1),...,A(k)]`

A(:,j:k)
: is `[A(:,j),A(:,j+1),...,A(:,k)]`

A(:)=w
: fills the matrix `A` with entries of `w` (taken column by column if `w` is a matrix).

See Also

matrix , for , linspace , logspace

Name

comma — (,) column, instruction, argument separator

Description

Commas are used to separate parameters in functions or to separate entries of row vectors.

Blanks can also be used to separate entries in a row vector but use preferably commas.

Also used to separate Scilab instructions. (Use `;` to have the result not displayed on the screen).

Examples

```
a=[1,2,3;4,5,6];
a=1,b=1;c=2
```

See Also

semicolon , brackets

Name

comments — comments

Description

A sequence of two consecutives slashes `//` out of a string definition marks the beginning of a comment. The slashes as well as all the following characters up to the end of the lines are not interpreted.

Inside a function, the first comment lines, up to the first instruction or an empty line may be used to provide the default contents for the function help.

Examples

```
g=9.81// the gravity

text='a//b'

function y=myfunction(x)
// myfunction computes y=x^2+1
// x shoud be a vector or matrix
  y=x^2+1
endfunction

help myfunction
```

Name

comp — scilab function compilation

```
comp(function [,opt])
```

Parameters

function
: a scilab function, not compiled (type 11)

opt
: flag with value 0 (default), 1 or 2.

Description

`comp(function)` compiles the function `function`. Compiled and interpreted functions are equivalent but usually compiled functions are much faster. The functions provided in the standard libraries are compiled.

The online definition as well as the short syntax of the commands `getf` and `deff` generate compiled functions. So comp has to be used in very particular cases. To produce uncompiled functions one must use >getf or deff with the option `"n"`.

The value opt==2 causes the function to be compiled "for profiling". Note that now it is possible to add profiling instruction after compilation using the add_profiling function.

The osolete opt==1 option was specific to code analysis purposes and is now ignored, i.e treated as opt==0.

Note: the compilation takes part "in place", i.e the original function is modified and no new object is created.

See Also

type, deff, getf, function, add_profiling, profile

Name

comparison — comparison, relational operators

```
a==b
a~=b or a<>b
a<b
a<=b
a>b
a>=b
```

Parameters

a
: any type of variable for `a==b`, `a~=b` `a<>b` equality comparisons and restricted to real floating point and integer array for order related comparisons `a<b`, `a<=b`, `a>b`,`a>=b`.

b
: any type of variable for `a==b`, `a~=b` `a< > b` equality comparisons and restricted to real floating point and integer arrays for order related comparisons `a<b`, `a<=b`, `a>b`,`a>=b`.

Description

Two classes of operators have to be distinguished:

The equality and inequality comparisons:
: `a==b`, `a~=b` (or equivalently `a<>b`). These operators apply to any type of operands.

The order related comparisons:
: `a<b`, `a<=b`, `a>b`,`a>=b`. These operators apply only to real floating point and integer arrays.

The semantics of the comparison operators also depend on the operands types:

With array variables
: like floating point and integer arrays, logical arrays, string arrays, polynomial and rationnal arrays, handle arrays, lists... the following rules apply:

  - If `a` and `b` evaluates as arrays with same types and identical dimensions, the comparison is performed element by element and the result is an array of booleans of the same.
  - If `a` and `b` evaluates as arrays with same types, but `a` or `b` is a 1 by 1 array the scalar is compared with each element of the other array. The result is an array of booleans of the size of the non scalar operand.
  - In the others cases the result is the boolean `%f`
  - If the operand data types are differents but "compatible" like floating points and integers a type conversion is performed before the comparison.

With other type of operands
: like `function`, `libraries`, the result is `%t` if the objects are identical and `%f` in the other cases.

  Equality comparison between operands of incompatible data types returns `%f`.

Examples

```
//element wise comparisons
(1:5)==3
(1:5)<=4
(1:5)<=[1 4 2 3 0]
1<[]
list(1,2,3)~=list(1,3,3)


//object wise comparisons
(1:10)==[4,3]
'foo'==3
1==[]
list(1,2,3)==1

isequal(list(1,2,3),1)
isequal(1:10,1)

//comparison with type conversion
int32(1)==1
int32(1)<1.5
int32(1:5)<int8(3)
p=poly(0,'s','c')
p==0
p/poly(1,'s','c')==0
```

See Also

less , boolean , isequal

Name

continue — keyword to pass control to the next iteration of a loop

Description

Inside a `for` or `while` loop, the command `continue` passes control to the next iteration of the loop in which it appears, skipping any remaining statements between this instruction and the loop's `end` instruction.

Examples

```
for k=1:10,K=k;if k>2&k<=8 then continue,disp('hello'),end,k,end

for j=1:2
  x=[];
  for k=1:10,if k>j+1&k<=8 then continue,end,x=[x,k];end
  x
end
```

See Also

while , for , break , return

Authors

Serge Steer, INRIA

Name

debug — debugging level

```
debug(level-int)
level-int=debug()
```

Parameters

level-int
: integer (-1 to 4)

Description

For the values 0,1,2,3,4 of `level-int` , `debug` defines various levels of debugging. This is targeted to the parser, not to Scilab scripts, and is for Scilab experts only.

For the value -1 of `level-int` , `debug` defines a special level of debugging dedicated to the ScilabEval Tcl instruction.

See also the `Scipad` debugger, which is targeted to debugging Scilab scripts.

See Also

ScilabEval , scipad

Name

delbpt — delete breakpoints

```
delbpt([macroname [,linenumb]])
```

Parameters

macroname
: string

linenumb
: scalar integer or vector of integers

Description

Deletes the breakpoint at line `linenumb` in the function `macroname`.

`linenumb` can be a line or column vector of line numbers, or a single scalar line number.

If `linenumb` is omitted, all the breakpoints in function `macroname` are deleted.

If both `macroname` and `linenumb` are omitted, all the breakpoints in all the functions are deleted.

Examples

```
setbpt('foo',1),setbpt('foo',10),delbpt('foo',10),dispbpt()
delbpt('foo',1),dispbpt()

setbpt('foo1',4),setbpt('foo1',9),setbpt('foo2',6),setbpt('foo3',8),dispbpt()
delbpt('foo2'),dispbpt()
delbpt(),dispbpt()

delbpt('foo',[1,2,5,6]),dispbpt()
```

See Also

setbpt , dispbpt , pause , resume

Name

dispbpt — display breakpoints

```
dispbpt()
```

Description

`dispbpt()` displays all active breakpoints currently inserted in functions.

See Also

setbpt , delbpt , pause , resume

Name

do — language keyword for loops

Description

May be used inside `for` pr `while` instructions to separate the loop variable definition and the set of instructions.

See Also

for , while

Name

dot — (.) symbol

```
123.33
a.*b
[123,..
456]
```

Description

.

Dot is used to mark decimal point for numbers : 3.25 and 0.001

.<op>

used in conjunction with other operator symbols (`* / \ ^ '`) to form other operators. Element-by-element multiplicative operations are obtained using `.* , .^ , ./ , .\` or `.'`. For example, C = A ./ B is the matrix with elements c(i,j) = a(i,j)/b(i,j). Kronecker product is noted .*. . Note that when dot follows a number it is alway prt of the number so 2.*x is evaluated as 2.0*x and 2 .*x is evaluated as (2).*x

..

Continuation mark. Two or more decimal points at the end of a line (or followed by a comment) causes the following line to be a continuation.

Continuation lines are handled by a preprocessor wich builds a long logical line from a given sequence of continuation lines. So the continuation marks can be used to cut a line at any point.

Examples

```
//decimal point
1.345

//used as part of an operator
x=[1 2 3];x.^2 .*x // a space is required between 2 and dot

// used to enter continuation lines
T=[123,..//first element
   456] //second one

a="here I start a very long string...  //but I'm not in the mood of continuing
     - and here I go on"
y=12..
45
```

See Also

star , hat , slash , backslash

Name

edit — function editing

```
edit(functionname)
```

Parameters

functionname
: character string

Description

If functionname is the name of a defined scilab function `edit(functionname)` try to open the associated file `functionname.sci`.

If functionname is the name of a undefined scilab function `edit` create a functionname.sci file in the TMPDIR directory.

Examples

```
edit('edit')  //opens editor with text of this function
edit('myfunction')  //opens editor for a new function
```

See Also

manedit, scipad

Name

else — keyword in if-then-else

Description

Used with `if`.

See Also

if

Name

elseif — keyword in if-then-else

Description

See `if,then,else` .

Name

empty — ([]) empty matrix

Description

`[]` denotes the empty matrix. It is uniquely defined and has 0 row and 0 column, i.e. `size([]) =[0,0]` . The following convenient conventions are made:

```
[] * A = A * [] = []
```

```
[] + A = A + [] = A
```

[[], A] = [A, []] = A `inv([]) =[]`

```
det([])=cond([])=rcond([])=1, rank([])=0
```

Matrix functions return `[]` or an error message when there is no obvious answer. Empty linear systems (`syslin` lists) may have several rows or columns.

Examples

```
s=poly(0,'s'); A = [s, s+1];
A+[], A*[]
A=rand(2,2); AA=A([],1), size(AA)
svd([])
w=ssrand(2,2,2); wr=[]*w; size(wr), w1=ss2tf(wr), size(w1)
```

See Also

matrices , poly , string , boolean , rational , syslin

Name

end — end keyword

Description

Used at end of loops or conditionals. `for, while, if, select` must be terminated by `end` .

See Also

for , while , if , select

Name

equal — (=) assignment , comparison, equal sign

Description

Assignment:

The equal sign = is used to denote the assignment of value(s) to variable(s). The syntax can be :

- `a=expr` where `a` is a variable name and `expr` a scilab expression which evaluates to a single result.
- `[a,b,...]=expr` where `a`,`b`,`...` are variable names and `expr` a scilab expression which results in as many results as given variable names.

Comparison:

The equal sign = is also used in the comparison operators:

- `a==b`, denotes equality comparison between the values of the expressions `a` and `b`.
- `a~=b`, denotes inequality comparison between the values of the expressions `a` and `b`:
- `a<=b` and `a>=b` denotes ordering comparison between the values of the expressions `a` and `b`:

See comparison for semantic details.

Examples

```
a = sin(3.2)
M = [2.1,3.3,8.5;7.6,6.7,6.9;0,6.3,8.8];
[u,s] = schur(M)
[1:10] == 4
1~=2
```

See Also

less , great , boolean , isequal , comparison

Name

errcatch — error trapping

```
errcatch(n [,'action'] [,'option'])
errcatch()
```

Parameters

n
: integer

action, option
: strings

Description

`errcatch` gives an "action" (error-handler) to be performed when an error of type `n` occurs. `n` has the following meaning:

if `n>0`, `n` is the error number to trap

if `n<0` all errors are to be trapped

`action` is one of the following character strings:

"pause"
: a pause is executed when trapping the error. This option is useful for debugging purposes. Use `whereami()` to get information on the current context.

"continue"
: next instruction in the function or exec files is executed, current instruction is ignored. It is possible to check if an error has occured using the `iserror` function. Do not forget to clear the error using the `errclear` function as soon as possible This option is useful for error recovery. In many cases, usage of `errcatch(n,"continue",..)` can be replaced by the use of execstr function or try control structure.

"kill"
: default mode, all intermediate functions are killed, scilab goes back to the level 0 prompt.

"stop"
: interrupts the current Scilab session (useful when Scilab is called from an external program).

`option` is the character string `'nomessage'` for killing error message.

To set back default mode, enter `errcatch(-1,"kill")` or similarly `errcatch(-1)`.`errcatch()` is an obsolete equivalent of `errcatch(-1)`.

The `errcatch` actions apply to the current evalution context (function, exec, pause) and all the sub-levels. A second `errcatch` call in a sub-level hides the initial one for this sub-level. If a second `errcatch` call is made at the same level, the effect of the first one is removed.

When called in the context of a Scilab function or exec the `errcatch` is automatically reset when the function returns.

See Also

try , errclear , iserror , whereami , execstr

Name

errclear — error clearing

```
errclear([n])
```

Description

clears the action (error-handler) connected to error of type n.

If n is positive, it is the number of the cleared error ; otherwise all errors are cleared (default case)

See Also

errcatch , iserror , lasterror

Name

error — error messages

```
error(message [,n])
error(n)
error(n,pos)
```

Parameters

message
: a character string. The error message to be displayed.

n
: an integer. The number associated with the error message

pos
: an integer. a parameter for the error message

Description

`error` function allow to issue an error message and to handle the error. By defaut `error` stops the current execution and resume to the prompt level. This default can be changed using the `errcatch` or `execstr(...,'errcatch')` functions.

`error(message)` prints the character string contained in `message`. The number associated with the error message is 10000

`error(message,n)` prints the character string contained in `message`. The number associated with the error message is given by `n`. This number should be greater than 10000.

`error(n)` prints the predefined error message associated with the error number `n`.

Some predefined error messages require a parameter (see error_table). In this case the `pos` argument must be used `error(n,pos)` to give the parameter value. In the other cases the `pos` argument is ignored.

see error_table for a list of error messages and the associated error numbers.

Examples

```
error('my error message')
error(43)
error(52,3)
```

See Also

warning , errcatch , execstr , lasterror

Name

error_table — table of error messages

Description

This page gives the table of the predefined error messages, and their associated error number. Some of these error messages are used by Scilab itself for parser errors or specific builtin errors. Some others are of a more general use and can be used in Scilab functions. The starred one are those for which the syntax `error(n,pos)` is handled.

1 "Incorrect assignment."

2 "Invalid factor."

3 "Waiting for right parenthesis."

4 "Undefined variable: %s"

5 "Inconsistent column/row dimensions."

6 "Inconsistent row/column dimensions."

7 "Dot cannot be used as modifier for this operator."

8 "Inconsistent addition."

9 "Inconsistent subtraction."

10 "Inconsistent multiplication."

11 "Inconsistent right division."

12 "Inconsistent left division."

13 "Redefining permanent variable."

14 "Eye variable undefined in this context."

15 "Submatrix incorrectly defined."

16 "Incorrect command!"

17 "stack size exceeded! Use stacksize function to increase it."

18 "Too many variables!"

19 "Problem is singular."

* 20 "Wrong type for argument %d: Square matrix expected."

21 "Invalid index."

22 "Recursion problems. Sorry..."

23 "Matrix norms available are 1, 2, inf, and fro."

24 "Convergence problem..."

25 "Bad call to primitive: %s"

26 "Too complex recursion! (recursion tables are full)"

27 "Division by zero..."

28 "Empty function..."

29 "Matrix is not positive definite."

30 "Invalid exponent."

31 "Incorrect string."

32 "singularity of log or tan function"

33 "too many "":"""

34 "Incorrect control intruction syntax."

34 "Syntax error in a '%s' instruction." (if,while,select/case)

* 36 "Wrong input argument %d."

* 37 "Incorrect function at line %d."

38 "Wrong file name."

39 "Incorrect number of input arguments."

40 "Waiting for end of command."

41 "Incompatible output argument."

42 "Incompatible input argument."

43 "Not implemented in scilab..."

* 44 "Wrong argument %d."

* 45 "null matrix (argument # %d)."

46 "Incorrect syntax."

47 " end or else is missing..."

* 48 " input line longer than buffer size: %d"

49 "Incorrect file or format."

50 "subroutine not found: %s"

* 52 "Wrong type for argument %d: Real matrix expected."

* 53 "Wrong type for input argument %d: Real or complex matrix expected."

* 54 "Wrong type for input argument %d: Polynomial expected."

* 55 "Wrong type for argument %d: String expected."

* 56 "Wrong type for argument %d: List expected."

57 "Problem with comparison symbol..."

58 "Function has no input argument..."

59 "Function has no output."

60 "Wrong size for argument: Incompatible dimensions."

61 "Direct access : give format."

* 62 "End of file at line %d."

* 63 "%d graphic terminal?"

64 "Integration fails."

* 65 "%d: logical unit already used."

66 "No more logical units available!"

67 "Unknown file format."

68 "Fatal error!!! Your variables have been saved in the file : %s"

69 "Floating point exception."

70 "Too many arguments in fort (max 30)."

71 "This variable is not valid in fort."

72 "%s is not valid in this context."

73 "Error while linking."

74 "Leading coefficient is zero."

75 "Too high degree (max 100)."

* 76 "for x=val with type(val)=%d is not implemented in Scilab."

77 "%s: Wrong number of input arguments."

78 "%s: Wrong number of output arguments."

79 "Indexing not allowed for output arguments of resume."

80 "Incorrect function (argument n: %d)."

81 "%s: Wrong type for argument %d: Real or complex matrix expected."

82 "%s: Wrong type for argument %d: Real matrix expected."

83 "%s: Wrong type for argument %d: Real vector expected."

84 "%s: Wrong type for argument %d: Scalar expected."

85 "Host does not answer..."

86 "Uncontrollable system."

87 "Unobservable system."

88 "sfact: singular or assymetric problem."

* 89 "Wrong size for argument %d."

* 90 "Wrong type for argument %d: Transfer matrix expected."

* 91 "Wrong type for argument %d: In state space form expected."

* 92 "Wrong type for argument %d: Rational matrix expected."

* 93 "Wrong type for argument %d: In continuous time expected."

* 94 "Wrong type for argument %d: In discrete time expected."

* 95 "Wrong type for argument %d: SISO expected."

* 96 "time domain of argument %d is not defined."

* 97 "Wrong type for argument %d: A system in state space or transfer matrix form expected."

98 "Variable returned by scilab argument function is incorrect."

* 99 "Elements of %dth argument must be in increasing order."

* 100 "Elements of %dth argument are not in (strictly) decreasing order."

* 101 "Last element of %dth argument <> first."

102 "Variable or function %s are not in file."

103 "Variable %s is not a valid rational function."

104 "Variable %s is not a valid state space representation."

105 "Undefined fonction."

106 "Function name already used."

* 107 "Too many functions are defined (maximum #:%d)."

108 "Too complex for scilab, may be a too long control instruction."

109 "Too large, can't be displayed."

110 "%s was a function when compiled but is now a primitive!"

111 "Trying to re-define function %s."

112 "No more memory."

113 "Too large string."

114 "Too many linked routines."

115 "Stack problem detected within a loop."

* 116 "Wrong value for argument %d."

* 117 "List element number %d is Undefined."

* 118 "Wrong type for argument %d: Named variable not an expression expected."

120 "Indices for non-zero elements must be given by a 2 column matrix."

121 "Incompatible indices for non-zero elements."

* 122 "Logical unit number should be larger than %d."

123 "Function not bounded from below."

125 "Problem may be unbounded: too high distance between two consecutive iterations."

126 "Inconsistent constraints."

127 "No feasible solution."

128 "Degenerate starting point."

129 "No feasible point has been found."

130 "Optimization fails: back to initial point."

131 "optim: Stop requested by simulator (ind=0)"

132 "optim: Wrong input parameters."

133 "Too small memory."

134 "optim: Problem with initial constants in simul."

135 "optim : Bounds and initial guess are incompatible."

136 "optim : This method is NOT implemented."

137 "NO hot restart available in this method."

138 "optim: Incorrect stopping parameters."

139 "optim: Incorrect bounds."

140 "Variable : %s must be a list"

* 141 "Incorrect function (argument n: %d)."

* 142 "Hot restart: dimension of working table (argument n:%d)."

143 "optim:: df0 must be positive !"

144 "Undefined operation for the given operands."

201 "%s: Wrong type for argument %d: Real or complex matrix expected."

202 "%s: Wrong type for argument %d: Real matrix expected."

203 "%s: Wrong type for argument %d: Real vector expected."

* 204 "%s: Wrong type for argument %d: Scalar expected."

205 "%s: Wrong size for argument %d: (%d,%d) expected."

206 "%s: Wrong size for argument %d: %d expected."

207 "%s: Wrong type for argument %d: Matrix of strings expected."

208 "%s: Wrong type for argument %d: Boolean matrix expected."

209 "%s: Wrong type for argument %d: Matrix expected."

210 "%s: Wrong type for argument %d: List expected."

211 "%s: Wrong type for argument %d: Function or string (external function) expected."

212 "%s: Wrong type for argument %d: Polynomial expected."

213 "%s: Wrong type for argument %d: Working integer matrix expected."

214 "Argument %d of %s: wrong type argument, expecting a vector"

* 215 "%dth argument type must be boolean."

* 216 "Wrong type for argument %d: Boolean or scalar expected."

* 217 "Wrong type for argument %d: Sparse matrix of scalars expected."

* 218 "Wrong type for argument %d: Handle to sparse lu factors expected."

* 219 "Wrong type argument %d: Sparse or full scalar matrix expected."

220 "Null variable cannot be used here."

221 "A sparse matrix entry is defined with two differents values."

222 "%s not yet implemented for full input parameter."

223 "It is not possible to redefine the %s primitive this way (see clearfun)."

224 "Type data base is full."

225 "This data type is already defined."

226 "Inequality comparison with empty matrix."

227 "Missing index."

228 "reference to the cleared global variable %s."

229 "Operands of / and \\ operations must not contain NaN of Inf."

230 "semi def fails."

231 "Wrong type for first input argument: Single string expected."

232 "Entry name not found."

233 "Maximum number of dynamic interfaces reached."

234 "link: expecting more than one argument."

235 "link: problem with one of the entry point."

236 "link: the shared archive was not loaded."

237 "link: Only one entry point allowed on this operating system."

238 "link: First argument cannot be a number."

239 "You cannot link more functions, maxentry reached."

240 "File '%s' already exists or directory write access denied."

241 "File '%s' does not exist or read access denied."

242 "Binary direct acces files must be opened by 'file'."

243 "C file logical unit not allowed here."

244 "Fortran file logical unit not allowed here."

* 245 "No input file associated to logical unit %d."

246 "function not defined for given argument type(s)"

247 "Wrong value for argument %d: the lu handle is no more valid."

* 248 "Wrong value for argument %d: Valid variable name expected."

* 249 "Wrong value for argument %d: Empty string expected."

250 "Recursive extraction is not valid in this context."

251 "bvode: ipar dimensioned at least 11."

252 "bvode: ltol must be of size ipar(4)."

253 "bvode: fixpnt must be of size ipar(11)."

254 "bvode: ncomp < 20 requested."

255 "bvode: m must be of size ncomp."

256 "bvode: sum(m) must be less than 40."

257 "bvode: sum(m) must be less than 40."

258 "bvode: input data error."

259 "bvode: no. of subintervals exceeds storage."

260 "bvode: Th colocation matrix is singular."

261 "Interface property table is full."

* 262 "Too many global variables! , max number is %d."

263 "Error while writing in file,(disk full or deleted file."

* 264 "Wrong value for argument %d: Must not contain NaN or Inf."

265 "A and B must have equal number of rows."

266 "A and B must have equal number of columns."

267 "A and B must have equal dimensions."

* 268 "Invalid return value for function passed in arg %d."

* 269 "Wrong value for argument %d: eigenvalues must have negative real parts."

* 270 "Wrong value for argument %d: eigenvalues modulus must be less than one."

* 271 "Size varying argument a*eye(), (arg %d) not allowed here."

272 "endfunction is missing."

273 "Instruction left hand side: waiting for a dot or a left parenthesis."

274 "Instruction left hand side: waiting for a name."

275 "varargout keyword cannot be used here."

276 "Missing operator, comma, or semicolon."

277 "Too many commands defined."

278 "%s: Input arguments should have the same formal variable name."

See Also

warning, errcatch, execstr, lasterror

Name

evstr — evaluation of expressions

```
H=evstr(Z)
[H,ierr]=evstr(Z)
```

Parameters

Z
: matrix of character strings `M` or `list(M,Subexp)`

  M
  : matrix of character strings

  Subexp
  : vector of character strings

H
: matrix

ierr
: integer, error indicator

Description

Returns the result of the evaluation of the matrix of character strings `M`. Each element of the matrix must define a valid Scilab expression.

If the evaluation of `M` expression leads to an error, the single return value version, `H=evstr(M)`, raises the error as usual. The two return values version, `[H,ierr]=evstr(M)`, on the other hand, produces no error, but returns the error number in `ierr`.

If `Z` is a list, `Subexp` is a vector of character strings, that defines sub_expressions which are evaluated before evaluating `M`. These sub_expressions must be referred to as `%(k)` in `M`, where `k` is the sub-expression's index in `Subexp`.

evstr('a=1') is not valid (use `execstr` instead).

Examples

```
a=1; b=2; Z=['a','b'] ; evstr(Z)

a=1; b=2; Z=list(['%(1)','%(1)-%(2)'],['a+1','b+1']);
evstr(Z)
```

See Also

execstr

Name

exec — script file execution

```
exec(path [,mode])
exec(fun [,mode])
ierr=exec(path,'errcatch' [,mode])
ierr=exec(fun,'errcatch' [,mode])
```

Parameters

path
: a string, the path of the script file

mode
: an integer scalar, the execution mode (see below)

fun
: a scilab function

ierr
: integer, 0 or error number

Description

`exec(path [,mode])` executes sequentialy the scilab instructions contained in the file given by `path` with an optional execution mode `mode` .

The different cases for `mode` are :

0 : the default value

-1 : nothing is printed

1 : echo of each command line

2 : prompt `-->` is printed

3 : echoes + prompts

4 : stops before each prompt. Execution resumes after a carriage return.

7 : stops + prompts + echoes : useful mode for demos.

`exec(fun [,mode])` executes function `fun` as a script: no input nor output argument nor specific variable environment. This form is more efficient, because script code may be pre-compiled (see getf, comp). This method for script evaluation allows to store scripts as function in libraries.

If an error is encountered while executing, if 'errcatch' flag is present `exec` issues no error message, aborts execution of the instructions and resumes with `ierr` equal to the error number. If 'errcatch' flag is not present, standard error handling works.

Remark

`exec` files may now be used to define functions using the inline function definition syntax (see function).

Examples

```
```

```
// create a script file
mputl('a=1;b=2',TMPDIR+'/myscript')
// execute it
exec(TMPDIR+'/myscript')
whos -name "a "

// create a function
deff('y=foo(x)','a=x+1;y=a^2')
clear a b
// call the function
foo(1)
// a is a variable created in the environment of the function foo
//    it is destroyed when foo returns
whos -name "a "


x=1 //create x to make it known by the script foo
exec(foo)
// a and y are created in the current environment
whos -name "a "
```

See Also

getf , execstr , evstr , comp , mode , chdir , getcwd

Name

exists — checks variable existence

```
exists(name [,where])
```

Parameters

name
: a character string

where
: an optional character with possible values: `'l'` (local), `'n'` (nolocal) and `'a'` (all). The default value is `'all'`.

Description

`exists(name)` returns `1` if the variable named `name` exists and `0` otherwise.

Caveats: a function which uses `exists` may return a result which depends on the environment!

`exists(name,'local')` returns `1` if the variable named `name` exists in the environment of the current function and `0` otherwise.

`exists(name,'nolocal')` returns `1` if the variable named `name` exists in any level of the calling environment (including the Scilab shell main level) of the current function and `0` otherwise.

Warning: the exists function does not check if a variable exists in the global namespace.

Examples

```
deff('foo(x)',..
['disp([exists(''a12''),exists(''a12'',''local'')])'
 'disp([exists(''x''),exists(''x'',''local'')])'])
foo(1)
a12=[];foo(1)

function level1()
  function level2()
    disp(exists("a","all"));
    disp(exists("a","local"));
    disp(exists("a","nolocal"));
  endfunction
  level2()
endfunction
function go()
  a=1;
  level1()
endfunction
go()
```

See Also

isdef, isglobal, whereis, type, typeof, macrovar

Name

exit — Ends the current Scilab session

Description

Ends the current Scilab session.

See Also

quit , abort , break , return , resume

Name

external — Scilab Object, external function or routine

Description

External function or routine for use with specific commands.

An "external" is a function or routine which is used as an argument of some high-level primitives (such as `ode`, `optim`, `schur`...).

The calling sequence of the external (function or routine) is imposed by the high-level primitive which sets the arguments of the external.

For example the external function `costfunc` is an argument of the `optim` primitive. Its calling sequence must be: `[f,g,ind]=costfunc(x,ind)` and `optim` (the high-level optimization primitive) is invoked as follows:

```
optim(costfunc,...)
```

Here `costfunc` (the cost function to be minimized by the primitive `optim`) evaluates `f=f(x)` and `g=` gradient of `f` at `x` (`ind` is an integer. Its use is precised in the `optim` help).

If other values are needed by the external function these variables can be defined in its environment. Also, they can be put in a list. For example,the external function

```
[f,g,ind]=costfunc(x,ind,a,b,c)
```

is valid for `optim` if the external is `list(costfunc,a,b,c)` and the call to `optim` is then:

```
optim(list(costfunc,a1,b1,c1),....
```

An external can also be a Fortran or C routine : this is convenient to speed up the computations.

The name of the routine is given to the high-level primitive as a character string. The calling sequence of the routine is also imposed. Examples are given in the `routines/default` directory (see the `README` file).

External Fortran or C routines can also be dynamically linked (see `link`)

See Also

ode , optim , impl , dassl , intg , schur , gschur

Name

extraction — matrix and list entry extraction

```
x(i)
x(i,j)
x(i,j,k,..)
[...]=l(i)
[...]=l(k1)...(kn)(i) or [...]=l(list(k1,...,kn,i))
l(k1)...(kn)(i,j) or l(list(k1,...,kn,list(i,j))
```

Parameters

x
: matrix of any possible types

l
: list variable

i,j, k
: indices

k1,...kn
: indices

Description

MATRIX CASE
: `i`, `j`, `k`,.. can be:

  a real scalar or a vector or a matrix with positive elements.

  - `r=x(i,j)` builds the matrix `r` such as `r(l,k)=x(int(i(l)),int(j(k)))` for `l` from 1 to `size(i,'*')` and `k` from 1 to `size(j,'*')`.

    `i` (`j`) Maximum value must be less or equal to `size(x,1)` (`size(x,2)`).

  - `r=x(i)` with `x` a 1x1 matrix builds the matrix `r` such as `r(l,k)=x(int(i(l)),int(i(k)))` for `l` from 1 to `size(i,1)` and `k` from 1 to `size(i,2)`.

    Note that in this case index `i` is valid only if all its entries are equal to one.

  - `r=x(i)` with `x` a row vector builds the row vector `r` such as `r(l)=x(int(i(l)))` for `l` from 1 to `size(i,'*')i` Maximum value must be less or equal to `size(x,'*')`.

  - `r=x(i)` with `x` a matrix with one or more columns builds the column vector `r` such as `r(l)` (`l` from 1 to `size(i,'*')`) contains the `int(i(l))` entry of the column vector formed by the concatenation of the `x`'s columns.

    `i` Maximum value must be less or equal to `size(x,'*')`.

  the symbol `:`
  : which stands for "all elements".

    - `r=x(i,:)` builds the matrix `r` such as `r(l,k)=x(int(i(l)),k))` for `l` from 1 to `size(i,'*')` and `k` from 1 to `size(x,2)`

- `r=x(:,j)` builds the matrix `r` such as `r(l,k)=x(l,int(j(k)))` for `l` from 1 to `size(r,1)` and `k` from 1 to `size(j,'*')`.
- `r=x(:)` builds the column vector `r` formed by the column concatenations of `x` columns. It is equivalent to `matrix(x,size(x,'*'),1)`.

a vector of boolean
: If an index (`i` or `j`) is a vector of booleans it is interpreted as `find(i)` or respectively `find(j)`

a polynomial
: If an index (`i` or `j`) is a vector of polynomials or implicit polynomial vector it is interpreted as `horner(i,m)` or respectively `horner(j,n)` where `m` and `n` are associated `x` dimensions. Even if this feature works for all polynomials, it is recommended to use polynomials in `$` for readability.

For matrices with more than 2 dimensions (see:hypermatrices) the dimensionality is automatically reduced when right-most dimensions are equal to 1.

LIST OR TLIST CASE
: If they are present the `ki` give the path to a sub-list entry of `l` data structure. They allow a recursive extraction without intermediate copies. The instructions

```
[...]=l(k1)...(kn)(i)
```

and

```
[...]=l(list(k1,...,kn,i))
```

are interpreted as:

`lk1 = l(k1) .. = .. lkn = lkn-1(kn) [...] = lkn(i)` And the `l(k1)...(kn)(i,j)` and

`l(list(k1,...,kn,list(i,j))` instructions are interpreted as: `lk1 = l(k1) .. = .. lkn = lkn-1(kn) lkn(i,j)` `i` and `j`, can be: When path points on more than one list component the instruction must have as many left hand side arguments as selected components. But if the extraction syntax is used within a function input calling sequence each returned list component is added to the function calling sequence.

Note that, `l(list())` is the same as `l`.

i and j may be :

real scalar or vector or matrix with positive elements.
: `[r1,...rn]=l(i)` extracts the `i(k)` elements from the list l and store them in `rk` variables for `k` from 1 to `size(i,'*')`

the symbol `:`
: which stands for "all elements".

a vector of booleans.
: If `i` is a vector of booleans it is interpreted as `find(i)`.

a polynomial.
: If `i` is a vector of polynomials or implicit polynomial vector it is interpreted as `horner(i,m)` where `m=size(l)`. Even if this feature works for all polynomials, it is recommended to use polynomials in `$` for readability.

k1,..kn may be :

real positive scalar,

a polynomial,
: interpreted as `horner(ki,m)` where `m` is the corresponding sub-list size.

a character string
: associated with a sub-list entry name.

Remarks

For soft coded matrix types such as rational functions and state space linear systems, `x(i)` syntax may not be used for vector element extraction due to confusion with list element extraction. `x(1,j)` or `x(i,1)` syntax must be used.

Examples

```
// MATRIX CASE
a=[1 2 3;4 5 6]
a(1,2)
a([1 1],2)
a(:,1)
a(:,3:-1:1)
a(1)
a(6)
a(:)
a([%t %f %f %t])
a([%t %f],[2 3])
a(1:2,$-1)
a($:-1:1,2)
a($)
//
x='test'
x([1 1;1 1;1 1])
//
b=[1/%s,(%s+1)/(%s-1)]
b(1,1)
b(1,$)
b(2) // the numerator
// LIST OR TLIST CASE
l=list(1,'qwerw',%s)
l(1)
[a,b]=l([3 2])
l($)
x=tlist(l(2:3)) //form a tlist with the last 2 components of l
//
dts=list(1,tlist(['x';'a';'b'],10,[2 3]));
dts(2)('a')
dts(2)('b')(1,2)
[a,b]=dts(2)(['a','b'])
```

See Also

find, horner, parents

Name

for — language keyword for loops

Description

Used to define loops. Its syntax is: `for variable=expression ,instruction, .. ,instruction,end`

`for variable=expression do instruction, ,instruction,end`

If `expression` is a matrix or a row vector, `variable` takes as values the values of each column of the matrix.

A particular case uses the colon operator to create regularly spaced row vectors, and ressemble to traditional for loop forms : `for variable=n1:step:n2, ...,end`

If `expression` is a list `variable` takes as values the successive entries of the list.

Warning: the number of characters used to define the body of any conditional instruction (if while for or select/case) must be limited to 16k.

Examples

```
// "traditional" for loops
n=5;
for i = 1:n, for j = 1:n, a(i,j) = 1/(i+j-1);end;end
for j = 2:n-1, a(j,j) = j; end; a
for j= 4:-1:1, disp(j),end // decreasing loop

//loop on matrix columns
for  e=eye(3,3),e,end
for v=a, write(6,v),end
for j=1:n,v=a(:,j), write(6,v),end

//loop on list entries
for l=list(1,2,'example'); l,end
```

See Also

while , end , do

Name

format — number printing and display format

```
format([type],[long])
v = format()
format(m)
```

Parameters

type
: character string

long
: integer (max number of digits (default 10))

v
: a vector for the current format v(1) type format : 0 for 'e' and 1 for 'v' v(2) number of digits

m
: a vector to set new format

  m(1) number of digits

  m(2) type format : 0 for 'e' and 1 for 'v'

Description

Sets the current printing format with the parameter `type` ; it is one of the following :

"v"
: for a variable format (default)

"e"
: for the e-format.

`long` defines the max number of digits (default 10). `format()` returns a vector for the current format: first component is the type of format (1 if `v` ; 0 if `e`); second component is the number of digits.

In the old Scilab versions, in "variable format" mode, vectors entries which are less than %eps times the maximum absolute value of the entries were displayed as "0". It is no more the case, the clean function can be used to set neglitible entries to zeros.

Examples

```
x=rand(1,5);
format('v',10);x
format(20);x
format('e',10);x
format(20);x

x=[100 %eps];
format('e',10);x
format('v',10);x

format("v")
```

See Also

write, disp, print

Name

funcprot — switch scilab functions protection mode

```
prot=funcprot()
funcprot(prot)
```

Parameters

prot
: integer with possible values 0,1,2

Description

Scilab functions are variable, funcprot allows the user to specify what scilab do when such variables are redefined.

- If prot==0 nothing special is done
- If prot==1 scilab issues a warning message when a function is redefined (default mode)
- If prot==2 scilab issues an error when a function is redefined

Examples

```
funcprot(1)
deff('[x]=foo(a)','x=a')
deff('[x]=foo(a)','x=a+1')
foo=33
funcprot(0)
deff('[x]=foo(a)','x=a')
deff('[x]=foo(a)','x=a+1')
foo=33
```

Name

funptr — coding of primitives (wizard stuff)

```
[numptr] = funptr(name)
```

Parameters

name
: a string, the name of a primitive

numptr
: the internal routine number of the primitive

Description

Utility function (for experts only) to get the internal routine number `numptr` of the primitive `'name'`. `numptr` is formed from the interface number `fun` and the routine number `fin` of the primitive in its interface by `numptr = 100*fun + fin` (fin < 100). From `numptr` you can get the interface number `fun = floor(numptr/100)` which may be useful to link a dynamical interface with arguments passed by reference (see example section).

Examples

```
// Suppose you want to load some codes via the dynamic
// loading facilities offers by addinter. By default
// arguments are passed by values but if you want to
// pass them by reference you can do the following
// (name being the scilab name of one of the interfaced
// routines) :
//
// addinter(files,spnames,fcts)  // args passed by values
// num_interface = floor(funptr(name)/100)
// intppty(num_interface)  // args now passed by reference
//
// Note that if you enter the following
//
// intppty()
//
// you will see all the interfaces working by reference
```

See Also

clearfun , newfun , intppty , addinter

Name

getdebuginfo — get information about Scilab to debug

```
getdebuginfo()
dynamic_info = getdebuginfo();
[dynamic_info,static_info] = getdebuginfo();
```

Description

getdebuginfo get information about Scilab to debug.

dynamic_info = getdebuginfo(); returns information about your system.

[dynamic_info,static_info] = getdebuginfo(); returns information about your system and about Scilab.

See Also

getversion

Authors

A.C

Name

getmd5 — get md5 checksum

```
res=getmd5(filename)
res=getmd5(ParamString,'string')
```

Parameters

res
: md5 result (string)

filename
: filename (string or matrix of strings)

ParamString
: string or matrix of strings

Description

`getmd5(...)` get md5 checksum of a file or a string.

Examples

```


getmd5('hello world','string')
getmd5(['hello' 'world'],'string')
getmd5(['hello' ; 'world'],'string')

getmd5( SCI+'/modules/core/etc/core.start' )
getmd5( SCI+'/modules/core/etc/'+['core.start' 'core.quit'])


```

Authors

A.C.

Name

getmemory — returns free and total system memory

```
[free, total]=getmemory()
```

Description

`getmemory()` returns free system memory (kilo-octets).

`[free, total]=getmemory()` returns free and total system memory (kilo-octets).

Authors

A.C

Name

getmodules — returns list of modules installed in Scilab

```
res=getmodules()
```

Parameters

res
: a string matrix

Description

Returns list of modules installed in Scilab.

See Also

with_module

Authors

A.c

Name

getos — return Operating System name and version

```
OS=getos()
[OS,Version]=getos()
```

Description

`getos` return Operating System name and version

Examples

```
OS=getos()
[OS,version]=getos()
```

Authors

A.C

Name

getscilabmode — returns scilab mode

```
mode = getscilabmode()
```

Description

returns scilab mode. 4 modes are possible : STD , API , NW , NWNI .

`API`Scilab is launch as an API.

`STD`The standard Scilab (gui, plot ...)

`NW`Scilab in command line with the plots.

`NWNI`Scilab in command line without any graphics.

Examples

```
getscilabmode()
```

See Also

scilab

Name

getshell — returns current command interpreter.

```
getshell()
```

Description

`getshell`returns current command interpreter.

Examples

```
   getshell()
```

Authors

Allan CORNET

Name

getvariablesonstack — get variable names on stack of scilab

```
s=getvariablesonstack()
s=getvariablesonstack('local')
s=getvariablesonstack('global')
```

Parameters

s
: a string matrix

Description

return in `S`variable names on scilab stack.

getvariablesonstack('local') returns local variables on scilab stack.

getvariablesonstack('global') returns global variables on scilab stack.

Variables are sorted by alphabetical order.

Examples

```
getvariablesonstack()
  getvariablesonstack('local')
  getvariablesonstack('global')
```

See Also

who

Name

getversion — get scilab and modules version information

```
version=getversion()
[version,opts]=getversion()
ver=getversion('scilab')
versioninfo=getversion('scilab','string_info')
ver=getversion('<module>')
versioninfo=getversion('<module>','string_info')
```

Parameters

version
: a string

versioninfo
: a string about version

ver
: a integer vector

  ver(1) Major version

  ver(2) Minor version

  ver(3) Maintenance version

  ver(4) GIT timestamp

opts
: a vector of string with four entries :`[compiler,pvm,tk,modelicac]`

Description

return in `version` the Scilab version name and in `opts` build options which can be used to determine if scilab has been build with pvm, tk or modelicac.

Examples

```
getversion()
[version,opts]=getversion()
ver=getversion('scilab')
verstr=getversion('scilab','string_info')
ver=getversion('overloading')
verstr=getversion('overloading','string_info')
```

See Also

getmodules

Name

global — Define global variable

```
global('nam1',...,'namn')
global nam1 ... namn
```

Parameters

nam1,..., namn
: valid variable names

Description

Ordinarily, each Scilab function, has its own local variables and can "read" all variables created in the base workspace or by the calling functions. The `global` keyword allow to make variables read/write across functions. Any assignment to that variable, in any function, is available to all the other functions declaring it `global`.

If the global variable doesn't exist the first time you issue the `global` statement, it will be initialized to the empty matrix.

Examples

```
//first: calling environnment and a function share a variable
global a
a=1
deff('y=f1(x)','global a,a=x^2,y=a^2')
f1(2)
a
//second: three functions share variables
deff('initdata()','global A C ;A=10,C=30')
deff('letsgo()','global A C ;disp(A) ;C=70')
deff('letsgo1()','global C ;disp(C)')
initdata()
letsgo()
letsgo1()
```

See Also

who , isglobal , clearglobal , gstacksize , resume

Name

gstacksize — set/get scilab global stack size

```
gstacksize(n)
gstacksize('max')
gstacksize('min')
sz=gstacksize()
```

Parameters

n
: integer, the required global stack size given in number of double precision words

sz
: 2-vector [total used]

Description

Scilab stores gobal variables in a stack

`gstacksize(n)` allows the user to increase or decrease the size of this stack. The maximum allowed size depends on the amount of free memory and swap space available at the time. Note that Scilab can increase automatically the global stacksize when needed

`sz=gstacksize()` returns a 2-vector which contains the current total and used global stack size.

`gstacksize('max')` allows the user to increase the size of this global stack to the maximum.

`gstacksize('min')` allows the user to decrease the size of this global stack to the minimum.

See Also

who , stacksize

Name

hat — (^) exponentiation

```
A^b
```

Description

Exponentiation of matrices or vectors by a constant vector.

If A is a vector or a rectangular matrix the exponentiation is done element-wise, with the usual meaning.

For square A matrices the exponentiation is done in the matrix sense.

For boolean, polynomial and rational matrices, the exponent must be an integer

Remarks

`123.^b` is interpreted as `(123).^b`. In such cases dot is part of the operator, not of the number.

For two real or complex numbers `x1` et `x2` the value of `x1^x2` is the "principal value" determined by `x1^x2 = exp(x2*log(x1))`.

Examples

```
2^4
(-0.5)^(1/3)
[1 2;2 4]^(1+%i)
s=poly(0,"s");
[1 2 s]^4
[s 1;1  s]^(-1)
```

See Also

exp , inv

Name

ieee — set floating point exception mode

```
mod=ieee()
ieee(mod)
```

Parameters

mod
: integer scalar whose possible values are 0,1,or 2

Description

`ieee()` returns the current floating point exception mode.

0
: floating point exception produces an error

1
: floating point exception produces a warning

2
: floating point exception produces Inf or Nan

`ieee(mod)` sets the current floating point exception mode.

The initial mode value is 0.

Remarks

Floating point exception arising inside some library algorithms are not yet handled by `ieee` modes.

Examples

```
ieee(1);1/0
ieee(2);1/0,log(0)
```

See Also

errcatch

Name

if then else — conditional execution

```
if expr1 then statements
elseif expri then statements
....
else statements
end
```

Description

The `if` statement evaluates a logical expression and executes a group of statements when the expression is true.

The `expri` are expressions with numeric or boolean values. If `expri` are matrix valued the condition is true only if all matrix entries are true or different from zero.

The optional `elseif` and `else` provide for the execution of alternate groups of statements. An `end` keyword, which matches the `if`, terminates the last group of statements. The line structure given above is not significant, the only constraint is that each `then` keyword must be on the same line as its corresponding `if` or `elseif` keyword.

The keyword `then` can be replaced by a carriage return or a comma.

Warning: the number of characters used to define the body of any conditionnal instruction (if while for or select/case) must be limited to 16k.

Examples

```
i=2
for j = 1:3,
   if i == j then
     a(i,j) = 2;
   elseif abs(i-j) == 1 then
     a(i,j) = -1;
   else a(i,j) = 0;
   end,
end
```

See Also

try , while , select , boolean , end , then , else

Name

insertion — partial variable assignation or modification
assignation — partial variable assignation

```
x(i,j)=a
x(i)=a
l(i)=a
l(k1)...(kn)(i)=a or l(list(k1,...,kn,i))=a
l(k1)...(kn)(i,j)=a or l(list(k1,...,kn,list(i,j))=a
```

Parameters

x
: matrix of any kind (constant, sparse, polynomial,...)

l
: list

i,j
: indices

k1,...kn
: indices with integer value

a
: new entry value

Description

MATRIX CASE
: if `x` is a matrix the indices `i` and `j`, may be:

  Real scalars or vectors or matrices
  : In this case the values given as indices should be positive and it is only their integer part which taken into account.

    - if `a` is a matrix with dimensions `(size(i,'*'),size(j,'*'))`, `x(i,j)=a` returns a new `x` matrix such as `x(int(i(l)),int(j(k)))=a(l,k)` for `l` from 1 to `size(i,'*')` and `k` from 1 to `size(j,'*')`, other initial entries of `x` are unchanged.
    - if `a` is a scalar `x(i,j)=a` returns a new `x` matrix such as `x(int(i(l)),int(j(k)))=a` for `l` from 1 to `size(i,'*')` and `k` from 1 to `size(j,'*')`, other initial entries of `x` are unchanged.
    - If `i` or `j` maximum value exceed corresponding `x` matrix dimension, array `x` is previously extended to the required dimensions with zeros entries for standard matrices, 0 length character string for string matrices and false values for boolean matrices.
    - `x(i,j)=[]` kills rows specified by `i` if `j` matches all columns of `x` or kills columns specified by `j` if `i` matches all rows of `x`. In other cases `x(i,j)=[]` produce an error.
    - `x(i)=a` with `a` a vector returns a new `x` matrix such as `x(int(i(l)))=a(l)` for `l` from 1 to `size(i,'*')` , other initial entries of `x` are unchanged.
    - `x(i)=a` with `a` a scalar returns a new `x` matrix such as `x(int(i(l)))=a` for `l` from 1 to `size(i,'*')` , other initial entries of `x` are unchanged.

      If `i` maximum value exceed `size(x,1)`, `x` is previously extended to the required dimension with zeros entries for standard matrices, 0 length character string for string matrices and false values for boolean matrices.

if
`x` is a 1x1

matrix `a` may be a row (respectively a column) vector with dimension `size(i,'*')`. Resulting `x` matrix is a row (respectively a column) vector

if
`x` is a row

vector `a` must be a row vector with dimension `size(i,'*')`

if
`x` is a column

vector `a` must be a column vector with dimension `size(i,'*')`

if
`x` is a general

matrix `a` must be a row or column vector with dimension `size(i,'*')` and `i` maximum value cannot exceed `size(x,'*')`,

- `x(i)=[]` kills entries specified by `i`.

The : symbol
the `:` symbol stands for "all elements".

- `x(i,:)=a` is interpreted as `x(i,1:size(x,2))=a`
- `x(:,j)=a` is interpreted as `x(1:size(x,1),j)=a`
- `x(:)=a` returns in `x` the `a` matrix reshaped according to x dimensions. `size(x,'*')` must be equal to `size(a,'*')`

Vectors of boolean
If an index (`i` or `j`)is a vector of booleans it is interpreted as `find(i)` or respectively `find(j)`

Polynomials
If an index (`i` or `j`)is a vector of polynomials or implicit polynomial vector it is interpreted as `horner(i,m)` or respectively `horner(j,n)` where `m` and `n` are associated `x` dimensions. Even if this feature works for all polynomials, it is recommended to use polynomials in `$` for readability.

LIST OR TLIST CASE

- If they are present the `ki` give the path to a sub-list entry of `l` data structure. They allow a recursive insertion without intermediate copies. The `l(k1)...(kn)(i)=a` and `l(list(k1,...,kn,i)=a)` instructions are interpreted as:

```
lk1 = l(k1)  .. = ..

lkn = lkn-1(kn)  lkn(i) = a

lkn-1(kn) = lkn  .. = ..  l(k1) = lk1
```

And the `l(k1)...(kn)(i,j)=a` and `l(list(k1,...,kn,list(i,j))=a` instructions are interpreted as:

```
lk1 = l(k1)  .. = ..
```

```
lkn = lkn-1(kn) lkn(i,j) = a

lkn-1(kn) = lkn  .. = ..  l(k1)= lk1
```

- `i` may be :
  - a real non negative scalar. `l(0)=a` adds an entry on the "left" of the list `l(i)=a` sets the `i` entry of the list `l` to `a`. if `i>size(l)`, `l` is previously extended with zero length entries (undefined). `l(i)=null()` suppress the `i`th list entry.
  - a polynomial. If `i` is a polynomial it is interpreted as `horner(i,m)` where `m=size(l)`. Even if this feature works for all polynomials, it is recommended to use polynomials in `$` for readability.
- `k1,..kn` may be :
  - real positive scalar.
  - a polynomial,interpreted as `horner(ki,m)` where `m` is the corresponding sub-list size.
  - a character string associated with a sub-list entry name.

Remarks

For soft coded matrix types such as rational functions and state space linear systems, `x(i)` syntax may not be used for vector entry insertion due to confusion with list entry insertion. `x(1,j)` or `x(i,1)` syntax must be used.

Examples

```
// MATRIX CASE
a=[1 2 3;4 5 6]
a(1,2)=10
a([1 1],2)=[-1;-2]
a(:,1)=[8;5]
a(1,3:-1:1)=[77 44 99]
a(1)=%s
a(6)=%s+1
a(:)=1:6
a([%t %f],1)=33
a(1:2,$-1)=[2;4]
a($:-1:1,1)=[8;7]
a($)=123
//
x='test'
x([4 5])=['4','5']
//
b=[1/%s,(%s+1)/(%s-1)]
b(1,1)=0
b(1,$)=b(1,$)+1
b(2)=[1 2] // the numerator
// LIST OR TLIST CASE
l=list(1,'qwerw',%s)
l(1)='Changed'
l(0)='Added'
l(6)=['one more';'added']
```

```
//
//
dts=list(1,tlist(['x';'a';'b'],10,[2 3]));
dts(2).a=33
dts(2)('b')(1,2)=-100
```

See Also

find , horner , parents , extraction

Name

intppty — set interface argument passing properties

```
funs=intppty()
intppty(fun)
```

Parameters

fun
: integer an interface number (see funptr)

funs
: integer vector, vector of interface number (see funptr)

Description

The interface programs may be written in 2 different ways for the mode of function argument passing.

In the first and default way, the arguments are passed by value. With the following syntax:

```
foo(A,1+2)
```

the argument associated with `A` will be passed by value (a copy of `A` is made before `foo` is called, and the argument associated with `1+2` will be passed by value.

In the second way arguments may be passed be reference if there are "named arguments" (no copy of the variable value is done). `intppty(fun)` with `fun>0` tells Scilab that the interface with number `fun` can handle arguments passed by reference. With the following syntax:

```
foo(A,1+2)
```

the argument associated with `A` will be passed by reference, and the argument associated with `1+2` will be passed by value.

Warning, declaring that the interface with number `fun` can handle arguments passed by reference if it is not the case should produce unpredictable results.

`intppty(fun)` with `fun<0` suppress this property for the interface `-fun`.

`intppty()` returns the vector of interfaces which handle arguments passed by reference.

This function may be useful for dynamically loaded interface (see addinter).

See Also

funptr , addinter

Name

inv_coeff — build a polynomial matrix from its coefficients

```
[P]=inv_coeff(C,[,d,[name])
```

Parameters

C
: big matrix of the coefficients

d
: Polynomial matrix degree. optional parameter with default value `d=-1+size(C,'c')/size(C,'r')`

name
: string giving the polynomial variable name (default value `'x'`).

Description

`P=inv_coeff(Mp,k)`, when `k` is compatible with `Mp` size, returns a polynomial matrix of degree k. `C=[C0,C1,...,Ck]` and `P= C0 + C1*x +... +Ck*x^k`.

Examples

```
A=int(10*rand(2,6))
// Building a degree 1 polynomial matrix
P=inv_coeff(A,1)
norm(coeff(P)-A)
// Using default value for degree
P1=inv_coeff(A)
norm(coeff(P1)-A)
```

See Also

poly , degree , coeff

Name

iserror — error occurence test

```
iserror([n])
```

Description

tests if error number `n` has occurred (after a call to `errcatch`). `iserror` returns `1` if the error occurred a nd `0` otherwise

`n>0` is the error number ; all errors are tested with `n<0`.

See Also

error , errcatch

Name

isglobal — check if a variable is global

```
t=isglobal(x)
```

Parameters

x
: any variable

t
: a boolean

Description

`isglobal(x)` returns true if x has been declared to be a global variable and false otherwise.

Examples

```
isglobal(1)
global a
isglobal(a)
```

See Also

global , clearglobal , who

Name

lasterror — get last recorded error message

```
str=lasterror( [opt] )
[str,n]=lasterror([opt])
[str,n,line,func]=lasterror([opt])
```

Parameters

str
: vector of character strings or an empty matrix. The last recorded error message.

n
: integer, 0 or the last recorded error number.

line
: integer, 0 or the last recorded function line number.

func
: string,the last recorded function name

opt
: boolean, if %t recorded message is cleared. Default is %t.

Description

Each time an error occurs, the Scilab error handler records it in internal tables (only the last one is retained). The `lasterror` function allows to get the message, the error number, the current function (if any) and the current line of the current function out of these tables.

The line number reported is the *physical* line number where the last error occurred. Note that Scilab versions before 5.0 used to report the *logical* line number of the last error. The difference does matter only if the function in error includes continued lines before the point where the error happened.

This function is useful while using `errcatch` or `execstr`.

The recorded error message may be retained for a further call to `lasterror` using `lasterror(%f)`.

Examples

```
ierr=execstr('a=zzzzzzz','errcatch')
if ierr>0 then disp(lasterror()),end
```

See Also

errcatch, execstr, error, errclear, edit_error

Name

left — ([) left bracket

```
[a11,a12,...;a21,a22,...;...]
[s1,s2,...]=func(...)
```

Parameters

a11,a12,...
: matrix of any compatibles types with compatibles dimensions s1,s2,... : any possible variable name

Description

Left and right brackets are used for vector and matrix concatenation. These symbols are also used to denote a multiple left-hand-side for a function call

Inside concatenation brackets blank or comma characters mean "column concatenation", semicolon and carriage-return mean "row concatenation".

Note : to avoid confusions it is safer to use comma instead of blank to separate columns.

Within multiple lhs brackets variable names must be separated by comma.

Examples

```
[6.9,9.64; sqrt(-1) 0]
[1 +%i  2 -%i  3]
[]
['this is';'a string';'vector']
s=poly(0,'s');[1/s,2/s]
[tf2ss(1/s),tf2ss(2/s)]

[u,s]=schur(rand(3,3))
```

See Also

comma , semicolon

Name

less — (<) lower than comparison
great — (<) greater than comparison

Description

logical comparison symbol

<>
: means "different" (same as ~=)

<
: means "lower than"

>
: means "larger than"

<=
: means lower than or equal to.

>=
: means larger than or equal to

See Also

if , comparison , equal

Name

librarieslist — get scilab libraries

```
s=librarieslist()
```

Parameters

s
: a string matrix

Description

return in `s` all libraries on scilab stack.

Examples

```
librarieslist()
```

See Also

libraryinfo

Name

libraryinfo — get macros and path of a scilab library

```
macros = libraryinfo(libraryname)
[macros,path] = libraryinfo(libraryname)
```

Parameters

macros
: a string matrix (all main functions of the library)

path
: a string (path of library)

libraryname
: a string (library name)

Description

get functions names and path of a scilab library.The function names returned correspond to those which correspond to the associated .sci or .bin file names. The other ones are subsidiary functions.

Examples

```
[m,p]=libraryinfo('corelib')
```

See Also

librarieslist

Name

macr2lst — function to list conversion

```
[txt]=macr2lst(function-name)
```

Description

This primitive converts a compiled Scilab function `function-name` into a list which codes the internal representation of the function (reverse polish notation).

The first entry of the list is the function name, the second and third are respectively the vectors of left hand side variables and right hand side variables names. The following entries are either basic operation records either lists with contains the hierachical control structures like if , for, ...

Basic operation records are described by a character string vector whose first element represents the opcode.

op codes	meaning	parameters
"0"	ignored opcode	none
"1"	No more used	
"2"	variable or function reference	variable name, #rhs, #lhs
"3"	put a string in the stack	the string
"4"	put am empty matrix in the stack	none
"5"	apply an operation	operation code, #rhs,#lhs
"6"	put a number in the stack	the number
"12"	pause command	none
"13"	break command	none
"14"	abort command	none
"15"	end of line mark	none
"17"	quit command	none
"18"	named variable	variable name
"19"	create recursive index structure	path length, number of final indices
"20"	function call	function name, #rhs, #lhs
"23"	create variable from name	variable name
"24"	put a variable with type 0 in the stack	none
"25"	profile record	number of call, time spend
"26"	put a vector of strings in the stack	#rows, #columns, element sequence
"27"	put a builtin reference in the stack	interface number, position in interface, function name
"28"	continue command	none
"29"	assignment	#lhs, display mode, (variable name, #rhs)*
"30"	logical expression short circuit	type, jump size
"31"	comment	the comment

"99"	return command	none
> "100"	builtin call (obsolete)	100*fun, #rhs, #lhs, fin

The fun2string function can be used to generate the intial code.

Examples

```
//DISPLAY
function y=foo(x,flag)
  if flag then
    y=sin(x)
  else
    y=cos(x)
  end
endfunction
L=macr2lst(foo)
fun2string(L)
```

See Also

macrovar , fun2string , macr2tree , tree2code

Name

macr2tree — function to tree conversion

```
t=macr2tree(function-name)
```

Parameters

function-name
: a Scilab macro

t
: a Scilab "tree"

Description

This primitive converts a compiled Scilab function `function-name` into a tree (imbricated tlists) which codes the internal representation of the function. For use with `tree2code`.

Examples

```
tree=macr2tree(cosh);
txt=tree2code(tree,%T);
write(%io(2),txt,'(a)');
```

See Also

tree2code

Authors

V.C.

Name

matrices — Scilab object, matrices in Scilab

Description

Matrices are basic objects defined in Scilab. They can be defined as follows:

```
E=[e11,e12,...,e1n;
   e21,e22,...,e2n;
        ....
   em1,em2,...,emn];
```

Entries `eij` can be real or complex numbers, polynomials, rationals, strings, booleans.

Vectors are seen as matrices with one row or one column.

Examples

```
E=[1,2;3,4]
E=[%T,%F;1==1,1~=1]
s=poly(0,'s');E=[s,s^2;1,1+s]
E=[1/s,0;s,1/(s+1)]
E=['A11','A12';'A21','A22']
```

See Also

poly , string , boolean , rational , empty , hypermatrices

Name

matrix — reshape a vector or a matrix to a different size matrix

```
y=matrix(v,n,m)
y=matrix(v,[sizes])
```

Parameters

v
: a vector, a matrix or an hypermatrix

n,m
: integers

sizes
: vector of integers

y
: a vector, a matrix or hypermatrix

Description

For a vector or a matrix with `n` x `m` entries, the command `y=matrix(v,n,m)` or similarily `y=matrix(v,[n,m])` transforms the `v` vector (or matrix) into an `nxm` matrix by stacking columnwise the entries of `v`.

if one of the dimension m or n is equal to -1 it is automatically assigned to the quotient of size(v,'*') by the other dimension,

For an hypermatrix such as `prod(size(v))==prod(sizes)`, the command `y=matrix(v,sizes)` (or equivalently `y=matrix(v,n1,n2,...nm)`) transforms `v` into an matrix or hypermatrix by stacking "columnwise" (first dimension is varying first) the entries of `v`. `y=matrix(v,sizes)` results in a regular matrix if sizes is a scalar or a 2-vector.

Examples

```
a=[1 2 3;4 5 6]
matrix(a,1,6)
matrix(a,1,-1)
matrix(a,3,2)
```

See Also

matrices , hypermatrices , ones , zeros , rand , poly , empty

Name

mode — select a mode in exec file

```
mode(k)
k=mode()
```

Description

Used exclusively inside an exec-file or a scilab function `mode(k)` allows to change the information displayed during the execution, depending on the value of `k`:

k=0
: The new variable values are displayed if required (see help on semicolon or comma).

k=-1
: the exec file or scilab function executes silently. (this is the default value for scilab functions)

k=1 or k=3
: each line of instructions is echoed preceded of the prompt(if possible). The new variable values are displayed if required. This is the default for exec files.

k=7
: The new variable values are displayed if required, each line of instructions is echoed (if possible) and a prompt (>>) is issued after each line waiting for a carriage return .

  If carriage return follows character "p" the execution is paused (see pause).

Line display is disabled for compiled scilab function (see getf or comp). By default, Scilab functions are executed using the silent ("-1") mode.

See Also

exec , getf , semicolon , comma

Name

mtlb_mode — switch Matlab like operations

```
mmode=mtlb_mode()
mtlb_mode(mmode)
```

Parameters

mmode
: boolean

Description

Scilab and Matlab additions and substractions work differently when used with empty matrices:

Scilab
:

```
a+[] -->a
a-[] -->a
[]+a -->a
[]-a -->-a
```

Matlab

```
a+[] -->[]
a-[] -->[]
[]+a -->[]
[]-a -->[]
```

`mtlb_mode(%t)` switches to Matlab evaluation mode for additions and substractions. `mtlb_mode(%f)` switches back to Scilab mode.

`mtlb_mode()` return the current mmode' value

See Also

empty

Name

names — scilab names syntax

Description

Names of variables and functions must begin with a letter or one of the following special characters '%', '_', '#', '!', '$', '?'.

Next characters may be letters or digits or any special character in '_', '#', '!', '$', '?'

Names may be as long as you want but only the first 24 characters are taken into account. Upper and lower case letters are different.

Examples

```
//Valid names
%eps
A1=123
#Color=8
My_Special_Color_Table=rand(10,3)
//Non valid names
//1A , b%, .C
```

Name

newfun — add a name in the table of functions

```
newfun("function-name",nameptr)
```

Description

Utility function (for experts only). Adds the name `"function-name"` in the table of functions known to the interpreter. `"nameptr"` is an integer `100*fun+fin` where `fun` and `fin` is the internal coding of the primitive `"function-name"`. This function is useful to associate a primitive to a routine interfaced in `"matusr.f"` (`fun=14`). Used with `funptr` and `clearfun` one can redefine a primitive by a function with same name.

See Also

clearfun

Name

null — delete an element in a list

```
l(i)=null()
```

Description

Deletion of objects inside a list

Examples

```
l=list(1,2,3);
l(2)=null() // get list(1,3)
```

See Also

list , clear

Name

parents — ( ) left and right parenthesis

```
(expression)
[...]=func(e1,e2,...)
[x1,x2,...]=(e1,e2,...)
x(i,j)
v(i)
[...]=l(i)
```

Parameters

x
: matrix of any possible type

v
: row or column vector of any possible type

l
: list variable

func
: any function name

e1,e2,...
: any possible type expression

Description

Left and right parenthesis are used to

* Specify evaluation order within expressions,

* Form right-hand-side functions argument list. Within multiple rhs arguments must be separated by comma.

* Select elements within vectors, matrices and lists. see help on extraction and insertion for more precisions

* [x1,x2,...]=(e1,e2,...) is equivalent to first performing %t_1 = e1, %t_2 = e2, ..., and then x1 = %t_1, x2 = %t_2, ..., where the variables %t_i, i = 1, 2, ... are invisible to the user.

Examples

```
3^(-1)
x=poly(0,"x");
//
(x+10)/2
i3=eye(3,3)
//
a=[1 2 3;4 5 6;7 8 9],a(1,3),a([1 3],:),a(:,3)
a(:,3)=[]
a(1,$)=33
a(2,[$ $-1])
a(:,$+1)=[10;11;12]
```

```
//
w=ssrand(2,2,2);ssprint(w)
ssprint(w(:,1))
ss2tf(w(:,1))
//
l=list(1,2,3,4)
[a,b,c,d]=l(:)
l($+1)='new'
//
v=%t([1 1 1 1 1])
//
[x,y,z]=(1,2,3)
```

See Also

colon , comma , brackets , list , extraction , insertion

Name

pause — pause mode, invoke keyboard

Description

Switch to the `pause` mode; inserted in the code of a function, `pause` interrupts the execution of the function: one receives a prompt symbol which indicates the level of the `pause` (e.g. `-1->`). The user is then in a new workspace in which all the lower-level variables (and in particular all the variable of the function) are available. To return to the calling workspace enter `"return"`

In this mode, `[...]=return(...)` returns the variables of the argument `(...)` to the calling workspace with names in the output `[...]`. Otherwise, the lower-level variables are protected and cannot be modified.

The `pause` is extremely useful for debugging purposes.

This mode is killed by the command `"abort"`.

See Also

halt , return , abort , quit , whereami , where

Name

percent — (%) special character

Description

Some predefined variables names begin with %, such as `%i` (for `sqrt(-1)`), `%inf` (for `Infinity`), `%pi` (for `3.14...`), `%T` (for the boolean constant `"true"`),...

In addition, functions whose names begin with % are special : they are used for primitives and operators overloading (see `overloading`).

For example the function `%rmr` performs the multiplication (`m`) operation `x*y` for `x` and `y` rational matrices (`r`). The coding conventions are given by the readme file in directory `SCIDIR/macros/percent`.

Examples

```
x1=tlist('x',1,2);
x2=tlist('x',2,3);
deff('x=%xmx(x1,x2)','x=list(''x'',x1(2)*x2(2),x2(3)*x2(3))');
x1*x2
```

See Also

overloading

Name

perl — Call Perl script using appropriate operating system executable

```
perl('perlfile')
perl('perlfile',arg1,arg2,...)
result = perl(...)
```

Description

perl('perlfile') calls the Perl script perlfile, using the appropriate operating system Perl executable.

perl('perlfile',arg1,arg2,...) calls the Perl script perlfile, using the appropriate operating system Perl executable, and passes the arguments arg1, arg2, and so on, to perlfile.

result = perl(...) returns the results of attempted Perl call to result.

See Also

unix

Authors

A.C

Name

plus — (+) addition operator

```
X+Y
str1+str2
```

Parameters

X,Y
: scalar or vector or matrix of numbers, polynomials or rationals. It may also be a `syslin` list

str1,str2
: a character string, a vector or a matrix of character strings

Description

Addition.

For numeric operands addition as its usual meaning. If one of the operands is a matrix and the other one a scalar the scalar is added to each matrix entries. if one of the operands is an empty matrix the other operand is returned (this default behavior can be modified by the function `mtlb_mode`).

For character strings + means concatenation.

Addition may also be defined for other data types through "soft-coded" operations (see `overloading`).

Examples

```
[1,2]+1
[]+2
s=poly(0,"s");
s+2
1/s+2
"cat"+"enate"
```

See Also

addf , mtlb_mode , overloading

Name

poly — polynomial definition

```
p=poly(a,vname, ["flag"])
```

Parameters

a
: matrix or real number

vname
: String, the symbolic variable name. If the string have more than 4 characters only the first 4 are taken into account.

"flag"
: string ("roots", "coeff"), default value is "roots".

Description

If a is a matrix,
: `p` is the characteristic polynomial i.e. `determinant(x*eye()-a)`, `x` being the symbolic variable.

If v is a vector,

- `poly(v,"x",["roots"])` is the polynomial with `roots` the entries of `v` and `"x"` as formal variable. (In this case, `roots` and `poly` are inverse functions). Note that Infinite roots gives zero highest degree coefficients.
- `poly(v,"x","coeff")` creates the polynomial with symbol `"x"` and with coefficients the entries of `v` (v(1) is the constant term of the polynomial). (Here `poly` and `coeff` are inverse functions).

`s=poly(0,"s")` is the seed for defining polynomials with symbol `"s"`.

Examples

```
s=poly(0,"s");p=1+s+2*s^2;
A=rand(2,2);poly(A,"x")
//rational fractions
h=(1+2*%s)/poly(1:4,'s','c')
```

See Also

coeff , roots , varn , horner , derivat , matrices , rational

Name

power — power operation (^,.^)

```
t=A^b
t=A**b
t=A.^b
```

Parameters

A,t
: scalar, polynomial or rational matrix.

b
: :a scalar, a vector or a scalar matrix.

Description

- `"(A:square)^(b:scalar)"`If `A` is a square matrix and `b` is a scalar then `A^b` is the matrix `A` to the power `b`.
- `"(A:matrix).^(b:scalar)"`If `b` is a scalar and `A` a matrix then `A.^b` is the matrix formed by the element of `A` to the power `b` (elementwise power). If `A` is a vector and `b` is a scalar then `A^b` and `A.^b` performs the same operation (i.e elementwise power).
- `"(A:scalar).^(b:matrix)"` If `A` is a scalar and `b` is a matrix (or vector) `A^b` and `A.^b` are the matrices (or vectors) formed by `a^(b(i,j))`.
- `"(A:matrix).^(b:matrix)"` If `A` and `b` are vectors (matrices) of the same size `A.^b` is the `A(i)^b(i)` vector (`A(i,j)^b(i,j)` matrix).

Notes:

- For square matrices `A^p` is computed through successive matrices multiplications if `p` is a positive integer, and by diagonalization if not.

- `**` and `^` operators are synonyms.

Examples

```
A=[1 2;3 4];
A^2.5,
A.^2.5
(1:10)^2
(1:10).^2

s=poly(0,'s')
s^(1:10)
```

See Also

exp

Name

predef — variable protection

```
n=predef()
oldnew=predef(n)
oldnew=predef('all')
oldnew=predef('clear')
```

Description

Utility function used for defining "oldest" variables as "protected". Protected variables cannot be killed. They are not saved by the `'save'` command. The "oldest" are those appearing last in the `who('get')`.

`predef()` gets the number of protected variables

`predef('a[ll]')` sets all the variables protected, it also return the old and new value of protected variables number.

`predef('c[lear]')` unprotect all but the last 7 variables, it also return the old and new value of protected variables number.

`predef(n)` sets the `max(n,7)` last defined variables as protected, it also return the old and new value of protected variables number.

Remark

A number of protected variables are set in the start-up file `SCI/etc/scilab.start`. User may in particular set its own predefined variables in user's startup files `SCIHOME/.scilab` and `SCIHOME/scilab.ini`

SCIHOME definition : On Windows : C:/Documents and Settings/<User>/Scilab/<Scilab-Version> On Linux/Unix : /home/<User>/.Scilab/<Scilab-Version>

See Also

clear , save

Name

pwd — print Scilab current directory
getcwd — get Scilab current directory

```
pwd
x=pwd()
x=getcwd()
```

Description

`pwd` returns in `ans` the Scilab current directory. `x=pwd()` or `x=getcwd()` returns in `x` the Scilab current directory.

Examples

```
pwd
x=pwd()
```

See Also

chdir , cd

Name

quit — Terminates Scilab or decreases the pause level

```
quit
```

Description

The `quit` command has two different meanings depending on the calling context:

If there is no `pause` active,
then the `quit` command makes Scilab terminate, even if the command is called inside a function.

If there is a `pause` active,
then the `quit` command makes aborts the instructions started at this pause level ando terminates the current pause level.

Examples

```
// quit Scilab
function foo(x),if x then quit,end,endfunction
foo(%t) //quits scilab

//terminate instruction started in a pause context
function foo(x),if x then quit,end,endfunction
pause
foo(%t) //returns at the main prompt level

function foo1(x),
  mprintf('P1\n')
  if x then pause, mprintf('P2\n'),end,
  mprintf('P3\n')
endfunction

foo1(%t) //enter quit at the following prompt
```

See Also

pause , break , abort , exit

Name

quote — (') transpose operator, string delimiter

Description

quote (')is used for (Conjugate) Transpose of matrix.

quote (.')is used for (non Conjugate) Transpose of matrix.

Simple (') or double (") quotes are also used to define character strings. (Character strings are defined between two quotes). A Quote within a character string is denoted by two quotes.

Examples

```
[1+%i, 2]'
[1+%i, 2].'
x='This is a character string'
'He said:''Good'''
```

Name

rational — Scilab objects, rational in Scilab

Description

A rational `r` is a quotient of two polynomials `r=num/den`. The internal representation of a rational is a list. `r=tlist('['r','num','den','dt'],num,den,[])` is the same as `r=num/den`. A rational matrix can be defined with the usual syntax e.g. `[r11,r12;r21,r22]` is a 2x2 matrix where `rij` are 1x1 rationals. A rational matrix can also be defined as above as a list `tlist(['r','num','den','dt'],num,den,[])` with `num` and `den` polynomial matrices.

Examples

```
s=poly(0,'s');
W=[1/s,1/(s+1)]
W'*W
Num=[s,s+2;1,s];Den=[s*s,s;s,s*s];
tlist(['r','num','den','dt'],Num,Den,[])
H=Num./Den
syslin('c',Num,Den)
syslin('c',H)
[Num1,Den1]=simp(Num,Den)
```

See Also

poly , syslin , simp

Name

readgateway — get primitives list of a module

```
readgateway(module_name)
primitives = readgateway(module_name);
[primitives,primitivesID] = readgateway(module_name);
[primitives,primitivesID,gatewayID] = readgateway(module_name);
```

Description

get primitives list of a module.

primitives : list of primitives of a module.

primitivesID : list of ID for primitives.

gatewayID : list of ID of gateway associated to a module.

Examples

```
[primitives,primitivesID,gatewayID] = readgateway('core');
primitives(1) // 'debug' primitive
primitivesID(1) // 1 is ID of 'debug' in 'core' gateway
gatewayID(1) // 13 is ID of 'core' gateway in scilab
```

See Also

getmodules

Name

resume — return or resume execution and copy some local variables

```
resume
[x1,..,xn]=resume(a1,..,an)
```

Parameters

x

...

Description

In a function `resume` stops the execution of the function, `[..]=resume(..)` stops the execution of the function and put the local variables `ai` in calling environnement under names `xi`.

In `pause` mode, it allows to return to lower level `[..]=resume(..)` returns to lower level and put the local variables `ai` in calling environnement under names `xi`.

In an `execstr` called by a function `[..]=resume(..)` stops the execution of the function and put the local variables `ai` in calling environnement under names `xi`.

`resume` is equivalent to `return`.

See Also

abort , break

Name

return — return or resume execution and copy some local variables

```
return
[x1,..,xn]=return(a1,..,an)
```

Parameters

x

...

Description

In a function `return` stops the execution of the function, `[..]=return(..)` stops the execution of the function and put the local variables `ai` in calling environnement under names `xi`.

In `pause` mode, it allows to return to upper level `[..]=return(..)` returns to upper level and put the local variables `ai` in calling environnement under names `xi`.

In an `execstr` called by a function `[..]=return(..)` stops the execution of the function and put the local variables `ai` in calling environnement under names `xi`.

`resume` is equivalent to `return`.

See Also

abort , break

Name

sciargs — scilab command line arguments

```
args=sciargs()
```

Description

This function returns a vector of character strings containing the arguments of the Scilab command line. First `args` entry contains the path of the lanched executable file.

This function corresponds to the `getarg` function in C langage

See Also

getenv

Name

scilab — Major unix script to execute Scilab and miscellaneous tools

```
scilab <Options>
```

Description

-args Arguments
: if this option is present arguments are passed to Scilab. They can then be got by sciargs function. For multi arguments passing use a quoted, blank separated sequence of words like: `scilab -args 'foo1 foo2'`

-display Display
: for use under Xwindow systems only to set a specific X server display. Default display is unix:0.0

`-display` can be abbreviated by `-d`

-debug
: Start Scilab under the debugguer gdb (Unix/linux only).

-e Instruction
: if this option is present then Scilab instruction `Instruction` is executed first (just after startup file execution) into Scilab. `-e` and `-f` options are mutually exclusive.

-f file
: if this option is present then Scilab script `file` is executed first (just after startup file execution) into Scilab. `-e` and `-f` options are mutually exclusive.

-l lang
: :if this option is present it fixes the user language. The possible `lang` values are `'fr'` for french and `'en'` for english. The default language is english. This default value is fixed the `scilab.start` file.

scilab -link <objects>
: Is used to produce a local `scilex` (executable code of Scilab) linked with the additional files given by the user in `<objects>`. This command also produces a `scilab` script, which when called will ran the new generated scilex file.

  For example:

```
scilab -link  C/interf.o C/evol.o C/bib.a
```

  will create a new `scilex` file in which the default `interf.o` file will be replaced by `C/interf.o`.

  `-link` option cannot be used with any of the other options.

-mem N
: :set the initial stacksize, for use with -ns option. Without -ns option the initial stacksize is set by scilab.start script.

-nb
: :if this option is present then the scilab welcome banner is not displayed.

-ns
:if this option is present the startup file `SCI/etc/scilab.start` and the user startup files `SCIHOME/.scilab`, `SCIHOME/scilab.ini` are not executed.

-nouserstartup
:if this option is present the user startup files `SCIHOME/.scilab`, `SCIHOME/scilab.ini` are not executed.

-nw
:if this option is present then scilab is not run in an specific window.

-nwni
:if this option is present then scilab is not run in an specific window and does not accept user interaction. This option may be used with -f or -e options.

--texmacs
:This option is reserved for TeXMacs.

-version
:This option print product version and exit.

Name

select — select keyword

Description

```
select expr,
  case expr1 then instructions1,
  case expr2 then instructions2,
  ...
  case exprn then instructionsn,
  [else instructions],
end
```

Notes:

- The only constraint is that each "then" keyword must be on the same line line as corresponding "case" keyword.
- The keyword "then" can be replaced by a carriage return or a comma. `instructions1` are executed if `expr1=expr`, etc.

Warning: the number of characters used to define the body of any conditionnal instruction (if while for or select/case) must be limited to 16k.

Examples

```
while %t do
  n=round(10*rand(1,1))
  select n
  case 0 then
    disp(0)
  case 1 then
    disp(1)
  else
    break
  end
end
```

See Also

if , while , for

Name

semicolon (;) — ending expression and row separator

Description

semicolons are used to separate rows in a matrix definition (within brackets).

semicolons may also be used at the end of an instruction (in a file or in Scilab console). In this case it means that the result(s) is(are) not displayed. Conversely use comma (,) to get the display.

Examples

```
a=[1,2,3;4,5,6];
a=1;b=1,c=2
```

See Also

comma , brackets

Name

setbpt — set breakpoints

```
setbpt(macroname [,linenumb])
```

Parameters

macroname
: string

linenumb
: scalar integer or vector of integers

Description

`setbpt` interactively inserts a breakpoint in the line number `linenumb` (default value is 1) of the function `macroname`

`linenumb` can be a line or column vector of line numbers, or a single scalar line number.

When reaching the breakpoint, Scilab evaluates the specified line, prints the number of the line and the name of the function. If the function is not compiled (see `comp`) the line is printed on the screen. Then Scilab goes into a `pause` mode in which the user can check current values. The `pause` is exited with `resume` or `abort`. Redefining the function does not clear the breakpoints, the user must explicitly delete breakpoints using `delbpt`. The maximum number of functions with breakpoints enabled must be less than 100 and the maximum number of breakpoints is set to 1000.

Examples

```
setbpt('foo'),setbpt('foo',10),dispbpt()
delbpt()

setbpt('foo',[1,2,5,6]),dispbpt()
```

See Also

delbpt , dispbpt , pause , resume

Name

sethomedirectory — Set Scilab home directory

```
[home,scilabhome]=sethomedirectory()
```

Description

Set Scilab home path : "SCIHOME" variable.

On Windows 2k and XP , C:\Documents and Settings\<User>\Scilab\<Scilab-Version>

On Windows Vista , C:\Users\<User>\Scilab\<Scilab-Version>

On Unix, /home/<User>/.Scilab/<Scilab-Version>

Authors

Allan CORNET

Name

slash — (/) right division and feed back

Description

Right division. `x=A / b` is the solution of `x*b=A` .

`b/a = (a' \ b')'` .

`a ./ b` is the matrix with entries `a(i,j)/ b(i,j)`. If `b` is scalar (1x1 matrix) this operation is the same as `a./b*ones(a)`. (Same convention if `a` is a scalar).

Remark that `123./b` is interpreted as `(123.)/b`. In this cases dot is part of the number not of the operator.

Backslash stands for left division.

System feed back. `S=G/.K` evaluates `S=G*(eye()+K*G)^(-1)` this operator avoid simplification problem.

Remark that `G/.5` is interpreted as `G/(.5)`. In such cases dot is part of the number, not of the operator.

Comment `//` comments a line i.e lines which begin by `//` are ignored by the interpreter.

See Also

inv , percent , backslash , ieee

Name

stacksize — set scilab stack size

```
stacksize(n)
stacksize('max')
stacksize('min')
sz=stacksize()
```

Parameters

n
: integer, the required stack size given in number of double precision words

sz
: 2-vector [total used]

Description

Scilab stores "usual" variables in a stack `stk` (for global variables see `gstacksize`).

`stacksize(n)` allows the user to increase or decrease the size of this stack. The maximum allowed size depends on the amount of free memory and swap space available at the time.

`stacksize('max')` allows the user to increase the size of this stack to the maximum.

`stacksize('min')` allows the user to decrease the size of this stack to the minimum.

This function with the `n` argument can now be used everywhere.

`sz=stacksize()` returns a 2-vector which contains the current total and used stack size.

See Also

who , gstacksize

Name

star — (*) multiplication operator

Description

Multiplication. Usual meaning. Valid for constant, boolean, polynomial, rational matrices and for `syslin` lists (the meaning is series connection)

Element-wise multiplication is denoted `x.*y`. If `x` or `y` is scalar (1x1 matrix) `.*` is the same as `*`.

Kronecker product is `x.*.y`

`A*.B` is an operator with no predefined meaning. It may be used to define a new operator (see overloading) with the same precedence as * or /.

See Also

slash , backslash , syslin

Name

startup — startup file

Description

The startup file `SCIHOME/.scilab` and `SCIHOME/scilab.ini` in are automatically executed (if present) when Scilab is invoked, in addition with the file `scilab.star` in the Scilab directory (SCI).

Remarks

Last line of startup file must be terminated by a newline to be taken into account.

SCIHOME definition : On Windows : C:/Documents and Settings/<User>/Scilab/<Scilab-Version>

or on Vista : C:/<User>/AppData/Roaming/Scilab/<Scilab-Version>

On Linux/Unix : /home/<User>/.Scilab/<Scilab-Version>

See Also

scilab

Name

symbols — scilab operator names

Description

Use the following names to get help on a specific symbol.

operator	name in Scilab help
',", .'	quote
+	plus
-	minus
,.	star
/, ./, /.	slash
\,.\ ,\.	backslash
.	dot
=, ==	equal
<,>,>=,<=,<>	less
~	tilda
[	left
]	right
()	parents
%	percent
:	colon
,	comma
;	semicolon
^	hat
.^	power
|	or
&	and
.*., ./., .\.	kron

Remark

For historical reasons, different symbols may represent the same operator:

{ as the same meaning as [

} as the same meaning as]

@ as the same meaning as ~

` as the same meaning as <

It is highly recommended not to use these features because they will be removed in the future

See Also

overloading

Name

testmatrix — generate some particular matrices

```
[y]=testmatrix(name,n)
```

Parameters

name
: a character string

n
: integers, matrix size

y
: `n` x `m` matrix

Description

Create some particular matrices

testmatrix('magi',n)
: returns a magic square of size `n` .

testmatrix('frk',n)
: returns the Franck matrix :

testmatrix('hilb',n)
: is the inverse of the nxn Hilbert matrix `(Hij= 1/(i+j-1))`.

Name

then — keyword in if-then-else

Description

Used with `if`.

See Also

if

Name

tilda — (~) logical not

```
~m
```

Parameters

m
: boolean matrix

Description

`~m` is the negation of `m`.

Name

try — beginning of try block in try-catch control instruction
catch — beginning of catch block in try-catch control instruction

```
try
statements
catch
statements
end
```

Description

The `try-catch` control instruction can be used to manage codes that could possibly generate errors.

When a `try-catch` control instruction is executed, normally only the statements between the `try` and `catch` keywords are executed. However, if an error occurs during execution of any of these statements, the error is recorded, the remaining statements up to the `catch` keyword are skipped and the statements between the `catch` and `end` keywords are executed using the default error handling mode (see: errcatch.

The recorded error can be retreived using the lasterror function.

The `catch` statements as well as the `catch` keyword can be omitted if no alternative statements are given.

Note that one can also use the execstr function with `'errcatch'` argument for error handling. This can be particularily useful for handling syntactical errors.

Examples

```
file_path=TMPDIR+'/wrong'
try
   u=mopen(file_path,'r')
   x=mget(10,'c',u)
 catch
   disp(['file '+file_path+ 'cannot be read',
         'using default values for x'])
   x=1:10
 end
 [error_message,error_number]=lasterror(%t)
```

See Also

error , execstr , if , lasterror , errcatch

Authors

Serge Steer, INRIA

Name

type — Returns the type of a variable

```
[i]=type(x)
```

Parameters

x
: Scilab object

i
: integer

Description

`type(x)` returns an integer which is the type of `x` as following :

1 : real or complex constant matrix.

2 : polynomial matrix.

4 : boolean matrix.

5 : sparse matrix.

6 : sparse boolean matrix.

7 : Matlab sparse matrix.

8 : matrix of integers stored on 1 2 or 4 bytes.

9 : matrix of graphic handles.

10 : matrix of character strings.

11 : un-compiled function (Scilab code).

13 : compiled function (Scilab code).

14 : function library.

15 : list.

16 : typed list (tlist).

17 : matrix oriented typed list (mlist).

128 : pointer (See lufact).

129 : size implicit polynomial used for indexing.

130 : Scilab intrinsic (C or Fortran code).

See Also

typeof

Name

typename — associates a name to variable type

```
[types [ [,names]]=typename()
typename(name,type)
```

Parameters

types
: integer column vector: the types codes of each defined data types.

names
: column vector of strings: the names associated to type codes.

type
: integer: the type code of new data type.

name
: string: the name associated to the type code

Description

The function and operator overloading make use of a formal name associated to data types to form the name of the overloading function (see overloading). The `typename` can be used to handle this formal names for hard coded data types (the `tlist` or `mlist` coded data types formal names are defined in an other way, see `overloading`).

Called without right hand side argument, `typename` returns information on defined data types.

Called with right hand side argument, `typename` associates a name to a data type code.

`typename('',type)` suppress the data type given by its code `type` out of the table of known data types.

See Also

type , typeof , overloading , tlist , mlist

Name

user — interfacing a Fortran or C routine

```
[s_1,s_2,...,s_lhs]=user(e_1,e_2,...,e_rhs)
```

Description

With this command it is possible to use an external program as a Scilab command where `(s_1,s_2,...,s_lhs)` are the output variables and `(e_1,e_2,...,e_rhs)` are the input variables. To insert this command in Scilab one has to write a few lines in the `user` fortran subroutine of Scilab. See `intersci` or the Scilab documentation for more information.

See Also

fort , link

Name

varn — symbolic variable of a polynomial

```
[symb]=varn(p)
[pm]=varn(x,var)
```

Parameters

p
: polynomial (or matrix polynomial)

symb
: character string

x
: polynomial or polynomial matrix

var
: symbolic variable (character string)

pm
: matrix or polynomial matrix

Description

`symb=varn(p)` returns in `symb` the symbolic variable of the polynomial `p` (i.e. `varn(poly(0,'x'))` is `'x'`).

`varn(x,'s')` returns a polynomial matrix with same coefficients as `x` but with `'s'` as symbolic variable (change of variable name).

Examples

```
//
s=poly(0,'s');p=[s^2+1,s];
varn(p)
varn(p,'x')
```

See Also

horner , poly

Name

ver — Version information for Scilab

```
r = ver()
```

Parameters

r
: a matrix of strings

Description

Version information for Scilab.

returns a matrix of string with information about Scilab.

Examples

```
ver
```

Authors

A.C

Name

warning — warning messages

```
warning('string')
warning('off')
warning('on')
mode = warning('query')
```

Description

prints the character string `'string'` in a warning message

`'on'` enable warning messages.

`'off'` disable warning messages.

`'query'` get state 'on' or 'off'.

Examples

```
warning('on')
warning('this is a warning')
warning('off')
warning('this is a warning')
warning('query')
warning('on')
```

See Also

error

Name

what — list the Scilab primitives

```
what()
[primitives,commands]=what();
```

Description

List of low level primitives and commands.

Authors

A.C

Name

where — get current instruction calling tree

```
[linenum,mac]=where()
```

Parameters

linenum
: column vector of integer

mac
: column vector of strings

Description

returns `linenum` and `mac` such as current instruction has been called by the linenum(1) line of function mac(1), mac(1) has been called by the linenum(2) line of function mac(2) and so on

mac(i) is in general the name of a function but it may also be "exec" or "execstr" if instruction lies in ans exec file or an execstr instruction

See Also

whereami , pause

Name

whereami — display current instruction calling tree

```
whereami()
```

Description

Displays calling tree to instruction which contain whereami(). May be used within pause levels.

Examples

```
deff('y=test(a)',['y=sin(a)+1';
                  'y=t1(y)';
                  'y=y+1'])
deff('y=t1(y)',['y=y^2';'whereami()'])
test(1)
```

See Also

where , pause , errcatch

Name

whereis — name of library containing a function

```
[librname]=whereis(function-name)
```

Description

returns as a character string the name of the library containing the function `function-name`. The path of the library is returned by typing `"librname"`.

See Also

lib

Name

while — while keyword

Description

`while` clause. Must be terminated by `"end"`

```
while expr ,instructions,...[,else instructions], end
```

```
while expr do instructions,...[,else instructions], end
```

```
while expr then instructions,...[,else instructions], end
```

Notes:

- The only constraint is that each `then` or `dod`" keyword must be on the same line line as `while` keyword.
- Keywords `then` or `do` can be replaced by a carriage return or a comma. For compatibility with Matlab it is also possible, but not recommended, to put a space between the end of the expression and the beginning of the first instruction.
- The optional `,else  instructions` construction allows to gives instructions which are executed when expr expression becomes false.

Warning: the number of characters used to define the body of any conditionnal instruction (if while for or select/case) must be limited to 16k.

Examples

```
e=1; a=1; k=1;
while norm(a-(a+e),1) > %eps, e=e/2; k=k+1; end
e,k
```

See Also

for , select , break , return , pause

Name

who — listing of variables

```
who
who()
names=who('local')
[names,mem]=who('local')
names=who('global')
[names,mem]=who('global')
who('sorted')
names=who('local','sorted')
[names,mem]=who('local','sorted')
names=who('global','sorted')
[names,mem]=who('global','sorted')
```

Description

`who` displays current variable names.

`who('local')` or `who('get')` Returns current variable names and memory used in double precision words.

`who('global')` returns global variable names and memory used in double precision words.

`who('sorted')` displays in alphabetical order all variables.

See Also

whos , who_user

Name

who_user — listing of user's variables

```
who_user()
```

Description

`who_user` displays user's variable names.

See Also

whos , who

Name

whos — listing of variables in long form

```
whos()
whos -type typ
whos -name nam
```

Parameters

typ
: name of selected variable type (see typeof)

nam
: first characters of selected names

Description

`whos()` displays all current variable names, types and memory used.

`whos -type typ` displays all current variables with specified type.

`whos -name nam` displays all current variables whose names begin with `nam`.

Note : If a variable is global, a * appears ahead of his type.

Examples

```
lines(0)
whos()
whos -type boolean
whos -name %
```

See Also

who , who_user , typeof

Name

with_atlas — Checks if Scilab has been built with Atlas Library

```
r=with_atlas()
```

Parameters

r
: a boolean

Description

Returns `%t` if Scilab as been built with Atlas Library or `%f` if not.

Name

with_gtk — Checks if Scilab has been built with the "GIMP Toolkit" library

```
r=with_gtk()
```

Parameters

r
: a boolean

Description

Returns always `%f` gtk library is not supported by Scilab 5 and more.

Name

with_javasci — Checks if Scilab has been built with the java interface

```
r=with_javasci()
```

Parameters

r
: a boolean

Description

Returns `%t` if Scilab as been built with the java interface or `%f` if not.

Name

with_macros_source — Checks if macros source are installed

```
r=with_macros_source()
```

Parameters

r
: a boolean

Description

Returns `%t` if macros source are installed or `%f` if not.

Name

with_module — Checks if a Scilab module is installed

```
r=with_module(module_name)
```

Parameters

r
: a boolean

module_name
: a string. example : 'core'

Description

Returns `%t` Checks if a Scilab module is installed.

See Also

getmodules

Authors

A.C

Name

with_pvm — Checks if Scilab has been built with the "Parallel Virtual Machine" interface

```
r=with_pvm()
```

Parameters

r
: a boolean

Description

Returns `%t` if Scilab as been built with the "Parallel Virtual Machine" interface or `%f` if not.

Name

with_texmacs — Checks if Scilab has been called by texmacs

```
r=with_texmacs()
```

Parameters

r
: a boolean

Description

Returns `%t` if Scilab as been been called by TeXmacs

Name

with_tk — Checks if Scilab has been built with TCL/TK

```
r=with_tk()
```

Parameters

r
: a boolean

Description

Returns `%t` if Scilab as been built with TCL/TK interface or `%f` if not.

ARnoldi PACKage

Name

dnaupd — Interface for the Implicitly Restarted Arnoldi Iteration, to compute approximations to a few eigenpairs of a real linear operator

```
[IDO,RESID,V,IPARAM,IPNTR,WORKD,WORKL,INFO] = dnaupd(ID0,BMAT,N,WHICH,NEV,TOL,R
```

Parameters

ID0
: Integer. (INPUT/OUTPUT) Reverse communication flag. IDO must be zero on the first call to dnaupd. IDO will be set internally to indicate the type of operation to be performed. Control is then given back to the calling routine which has the responsibility to carry out the requested operation and call dnaupd with the result. The operand is given in WORKD(IPNTR(1)), the result must be put in WORKD(IPNTR(2)).

  IDO = 0: first call to the reverse communication interface

  IDO = -1: compute Y = OP * X where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y. This is for the initialization phase to force the starting vector into the range of OP.

  IDO = 1: compute Y = OP * X where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y. In mode 3 and 4, the vector B * X is already available in WORKD(ipntr(3)). It does not need to be recomputed in forming OP * X.

  IDO = 2: compute Y = B * X where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y.

  IDO = 3: compute the IPARAM(8) real and imaginary parts of the shifts where INPTR(14) is the pointer into WORKL for placing the shifts. See Remark 5 below.

  IDO = 99: done

BMAT
: Character, specifies the type of the matrix B that defines the semi-inner product for the operator OP.

  B = 'I' -> standard eigenvalue problem A*x = lambda*x

  B = 'G' -> generalized eigenvalue problem A*x = lambda*B*x

N
: Integer, dimension of the eigenproblem.

WHICH
: string of length 2, Specify which of the Ritz values of OP to compute.

  'LM' - want the NEV eigenvalues of largest magnitude.

  'SM' - want the NEV eigenvalues of smallest magnitude.

  'LR' - want the NEV eigenvalues of largest real part.

  'SR' - want the NEV eigenvalues of smallest real part.

  'LI' - want the NEV eigenvalues of largest imaginary part.

  'SI' - want the NEV eigenvalues of smallest imaginary part.

NEV
: Integer, number of eigenvalues of OP to be computed. 0 < NEV < N-1.

TOL
: scalar. Stopping criterion: the relative accuracy of the Ritz value is considered acceptable if BOUNDS(I) <= TOL*ABS(RITZ(I)). If TOL <= 0. is passed the machine precision is set.

RESID
: array of length N (INPUT/OUTPUT)

  On INPUT: If INFO==0, a random initial residual vector is used, else RESID contains the initial residual vector, possibly from a previous run.

  On OUTPUT: RESID contains the final residual vector.

NCV
: Integer. Number of columns of the matrix V. NCV must satisfy the two inequalities 2 <= NCV-NEV and NCV <= N. This will indicate how many Arnoldi vectors are generated at each iteration. After the startup phase in which NEV Arnoldi vectors are generated, the algorithm generates approximately NCV-NEV Arnoldi vectors at each subsequent update iteration. Most of the cost in generating each Arnoldi vector is in the matrix-vector operation OP*x.

  NOTE: 2 <= NCV-NEV in order that complex conjugate pairs of Ritz values are kept together. (See remark 4 below)

V
: N by NCV array. Contains the final set of Arnoldi basis vectors.

IPARAM
: array of length 11. (INPUT/OUTPUT)

  IPARAM(1) = ISHIFT: method for selecting the implicit shifts. The shifts selected at each iteration are used to restart the Arnoldi iteration in an implicit fashion.

  ISHIFT = 0: the shifts are provided by the user via reverse communication. The real and imaginary parts of the NCV eigenvalues of the Hessenberg matrix H are returned in the part of the WORKL array corresponding to RITZR and RITZI. See remark 5 below.

  ISHIFT = 1: exact shifts with respect to the current Hessenberg matrix H. This is equivalent to restarting the iteration with a starting vector that is a linear combination of approximate Schur vectors associated with the "wanted" Ritz values.

  IPARAM(2) = LEVEC No longer referenced.

  IPARAM(3) = MXITER On INPUT: maximum number of Arnoldi update iterations allowed. On OUTPUT: actual number of Arnoldi update iterations taken.

  IPARAM(4) = NB: blocksize to be used in the recurrence. The code currently works only for NB = 1.

  IPARAM(5) = NCONV: number of "converged" Ritz values. This represents the number of Ritz values that satisfy the convergence criterion.

  IPARAM(6) = IUPD No longer referenced. Implicit restarting is ALWAYS used.

  IPARAM(7) = MODE On INPUT determines what type of eigenproblem is being solved. Must be 1,2,3,4; See under Description of dnaupd for the five modes available.

  IPARAM(8) = NP When ido = 3 and the user provides shifts through reverse communication (IPARAM(1)=0), dnaupd returns NP, the number of shifts the user is to provide. 0 < NP <= NCV-NEV. See Remark 5 below.

IPARAM(9) = NUMOP, IPARAM(10) = NUMOPB, IPARAM(11) = NUMREO, OUTPUT: NUMOP = total number of OP*x operations, NUMOPB = total number of B*x operations if BMAT='G', NUMREO = total number of steps of re-orthogonalization.

IPNTR
array of length 14. Pointer to mark the starting locations in the WORKD and WORKL arrays for matrices/vectors used by the Arnoldi iteration.

IPNTR(1): pointer to the current operand vector X in WORKD.

IPNTR(2): pointer to the current result vector Y in WORKD.

IPNTR(3): pointer to the vector B * X in WORKD when used in the shift-and-invert mode.

IPNTR(4): pointer to the next available location in WORKL that is untouched by the program.

IPNTR(5): pointer to the NCV by NCV upper Hessenberg matrix H in WORKL.

IPNTR(6): pointer to the real part of the ritz value array RITZR in WORKL.

IPNTR(7): pointer to the imaginary part of the ritz value array RITZI in WORKL.

IPNTR(8): pointer to the Ritz estimates in array WORKL associated with the Ritz values located in RITZR and RITZI in WORKL.

IPNTR(14): pointer to the NP shifts in WORKL. See Remark 5 below.

Note: IPNTR(9:13) is only referenced by dneupd . See Remark 2.

IPNTR(9): pointer to the real part of the NCV RITZ values of the original system.

IPNTR(10): pointer to the imaginary part of the NCV RITZ values of the original system.

IPNTR(11): pointer to the NCV corresponding error bounds.

IPNTR(12):pointer to the NCV by NCV upper quasi-triangular Schur matrix for H.

IPNTR(11): pointer to the NCV by NCV matrix of eigenvectors of the upper Hessenberg matrix H. Only referenced by dneupd if RVEC = .TRUE. See Remark 2 below.

WORKD
Double precision work array of length 3*N. (REVERSE COMMUNICATION) Distributed array to be used in the basic Arnoldi iteration for reverse communication. The user should not use WORKD as temporary workspace during the iteration. Upon termination WORKD(1:N) contains B*RESID(1:N). If an invariant subspace associated with the converged Ritz values is desired, see remark 2 below, subroutine dneupd uses this output. See Data Distribution Note below.

WORKL
work array of length at least 3*NCV**2 + 6*NCV. (OUTPUT/WORKSPACE) Private (replicated) array on each PE or array allocated on the front end. See Data Distribution Note below.

INFO
Integer. (INPUT/OUTPUT)

If INFO == 0, a randomly initial residual vector is used, else RESID contains the initial residual vector, possibly from a previous run.

Error flag on output.

= 0: Normal exit.

= 1: Maximum number of iterations taken. All possible eigenvalues of OP has been found. IPARAM(5) returns the number of wanted converged Ritz values.

= 2: No longer an informational error. Deprecated starting with release 2 of ARPACK.

= 3: No shifts could be applied during a cycle of the Implicitly restarted Arnoldi iteration. One possibility is to increase the size of NCV relative to NEV. See remark 4 below.

= -1: N must be positive.

= -2: NEV must be positive.

= -3: NCV-NEV >= 2 and less than or equal to N.

= -4: The maximum number of Arnoldi update iterations allowed must be greater than zero.

= -5: WHICH must be one of 'LM', 'SM', 'LR', 'SR', 'LI', 'SI'

= -6: BMAT must be one of 'I' or 'G'.

= -7: Length of private work array WORKL is not sufficient.

= -8: Error return from LAPACK eigenvalue calculation;

= -9: Starting vector is zero.

= -10: IPARAM(7) must be 1,2,3,4.

= -11: IPARAM(7) = 1 and BMAT = 'G' are incompatable.

= -12: IPARAM(1) must be equal to 0 or 1.

= -9999: Could not build an Arnoldi factorization. IPARAM(5) returns the size of the current Arnoldi factorization. The user is advised to check that enough workspace and array storage has been allocated.

Description

Reverse communication interface for the Implicitly Restarted Arnoldi iteration. This subroutine computes approximations to a few eigenpairs of a linear operator "OP" with respect to a semi-inner product defined by a symmetric positive semi-definite real matrix B. B may be the identity matrix. NOTE: If the linear operator "OP" is real and symmetric with respect to the real positive semi-definite symmetric matrix B, i.e. B*OP = (OP`)*B, then subroutine dsaupd should be used instead.

The computed approximate eigenvalues are called Ritz values and the corresponding approximate eigenvectors are called Ritz vectors.

dnaupd is usually called iteratively to solve one of the following problems:

- Mode 1: A*x = lambda*x. `OP = A , B = I`.
- Mode 2: A*x = lambda*M*x, M symmetric positive definite `OP = inv[M]*A, B = M`. (If M can be factored see remark 3 below)
- Mode 3: A*x = lambda*M*x, M symmetric positive semi-definite. `OP = Real_Part{ inv[A - sigma*M]*M }, B = M`. shift-and-invert mode (in real arithmetic)

  If `OP*x = amu*x`, then

  `amu = 1/2 * [ 1/(lambda-sigma) + 1/(lambda-conjg(sigma))]`.

  Note: If sigma is real, i.e. imaginary part of sigma is zero; `Real_Part{ inv[A - sigma*M]*M } == inv[A - sigma*M]*M amu == 1/(lambda-sigma)`.

- Mode 4: A*x = lambda*M*x, M symmetric semi-definite `OP = Imaginary_Part{ inv[A - sigma*M]*M } , B = M`. shift-and-invert mode (in real arithmetic)

  If `OP*x = amu*x`, then `amu = 1/2i * [ 1/(lambda-sigma) - 1/(lambda-conjg(sigma)) ]`.

Both mode 3 and 4 give the same enhancement to eigenvalues close to the (complex) shift sigma. However, as lambda goes to infinity, the operator OP in mode 4 dampens the eigenvalues more strongly than does OP defined in mode 3.

NOTE: The action of w <- inv[A - sigma*M]*v or w <- inv[M]*v should be accomplished either by a direct method using a sparse matrix factorization and solving `[A - sigma*M]*w = v` or `M*w = v`, or through an iterative method for solving these systems. If an iterative method is used, the convergence test must be more stringent than the accuracy requirements for the eigenvalue approximations.

Remarks

1. The computed Ritz values are approximate eigenvalues of OP. The selection of WHICH should be made with this in mind when Mode = 3 and 4. After convergence, approximate eigenvalues of the original problem may be obtained with the ARPACK subroutine dneupd.

2. If a basis for the invariant subspace corresponding to the converged Ritz values is needed, the user must call dneupd immediately following completion of dnaupd. This is new starting with release 2 of ARPACK.

3. If M can be factored into a Cholesky factorization M = LL` then Mode = 2 should not be selected. Instead one should use Mode = 1 with OP = inv(L)*A*inv(L`). Appropriate triangular linear systems should be solved with L and L` rather than computing inverses. After convergence, an approximate eigenvector z of the original problem is recovered by solving L`z = x where x is a Ritz vector of OP.

4. At present there is no a-priori analysis to guide the selection of NCV relative to NEV. The only formal requrement is that NCV > NEV + 2. However, it is recommended that NCV >= 2*NEV+1. If many problems of the same type are to be solved, one should experiment with increasing NCV while keeping NEV fixed for a given test problem. This will usually decrease the required number of OP*x operations but it also increases the work and storage required to maintain the orthogonal basis vectors. The optimal "cross-over" with respect to CPU time is problem dependent and must be determined empirically. See Chapter 8 of Reference 2 for further information.

5. When IPARAM(1) = 0, and IDO = 3, the user needs to provide the NP = IPARAM(8) real and imaginary parts of the shifts in locations

```
real part                  imaginary part
-----------------------    --------------
1   WORKL(IPNTR(14))           WORKL(IPNTR(14)+NP)
2   WORKL(IPNTR(14)+1)         WORKL(IPNTR(14)+NP+1)
.                          .
.                          .
.                          .
NP  WORKL(IPNTR(14)+NP-1)      WORKL(IPNTR(14)+2*NP-1).
```

Only complex conjugate pairs of shifts may be applied and the pairs must be placed in consecutive locations. The real part of the eigenvalues of the current upper Hessenberg matrix are located in WORKL(IPNTR(6)) through WORKL(IPNTR(6)+NCV-1) and the imaginary part in WORKL(IPNTR(7)) through WORKL(IPNTR(7)+NCV-1). They are ordered according to the order defined by WHICH. The complex conjugate pairs are kept together and the associated Ritz estimates are located in WORKL(IPNTR(8)), WORKL(IPNTR(8)+1), ... , WORKL(IPNTR(8)+NCV-1).

See Also

dsaupd

Authors

Danny Sorensen, Richard Lehoucq, Phuong Vu
CRPC / Rice University Applied Mathematics Rice University Houston, Texas

Bibliography

1. D.C. Sorensen, "Implicit Application of Polynomial Filters in a k-Step Arnoldi Method", SIAM J. Matr. Anal. Apps., 13 (1992), pp 357-385.

2. R.B. Lehoucq, "Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration", Rice University Technical Report TR95-13, Department of Computational and Applied Mathematics.

3. B.N. Parlett, "The Symmetric Eigenvalue Problem". Prentice-Hall, 1980.

4. B.N. Parlett, B. Nour-Omid, "Towards a Black Box Lanczos Program", Computer Physics Communications, 53 (1989), pp 169-179.

5. B. Nour-Omid, B.N. Parlett, T. Ericson, P.S. Jensen, "How to Implement the Spectral Transformation", Math. Comp., 48 (1987), pp 663-673.

6. R.G. Grimes, J.G. Lewis and H.D. Simon, "A Shifted Block Lanczos Algorithm for Solving Sparse Symmetric Generalized Eigenproblems", SIAM J. Matr. Anal. Apps., January (1993).

7. L. Reichel, W.B. Gragg, "Algorithm 686: FORTRAN Subroutines for Updating the QR decomposition", ACM TOMS, December 1990, Volume 16 Number 4, pp 369-377.

8. R.B. Lehoucq, D.C. Sorensen, "Implementation of Some Spectral Transformations in a k-Step Arnoldi Method". In Preparation.

Used Functions

Based on ARPACK routine dnaupd

Name

dneupd — ARnoldi Package (not documented 5)

Description

not documented

Name

dsaupd — Interface for the Implicitly Restarted Arnoldi Iteration, to compute approximations to a few eigenpairs of a real and symmetric linear operator

```
[IDO,RESID,V,IPARAM,IPNTR,WORKD,WORKL,INFO] = dsaupd(ID0,BMAT,N,WHICH,NEV,TOL,R
```

Parameters

ID0

Integer. (INPUT/OUTPUT) Reverse communication flag. IDO must be zero on the first call to dsaupd . IDO will be set internally to indicate the type of operation to be performed. Control is then given back to the calling routine which has the responsibility to carry out the requested operation and call dsaupd with the result. The operand is given in WORKD(IPNTR(1)), the result must be put in WORKD(IPNTR(2)). (If Mode = 2 see remark 5 below)

IDO = 0: first call to the reverse communication interface

IDO = -1: compute Y = OP * X where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y. This is for the initialization phase to force the starting vector into the range of OP.

IDO = 1: compute Y = OP * X where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y. In mode 3,4 and 5, the vector B * X is already available in WORKD(ipntr(3)). It does not need to be recomputed in forming OP * X.

IDO = 2: compute Y = B * X where IPNTR(1) is the pointer into WORKD for X, IPNTR(2) is the pointer into WORKD for Y.

IDO = 3: compute the IPARAM(8) shifts where IPNTR(11) is the pointer into WORKL for placing the shifts. See remark 6 below.

IDO = 99: done

BMAT

Character, specifies the type of the matrix B that defines the semi-inner product for the operator OP.

B = 'I' -> standard eigenvalue problem A*x = lambda*x

B = 'G' -> generalized eigenvalue problem A*x = lambda*B*x

N

Integer, dimension of the eigenproblem.

WHICH

string of length 2, Specify which of the Ritz values of OP to compute.

'LA' - compute the NEV largest (algebraic) eigenvalues.

'SA' - compute the NEV smallest (algebraic) eigenvalues.

'LM' - compute the NEV largest (in magnitude) eigenvalues.

'SM' - compute the NEV smallest (in magnitude) eigenvalues.

'BE' - compute NEV eigenvalues, half from each end of the spectrum. When NEV is odd, compute one more from the high end than from the low end. (see remark 1 below)

NEV

Integer, number of eigenvalues of OP to be computed. 0 < NEV < N.

TOL

4. B.N. Parlett, B. Nour-Omid, "Towards a Black Box Lanczos Program", Computer Physics Communications, 53 (1989), pp 169-179.

5. B. Nour-Omid, B.N. Parlett, T. Ericson, P.S. Jensen, "How to Implement the Spectral Transformation", Math. Comp., 48 (1987), pp 663-673.

6. R.G. Grimes, J.G. Lewis and H.D. Simon, "A Shifted Block Lanczos Algorithm for Solving Sparse Symmetric Generalized Eigenproblems", SIAM J. Matr. Anal. Apps., January (1993).

7. L. Reichel, W.B. Gragg, "Algorithm 686: FORTRAN Subroutines for Updating the QR decomposition", ACM TOMS, December 1990, Volume 16 Number 4, pp 369-377.

8. R.B. Lehoucq, D.C. Sorensen, "Implementation of Some Spectral Transformations in a k-Step Arnoldi Method". In Preparation.

scalar. Stopping criterion: the relative accuracy of the Ritz value is considered acceptable if BOUNDS(I) <= TOL*ABS(RITZ(I)). If TOL <= 0. is passed the machine precision is set.

RESID

array of length N (INPUT/OUTPUT)

On INPUT: If INFO==0, a random initial residual vector is used, else RESID contains the initial residual vector, possibly from a previous run.

On OUTPUT: RESID contains the final residual vector.

NCV

Integer.Number of columns of the matrix V (less than or equal to N). This will indicate how many Lanczos vectors are generated at each iteration. After the startup phase in which NEV Lanczos vectors are generated, the algorithm generates NCV-NEV Lanczos vectors at each subsequent update iteration. Most of the cost in generating each Lanczos vector is in the matrix-vector product OP*x. (See remark 4 below).

V

N by NCV array. The NCV columns of V contain the Lanczos basis vectors.

IPARAM

array of length 11. (INPUT/OUTPUT)

IPARAM(1) = ISHIFT: method for selecting the implicit shifts. The shifts selected at each iteration are used to restart the Arnoldi iteration in an implicit fashion.

ISHIFT = 0: the shifts are provided by the user via reverse communication. The NCV eigenvalues of the current tridiagonal matrix T are returned in the part of WORKL array corresponding to RITZ. See remark 6 below.

ISHIFT = 1: exact shifts with respect to the reduced tridiagonal matrix T. This is equivalent to restarting the iteration with a starting vector that is a linear combination of Ritz vectors associated with the "wanted" Ritz values.

IPARAM(2) = LEVEC No longer referenced. See remark 2 below.

IPARAM(3) = MXITER On INPUT: maximum number of Arnoldi update iterations allowed. On OUTPUT: actual number of Arnoldi update iterations taken.

IPARAM(4) = NB: blocksize to be used in the recurrence. The code currently works only for NB = 1.

IPARAM(5) = NCONV: number of "converged" Ritz values. This represents the number of Ritz values that satisfy the convergence criterion.

IPARAM(6) = IUPD No longer referenced. Implicit restarting is ALWAYS used.

IPARAM(7) = MODE On INPUT determines what type of eigenproblem is being solved. Must be 1,2,3,4,5; See under Description of dsaupd for the five modes available.

IPARAM(8) = NP When ido = 3 and the user provides shifts through reverse communication (IPARAM(1)=0), dsaupd returns NP, the number of shifts the user is to provide. 0 < NP <= NCV-NEV. See Remark 6 below.

IPARAM(9) = NUMOP, IPARAM(10) = NUMOPB, IPARAM(11) = NUMREO, OUTPUT: NUMOP = total number of OP*x operations, NUMOPB = total number of B*x operations if BMAT='G', NUMREO = total number of steps of re-orthogonalization.

IPNTR
array of length 11. Pointer to mark the starting locations in the WORKD and WORKL arrays for matrices/vectors used by the Lanczos iteration.

IPNTR(1): pointer to the current operand vector X in WORKD.

IPNTR(2): pointer to the current result vector Y in WORKD.

IPNTR(3): pointer to the vector B * X in WORKD when used in the shift-and-invert mode.

IPNTR(4): pointer to the next available location in WORKL that is untouched by the program.

IPNTR(5): pointer to the NCV by 2 tridiagonal matrix T in WORKL.

IPNTR(6): pointer to the NCV RITZ values array in WORKL.

IPNTR(7): pointer to the Ritz estimates in array WORKL associated with the Ritz values located in RITZ in WORKL.

IPNTR(11): pointer to the NP shifts in WORKL. See Remark 6 below.

Note: IPNTR(8:10) is only referenced by dseupd . See Remark 2.

IPNTR(8): pointer to the NCV RITZ values of the original system.

IPNTR(9): pointer to the NCV corresponding error bounds.

IPNTR(10): pointer to the NCV by NCV matrix of eigenvectors of the tridiagonal matrix T. Only referenced by dseupd if RVEC = .TRUE. See Remarks

WORKD
work array of length 3*N. (REVERSE COMMUNICATION) Distributed array to be used in the basic Arnoldi iteration for reverse communication. The user should not use WORKD as temporary workspace during the iteration. Upon termination WORKD(1:N) contains B*RESID(1:N). If the Ritz vectors are desired subroutine dseupd uses this output. See Data Distribution Note below.

WORKL
work array of length at least NCV**2 + 8*NCV. (OUTPUT/WORKSPACE) Private (replicated) array on each PE or array allocated on the front end. See Data Distribution Note below. add here the parameter description

INFO
Integer. (INPUT/OUTPUT)

If INFO == 0, a randomly initial residual vector is used, else RESID contains the initial residual vector, possibly from a previous run.

Error flag on output.

= 0: Normal exit.

= 1: Maximum number of iterations taken. All possible eigenvalues of OP has been found. IPARAM(5) returns the number of wanted converged Ritz values.

= 2: No longer an informational error. Deprecated starting with release 2 of ARPACK.

= 3: No shifts could be applied during a cycle of the Implicitly restarted Arnoldi iteration. One possibility is to increase the size of NCV relative to NEV. See remark 4 below.

= -1: N must be positive.

= -2: NEV must be positive.

= -3: NCV must be greater than NEV and less than or equal to N.

= -4: The maximum number of Arnoldi update iterations allowed must be greater than zero.

= -5: WHICH must be one of 'LM', 'SM', 'LA', 'SA' or 'BE'.

= -6: BMAT must be one of 'I' or 'G'.

= -7: Length of private work array WORKL is not sufficient.

= -8: Error return from trid. eigenvalue calculation; Informatinal error from LAPACK routine dsteqr .

= -9: Starting vector is zero.

= -10: IPARAM(7) must be 1,2,3,4,5.

= -11: IPARAM(7) = 1 and BMAT = 'G' are incompatable.

= -12: IPARAM(1) must be equal to 0 or 1.

= -13: NEV and WHICH = 'BE' are incompatable.

= -9999: Could not build an Arnoldi factorization. IPARAM(5) returns the size of the current Arnoldi factorization. The user is advised to check that enough workspace and array storage has been allocated.

Description

Reverse communication interface for the Implicitly Restarted Arnoldi Iteration. For symmetric problems this reduces to a variant of the Lanczos method. This method has been designed to compute approximations to a few eigenpairs of a linear operator OP that is real and symmetric with respect to a real positive semi-definite symmetric matrix B, i.e.`B*OP = (OP`)*B`.

Another way to express this condition is `< x,OPy > = < OPx,y > where <z,w > = z`Bw .`

In the standard eigenproblem B is the identity matrix. (A` denotes transpose of A)

The computed approximate eigenvalues are called Ritz values and the corresponding approximate eigenvectors are called Ritz vectors.

dsaupd is usually called iteratively to solve one of the following problems:

- Mode 1: A*x = lambda*x, A symmetric ===> OP = A and B = I.
- Mode 2: A*x = lambda*M*x, A symmetric, M symmetric positive definite ===> OP = inv[M]*A and B = M. ===> (If M can be factored see remark 3 below)
- Mode 3: K*x = lambda*M*x, K symmetric, M symmetric positive semi-definite ===> OP = (inv[K - sigma*M])*M and B = M. ===> Shift-and-Invert mode

- Mode 4: K*x = lambda*KG*x, K symmetric positive semi-definite, KG symmetric indefinite ===> OP = (inv[K - sigma*KG])*K and B = K. ===> Buckling mode

- Mode 5: A*x = lambda*M*x, A symmetric, M symmetric positive semi-definite ===> OP = inv[A - sigma*M]*[A + sigma*M] and B = M. ===> Cayley transformed mode

NOTE: The action of w <- inv[A - sigma*M]*v or w <- inv[M]*v should be accomplished either by a direct method using a sparse matrix factorization and solving `[A - sigma*M]*w = v or M*w = v`,

or through an iterative method for solving these systems. If an iterative method is used, the convergence test must be more stringent than the accuracy requirements for the eigenvalue approximations.

Remarks

1. The converged Ritz values are always returned in ascending algebraic order. The computed Ritz values are approximate eigenvalues of OP. The selection of WHICH should be made with this in mind when Mode = 3,4,5. After convergence, approximate eigenvalues of the original problem may be obtained with the ARPACK subroutine dseupd .

2. If the Ritz vectors corresponding to the converged Ritz values are needed, the user must call dseupd immediately following completion of dsaupd . This is new starting with version 2.1 of ARPACK.

3. If M can be factored into a Cholesky factorization M = LL` then Mode = 2 should not be selected. Instead one should use Mode = 1 with OP = inv(L)*A*inv(L`). Appropriate triangular linear systems should be solved with L and L` rather than computing inverses. After convergence, an approximate eigenvector z of the original problem is recovered by solving L`z = x where x is a Ritz vector of OP.

4. At present there is no a-priori analysis to guide the selection of NCV relative to NEV. The only formal requrement is that NCV > NEV. However, it is recommended that NCV >= 2*NEV. If many problems of the same type are to be solved, one should experiment with increasing NCV while keeping NEV fixed for a given test problem. This will usually decrease the required number of OP*x operations but it also increases the work and storage required to maintain the orthogonal basis vectors. The optimal "cross-over" with respect to CPU time is problem dependent and must be determined empirically.

5. If IPARAM(7) = 2 then in the Reverse commuication interface the user must do the following. When IDO = 1, Y = OP * X is to be computed. When IPARAM(7) = 2 OP = inv(B)*A. After computing A*X the user must overwrite X with A*X. Y is then the solution to the linear set of equations B*Y = A*X.

6. When IPARAM(1) = 0, and IDO = 3, the user needs to provide the NP = IPARAM(8) shifts in locations: 1 WORKL(IPNTR(11)) 2 WORKL(IPNTR(11)+1) . . . NP WORKL(IPNTR(11)+NP-1). The eigenvalues of the current tridiagonal matrix are located in WORKL(IPNTR(6)) through WORKL(IPNTR(6)+NCV-1). They are in the order defined by WHICH. The associated Ritz estimates are located in WORKL(IPNTR(8)), WORKL(IPNTR(8)+1), ... , WORKL(IPNTR(8)+NCV-1).

See Also

dnaupd

Authors

Danny Sorensen, Richard Lehoucq, Phuong Vu
CRPC / Rice University Applied Mathematics Rice University Houston, Texas

Bibliography

1. D.C. Sorensen, "Implicit Application of Polynomial Filters in a k-Step Arnoldi Method", SIAM J. Matr. Anal. Apps., 13 (1992), pp 357-385.

2. R.B. Lehoucq, "Analysis and Implementation of an Implicitly Restarted Arnoldi Iteration", Rice University Technical Report TR95-13, Department of Computational and Applied Mathematics.

3. B.N. Parlett and Y. Saad, "Complex Shift and Invert Strategies for Real Matrices", Linear Algebra and its Applications, vol 88/89, pp 575-595, (1987).

Used Functions

Based on ARPACK routine dsaupd

Name

dseupd — ARnoldi Package (not documented 4)

Description

not documented

Name

znaupd — ARnoldi Package (not documented 3)

Description

not documented

Name

zneupd — ARnoldi Package (not documented 6)

Description

not documented

Boolean

Name

bool2s — convert boolean matrix to a zero one matrix.

```
bool2s(x)
```

Parameters

x
: a boolean vector or a boolean matrix or a constant matrix

Description

If `x` is a boolean matrix, `bool2s(x)` returns the matrix where "true" values are replaced by 1 and "false" value by 0.

If `x` is a "standard" matrix, `bool2s(x)` returns the matrix where non-zero values are replaced by 1.

Examples

```
bool2s([%t %t %f %t])
bool2s([2.3 0 10 -1])
```

See Also

boolean , find

Name

find — find indices of boolean vector or matrix true elements

```
[ii]=find(x [,nmax])
[i1,i2,..]=find(x [,nmax])
```

Parameters

x
: may be a boolean vector, a boolean matrix, a boolean hypermatrix, a "standard" matrix or hypermatrix

nmax
: an integer giving the maximum number of indices to return. The default value is -1 which stands for "all". This option can be used for efficiency, to avoid searching all indices.

ii, i1, i2, ..
: integer vectors of indices or empty matrices

Description

If `x` is a boolean matrix,

`ii=find(x)` returns the vector of indices `i` for which `x(i)` is "true". If no true element found find returns an empty matrix.

`[i1,i2,..]=find(x)` returns vectors of indices `i1` (for rows) and `i2` (for columns),.. such that `x(i1(n),i2(n),..)` is "true". If no true element found find returns empty matrices in `i1`, `i2`, ...

if `x` is a standard matrix or hypermatrix `find(x)` is interpreted as `find(x<>0)`

`find([])` returns `[]`

Examples

```
beers=["Desperados", "Leffe", "Kronenbourg", "Heineken"];
find(beers=="Leffe") // OK
find(beers=="1664") // KO
find(beers=="Foster") // KO
beers=[beers, "Foster"]
find(beers=="Foster") // OK

A=rand(1,20);
w=find(A<0.4)
A(w)
w=find(A>100)

B=rand(1,20);
w=find(B<0.4,2) //at most 2 returned values

H=rand(4,3,5); //an hypermatrix
[i,j,k]=find(H>0.9)

H(i(1),j(1),k(1))
```

See Also

boolean , extraction , insertion , vectorfind

CACSD

Name

abcd — state-space matrices

```
[A,B,C,D]=abcd(sl)
```

Parameters

sl
: linear system (`syslin` list) in state-space or transfer form

A,B,C,D
: real matrices of appropriate dimensions

Description

returns the `A,B,C,D` matrices from a linear system `Sl`.

Utility function. For transfer matrices `Sl` is converted into state-space form by `tf2ss`.

The matrices `A,B,C,D` are the elements `2` to `5` of the `syslin` list `Sl`, i.e. `[A,B,C,D] = Sl(2:5)` .

Examples

```
A=diag([1,2,3]);B=[1;1;1];C=[2,2,2];
sys=syslin('c',A,B,C);
sys("A")
sys("C")
[A1,B1,C1,D1]=abcd(sys);
A1
systf=ss2tf(sys);
[a,b,c,d]=abcd(systf)
spec(a)
c*b-C*B
c*a*b-C*A*B
```

See Also

syslin , ssrand

Name

abinv — AB invariant subspace

```
[X,dims,F,U,k,Z]=abinv(Sys,alpha,beta,flag)
```

Parameters

Sys
: `syslin` list containing the matrices `[A,B,C,D]`.

alpha
(optional) real number or vector (possibly complex, location of closed loop poles)

beta
(optional) real number or vector (possibly complex, location of closed loop poles)

flag
(optional) character string `'ge'` (default) or `'st'` or `'pp'`

X
orthogonal matrix of size nx (dim of state space).

dims
integer row vector `dims=[dimR,dimVg,dimV,noc,nos]` with `dimR<=dimVg<=dimV<=noc<=nos`. If `flag='st'`, (resp. `'pp'`), `dims` has 4 (resp. 3) components.

F
real matrix (state feedback)

k
integer (normal rank of `Sys`)

Z
non-singular linear system (`syslin` list)

Description

Output nulling subspace (maximal unobservable subspace) for `Sys` = linear system defined by a syslin list containing the matrices [A,B,C,D] of `Sys`. The vector `dims=[dimR,dimVg,dimV,noc,nos]` gives the dimensions of subspaces defined as columns of `X` according to partition given below. The `dimV` first columns of `X` i.e `V=X(:,1:dimV)`, span the AB-invariant subspace of `Sys` i.e the unobservable subspace of `(A+B*F,C+D*F)`. (`dimV=nx` iff `C^(-1)(D)=X`).

The `dimR` first columns of `X` i.e. `R=X(:,1:dimR)` spans the controllable part of `Sys` in `V`, `(dimR<=dimV)`. (`dimR=0` for a left invertible system). `R` is the maximal controllability subspace of `Sys` in `kernel(C)`.

The `dimVg` first columns of `X` spans `Vg`=maximal AB-stabilizable subspace of `Sys`. `(dimR<=dimVg<=dimV)`.

`F` is a decoupling feedback: for `X=[V,X2]` `(X2=X(:,dimV+1:nx))` one has `X2'*(A+B*F)*V=0 and (C+D*F)*V=0`.

The zeros od `Sys` are given by : `X0=X(:,dimR+1:dimV); spec(X0'*(A+B*F)*X0)` i.e. there are `dimV-dimR` closed-loop fixed modes.

If the optional parameter alpha is given as input, the dimR controllable modes of (A+BF) in V are set to alpha (or to [alpha(1), alpha(2), ...]. (alpha can be a vector (real or complex pairs) or a (real) number). Default value alpha=-1.

If the optional real parameter beta is given as input, the noc-dimV controllable modes of (A+BF) "outside" V are set to beta (or [beta(1),beta(2),...]). Default value beta=-1.

In the X,U bases, the matrices [X'*(A+B*F)*X,X'*B*U;(C+D*F)*X,D*U] are displayed as follows:

```
[A11,*,*,*,*,*]  [B11 * ]
[0,A22,*,*,*,*]  [0   * ]
[0,0,A33,*,*,*]  [0   * ]
[0,0,0,A44,*,*]  [0  B42]
[0,0,0,0,A55,*]  [0   0 ]
[0,0,0,0,0,A66]  [0   0 ]

[0,0,0,*,*,*]    [0  D2]
```

where the X-partitioning is defined by dims and the U-partitioning is defined by k.

A11 is (dimR x dimR) and has its eigenvalues set to alpha(i)'s. The pair (A11,B11) is controllable and B11 has nu-k columns. A22 is a stable (dimVg-dimR x dimVg-dimR) matrix. A33 is an unstable (dimV-dimVg x dimV-dimVg) matrix (see st_ility).

A44 is (noc-dimV x noc-dimV) and has its eigenvalues set to beta(i)'s. The pair (A44,B42) is controllable. A55 is a stable (nos-noc x nos-noc) matrix. A66 is an unstable (nx-nos x nx-nos) matrix (see st_ility).

Z is a column compression of Sys and k is the normal rank of Sys i.e Sys*Z is a column-compressed linear system. k is the column dimensions of B42,B52,B62 and D2. [B42;B52;B62;D2] is full column rank and has rank k.

If flag='st' is given, a five blocks partition of the matrices is returned and dims has four components. If flag='pp' is given a four blocks partition is returned. In case flag='ge' one has dims=[dimR,dimVg,dimV,dimV+nc2,dimV+ns2] where nc2 (resp. ns2) is the dimension of the controllable (resp. stabilizable) pair (A44,B42) (resp. ([A44,*;0,A55], [B42;0])). In case flag='st' one has dims=[dimR,dimVg,dimVg+nc,dimVg+ns] and in case flag='pp' one has dims=[dimR,dimR+nc,dimR+ns]. nc (resp. ns) is here the dimension of the controllable (resp. stabilizable) subspace of the blocks 3 to 6 (resp. 2 to 6).

This function can be used for the (exact) disturbance decoupling problem.

```
DDPS:
   Find u=Fx+Rd=[F,R]*[x;d] which rejects Q*d and stabilizes the plant:

    xdot = Ax+Bu+Qd
       y = Cx+Du+Td

DDPS has a solution if Im(Q) is included in Vg + Im(B) and stabilizability
assumption is satisfied.
Let G=(X(:,dimVg+1:$))'= left annihilator of Vg i.e. G*Vg=0;
B2=G*B; Q2=G*Q; DDPS solvable iff [B2;D]*R + [Q2;T] =0 has a solution.
```

```
The pair F,R  is the solution  (with F=output of abinv).
Im(Q2) is in Im(B2) means row-compression of B2=>row-compression of Q2
Then C*[(sI-A-B*F)^(-1)+D]*(Q+B*R) =0   (<=>G*(Q+B*R)=0)
```

Examples

```
nu=3;ny=4;nx=7;
nrt=2;ngt=3;ng0=3;nvt=5;rk=2;
flag=list('on',nrt,ngt,ng0,nvt,rk);
Sys=ssrand(ny,nu,nx,flag);my_alpha=-1;my_beta=-2;
[X,dims,F,U,k,Z]=abinv(Sys,my_alpha,my_beta);
[A,B,C,D]=abcd(Sys);dimV=dims(3);dimR=dims(1);
V=X(:,1:dimV);X2=X(:,dimV+1:nx);
X2'*(A+B*F)*V
(C+D*F)*V
X0=X(:,dimR+1:dimV); spec(X0'*(A+B*F)*X0)
trzeros(Sys)
spec(A+B*F)   //nr=2 evals at -1 and noc-dimV=2 evals at -2.
clean(ss2tf(Sys*Z))
// 2nd Example
nx=6;ny=3;nu=2;
A=diag(1:6);A(2,2)=-7;A(5,5)=-9;B=[1,2;0,3;0,4;0,5;0,0;0,0];
C=[zeros(ny,ny),eye(ny,ny)];D=[0,1;0,2;0,3];
sl=syslin('c',A,B,C,D);//sl=ss2ss(sl,rand(6,6))*rand(2,2);
[A,B,C,D]=abcd(sl);  //The matrices of sl.
my_alpha=-1;my_beta=-2;
[X,dims,F,U,k,Z]=abinv(sl,my_alpha,my_beta);dimVg=dims(2);
clean(X'*(A+B*F)*X)
clean(X'*B*U)
clean((C+D*F)*X)
clean(D*U)
G=(X(:,dimVg+1:$))';
B2=G*B;nd=3;
R=rand(nu,nd);Q2T=-[B2;D]*R;
p=size(G,1);Q2=Q2T(1:p,:);T=Q2T(p+1:$,:);
Q=G\Q2;   //a valid [Q;T] since
[G*B;D]*R + [G*Q;T]  // is zero
closed=syslin('c',A+B*F,Q+B*R,C+D*F,T+D*R); // closed loop: d-->y
ss2tf(closed)       // Closed loop is zero
spec(closed('A'))   //The plant is not stabilizable!
[ns,nc,W,sl1]=st_ility(sl);
[A,B,C,D]=abcd(sl1);A=A(1:ns,1:ns);B=B(1:ns,:);C=C(:,1:ns);
slnew=syslin('c',A,B,C,D);  //Now stabilizable
//Fnew=stabil(slnew('A'),slnew('B'),-11);
//slnew('A')=slnew('A')+slnew('B')*Fnew;
//slnew('C')=slnew('C')+slnew('D')*Fnew;
[X,dims,F,U,k,Z]=abinv(slnew,my_alpha,my_beta);dimVg=dims(2);
[A,B,C,D]=abcd(slnew);
G=(X(:,dimVg+1:$))';
B2=G*B;nd=3;
R=rand(nu,nd);Q2T=-[B2;D]*R;
p=size(G,1);Q2=Q2T(1:p,:);T=Q2T(p+1:$,:);
Q=G\Q2;   //a valid [Q;T] since
[G*B;D]*R + [G*Q;T]  // is zero
```

```
closed=syslin('c',A+B*F,Q+B*R,C+D*F,T+D*R); // closed loop: d-->y
ss2tf(closed)       // Closed loop is zero
spec(closed('A'))
```

See Also

cainv , st_ility , ssrand , ss2ss , ddp

Authors

F.D.

Name

arhnk — Hankel norm approximant

```
[slm]=arhnk(sl,ord,[tol])
```

Parameters

sl
: linear system (`syslin` list)

ord
: integer, order of the approximant

tol
: threshold for rank determination in `equil1`

Description

computes `slm`, the optimal Hankel norm approximant of the stable continuous-time linear system `sl` with matrices `[A,B,C,D]`.

Examples

```
A=diag([-1,-2,-3,-4,-5]);B=rand(5,1);C=rand(1,5);
sl=syslin('c',A,B,C);
slapprox=arhnk(sl,2);
[nk,W]=hankelsv(sl);nk
[nkred,Wred]=hankelsv(slapprox);nkred
```

See Also

equil , equil1 , hankelsv

Name

arl2 — SISO model realization by L2 transfer approximation

```
h=arl2(y,den0,n [,imp])
h=arl2(y,den0,n [,imp],'all')
[den,num,err]=arl2(y,den0,n [,imp])
[den,num,err]=arl2(y,den0,n [,imp],'all')
```

Parameters

y
: real vector or polynomial in `z^-1`, it contains the coefficients of the Fourier's series of the rational system to approximate (the impulse response)

den0
: a polynomial which gives an initial guess of the solution, it may be `poly(1,'z','c')`

n
: integer, the degree of approximating transfer function (degree of den)

imp
: integer in `(0,1,2)` (verbose mode)

h
: transfer function `num/den` or transfer matrix (column vector) when flag `'all'` is given.

den
: polynomial or vector of polynomials, contains the denominator(s) of the solution(s)

num
: polynomial or vector of polynomials, contains the numerator(s) of the solution(s)

err
: real constant or vector , the l2-error achieved for each solutions

Description

`[den,num,err]=arl2(y,den0,n [,imp])` finds a pair of polynomials `num` and `den` such that the transfer function `num/den` is stable and its impulse response approximates (with a minimal l2 norm) the vector `y` assumed to be completed by an infinite number of zeros.

If `y(z) = y(1)(1/z)+y(2)(1/z^2)+ ...+ y(ny)(1/z^ny)`

then l2-norm of `num/den - y(z)` is `err`.

`n` is the degree of the polynomial `den`.

The `num/den` transfer function is a L2 approximant of the Fourier's series of the rational system.

Various intermediate results are printed according to `imp`.

`[den,num,err]=arl2(y,den0,n [,imp],'all')` returns in the vectors of polynomials `num` and `den` a set of local optimums for the problem. The solutions are sorted with increasing errors `err`. In this case `den0` is already assumed to be `poly(1,'z','c')`

Examples

```
v=ones(1,20);
xbasc();
plot2d1('enn',0,[v';zeros(80,1)],2,'051',' ',[1,-0.5,100,1.5])

[d,n,e]=arl2(v,poly(1,'z','c'),1)
plot2d1('enn',0,ldiv(n,d,100),2,'000')
[d,n,e]=arl2(v,d,3)
plot2d1('enn',0,ldiv(n,d,100),3,'000')
[d,n,e]=arl2(v,d,8)
plot2d1('enn',0,ldiv(n,d,100),5,'000')

[d,n,e]=arl2(v,poly(1,'z','c'),4,'all')
plot2d1('enn',0,ldiv(n(1),d(1),100),10,'000')
```

See Also

ldiv , imrep2ss , time_id , armax , frep2tf

Name

arma — Scilab arma library

Description

Armax processes can be coded with Scilab tlist of type `'ar'`. `armac` is used to build `Armax` scilab object. An `'ar'` tlist contains the fields `['a','b','d','ny','nu','sig']`.

armac
: this function creates a Scilab tlist which code an Armax process `A(z^-1)y= B(z^-1)u + D(z^-1)sig*e(t)`

```
-->ar=armac([1,2],[3,4],1,1,1,sig);

-->ar('a')
 ans  =

!   1.    2. !
-->ar('sig')
 ans  =

   1.
```

armap(ar [,out])
: Display the armax equation associated with `ar`

armap_p(ar [,out])
: Display the armax equation associated with `ar` using polynomial matrix display.

[A,B,D]=armap2p(ar)
: extract polynomial matrices from ar representation

armax
: is used to identify the coefficients of a n-dimensional ARX process `A(z^-1)y= B(z^-1)u + sig*e(t)`

armax1
: armax1 is used to identify the coefficients of a 1-dimensional ARX process `A(z^-1)y= B(z^-1)u + D(z^-1)sig*e(t)`

arsimul
: armax trajectory simulation.

narsimul
: armax simulation (using rtitr)

odedi
: Simple tests of ode and arsimul. Tests the option 'discret' of ode

prbs_a
: pseudo random binary sequences generation

reglin
: Linear regression

Authors

J.P.C ; ;

Name

arma2p — extract polynomial matrices from ar representation

```
[A,B,D]=arma2p(ar)
```

Parameters

A,B,D
: three polynomial matrices

ar
: Scilab 'ar' tlist for arma storage (see `armac`).

Description

this function extract polynomial matrices `(A,B,D)` from an armax description.

Examples

```
a=[1,-2.851,2.717,-0.865].*.eye(2,2)
b=[0,1,1,1].*.[1;1];
d=[1,0.7,0.2].*.eye(2,2);
sig=eye(2,2);
ar=armac(a,b,d,2,1,sig)
// extract polynomial matrices from ar representation
[A,B,D]=arma2p(ar);
```

See Also

arma , armax , armax1 , arsimul , armac

Name

armac — Scilab description of an armax process

```
[ar]=armac(a,b,d,ny,nu,sig)
```

Parameters

a=[Id,a1,..,a_r]
: is a matrix of size (ny,r*ny)

b=[b0,.....,b_s]
: is a matrix of size (ny,(s+1)*nu)

d=[Id,d1,..,d_p]
: is a matrix of size (ny,p*ny);

ny
: dimension of the output y

nu
: dimension of the output u

sig
: a matrix of size (ny,ny)

Description

This function creates a description as a tlist of an ARMAX process

`ar` is defined by

```
ar=tlist(['ar','a','b','d','ny','nu','sig'],a,b,d,ny,nu,sig);
```

and thus the coefficients of `ar` can be retrieved by e.g. `ar('a')` .

Examples

```
a=[1,-2.851,2.717,-0.865].*.eye(2,2)
b=[0,1,1,1].*.[1;1];
d=[1,0.7,0.2].*.eye(2,2);
sig=eye(2,2);
ar=armac(a,b,d,2,1,sig)
// extract polynomial matrices from ar representation
[A,B,D]=arma2p(ar);
```

See Also

arma , armax , armax1 , arsimul , arma2p , tlist

Name

armax — armax identification

```
[arc,la,lb,sig,resid]=armax(r,s,y,u,[b0f,prf])
```

Parameters

y
: output process y(ny,n); (ny: dimension of y , n : sample size)

u
: input process u(nu,n); (nu: dimension of u , n : sample size)

r and s
: auto-regression orders r >=0 et s >=-1

b0f
: optional parameter. Its default value is 0 and it means that the coefficient b0 must be identified. if bof=1 the b0 is supposed to be zero and is not identified

prf
: optional parameter for display control. If prf =1, the default value, a display of the identified Arma is given.

arc
: a Scilab arma object (see armac)

la
: is the list(a,a+eta,a-eta) (la = a in dimension 1) ; where eta is the estimated standard deviation. , a=[Id,a1,a2,...,ar] where each ai is a matrix of size (ny,ny)

lb
: is the list(b,b+etb,b-etb) (lb =b in dimension 1) ; where etb is the estimated standard deviation. b=[b0,.....,b_s] where each bi is a matrix of size (nu,nu)

sig
: is the estimated standard deviation of the noise and resid=[sig*e(t0),....] (

Description

armax is used to identify the coefficients of a n-dimensional ARX process

```
A(z^-1)y= B(z^-1)u + sig*e(t)
```

where e(t) is a n-dimensional white noise with variance I. sig an nxn matrix and A(z) and B(z):

```
A(z) = 1+a1*z+...+a_r*z^r; ( r=0 => A(z)=1)
B(z) = b0+b1*z+...+b_s z^s ( s=-1 => B(z)=0)
```

for the method see Eykhoff in trends and progress in system identification, page 96. with `z(t)=[y(t-1),..,y(t-r),u(t),...,u(t-s)]` and `coef= [-a1,..,-`

ar,b0,...,b_s] we can write y(t)= coef* z(t) + sig*e(t) and the algorithm minimises sum_{t=1}^N ([y(t)- coef'z(t)]^2) where t0=maxi(maxi(r,s)+1,1))).

Examples

```
//-Ex1- Arma model : y(t) = 0.2*u(t-1)+0.01*e(t-1)
ny=1,nu=1,sig=0.01;
Arma=armac(1,[0,0.2],[0,1],ny,nu,sig)  //defining the above arma model
u=rand(1,1000,'normal');     //a random input sequence u
y=arsimul(Arma,u); //simulation of a y output sequence associated with u.
Armaest=armax(0,1,y,u);   //Identified model given u and y.
Acoeff=Armaest('a');    //Coefficients of the polynomial A(x)
Bcoeff=Armaest('b')     //Coefficients of the polynomial B(x)
Dcoeff=Armaest('d');    //Coefficients of the polynomial D(x)
[Ax,Bx,Dx]=arma2p(Armaest)   //Results in polynomial form.

//-Ex2- Arma1: y_t -0.8*y_{t-1} + 0.2*y_{t-2} =  sig*e(t)
ny=1,nu=1;sig=0.001;
// First step: simulation the Arma1 model, for that we define
// Arma2: y_t -0.8*y_{t-1} + 0.2*y_{t-2} = sig*u(t)
// with normal deviates for u(t).
Arma2=armac([1,-0.8,0.2],sig,0,ny,nu,0);
//Definition of the Arma2 arma model (a model with B=sig and without noise!)
u=rand(1,10000,'normal');  // An input sequence for Arma2
y=arsimul(Arma2,u); // y = output of Arma2 with input u
//                     can be seen as output of Arma1.
// Second step: identification. We look for an Arma model
// y(t) + a1*y(t-1) + a2 *y(t-2) =  sig*e(t)
Arma1est=armax(2,-1,y,[]);
[A,B,D]=arma2p(Arma1est)
```

See Also

imrep2ss , time_id , arl2 , armax , frep2tf

Authors

J-Ph. Chancelier.

Name

armax1 — armax identification

```
[arc,resid]=armax1(r,s,q,y,u [,b0f])
```

Parameters

y
: output signal

u
: input signal

r,s,q
: auto regression orders with r >=0, s >=-1.

b0f
: optional parameter. Its default value is 0 and it means that the coefficient b0 must be identified. if bof=1 the b0 is supposed to be zero and is not identified

arc
: is tlist with type "ar" and fields a, b, d, ny, nu, sig

  a
  : is the vector `[1,a1,...,a_r]`

  b
  : is the vector `[b0,.....,b_s]`

  d
  : is the vector `[1,d1,....,d_q]`

  sig
  : resid=[sig*echap(1),....,];

Description

armax1 is used to identify the coefficients of a 1-dimensional ARX process:

```
A(z^-1)y= B(z^-1)u + D(z^-1)sig*e(t)
e(t) is a 1-dimensional white noise with variance 1.
A(z)= 1+a1*z+...+a_r*z^r; ( r=0 => A(z)=1)
B(z)= b0+b1*z+...+b_s z^s ( s=-1 => B(z)=0)
D(z)= 1+d1*z+...+d_q*z^q  ( q=0 => D(z)=1)
```

for the method, see Eykhoff in trends and progress in system identification) page 96. with

```
          z(t)=[y(t-1),..,y(t-r),u(t),...,
                u(t-s),e(t-1),...,e(t-q)]
```

and

```
            coef= [-a1,..,-ar,b0,...,b_s,d1,...,d_q]'
            y(t)= coef'* z(t) + sig*e(t).
```

a sequential version of the AR estimation where e(t-i) is replaced by an estimated value is used (RLLS). With q=0 this method is exactly a sequential version of armax

Important notice

In Scilab versions up to 4.1.2 the returned value in `arc.sig` is the square of `sig` square. To be conform with the help, the display of arma models and the armax function, starting from Scilab-5.0 version the returned `arc.sig` is `sig`.

Authors

J.-Ph.C; ;

Name

arsimul — armax simulation

```
[z]=arsimul(a,b,d,sig,u,[up,yp,ep])
[z]=arsimul(ar,u,[up,yp,ep])
```

Parameters

ar
: an armax process. See armac.

a
: is the matrix `[Id,a1,...,a_r]` of dimension (n,(r+1)*n)

b
: is the matrix `[b0,.....,b_s]` of dimension (n,(s+1)*m)

d
: is the matrix `[Id,d_1,.....,d_t]` of dimension (n,(t+1)*n)

u
: is a matrix (m,N), which gives the entry u(:,j)=u_j

sig
: is a (n,n) matrix e_{k} is an n-dimensional Gaussian process with variance I

up, yp
: optional parameter which describe the past. `up=[ u_0,u_{-1},...,u_{s-1}]`; `yp=[ y_0,y_{-1},...,y_{r-1}];ep=[ e_0,e_{-1},...,e_{r-1}]`; if they are omitted, the past value are supposed to be zero

z
: `z=[y(1),....,y(N)]`

Description

simulation of an n-dimensional armax process `A(z^-1) z(k)= B(z^-1)u(k) + D(z^-1)*sig*e(k)`

```
      A(z)= Id+a1*z+...+a_r*z^r;  ( r=0  => A(z)=Id)
      B(z)= b0+b1*z+...+b_s z^s;  ( s=-1 => B(z)=[])
      D(z)= Id+d1*z+...+d_t z^t;  ( t=0  => D(z)=Id)
```

z et e are in `R^n` et u in `R^m`

Method

a state-space representation is constructed and ode with the option "discr" is used to compute z

Authors

J-Ph.C.

Name

augment — augmented plant

```
[P,r]=augment(G)
[P,r]=augment(G,flag1)
[P,r]=augment(G,flag1,flag2)
```

Parameters

G
: linear system (`syslin` list), the nominal plant

flag1
: one of the following (upper case) character string: `'S'` , `'R'` , `'T'` `'SR'` , `'ST'` , `'RT'` `'SRT'`

flag2
: one of the following character string: `'o'` (stands for 'output', this is the default value) or `'i'` (stands for 'input').

P
: linear system (`syslin` list), the ``augmented'' plant

r
: 1x2 row vector, dimension of `P22 = G`

Description

If `flag1='SRT'` (default value), returns the "full" augmented plant

```
    [ I | -G]   -->'S'
    [ 0 |  I]   -->'R'
P = [ 0 |  G]   -->'T'
    [-------]
    [ I | -G]
```

`'S'` , `'R'` , `'T'` refer to the first three (block) rows of `P` respectively.

If one of these letters is absent in `flag1`, the corresponding row in `P` is missing.

If `G` is given in state-space form, the returned `P` is minimal. `P` is calculated by: `[I,0,0;0,I,0;-I,0,I;I,0,0]*[I,-G;0,I;I,0]`.

The augmented plant associated with input sensitivity functions, namely

```
    [ I | -I]   -->'S'  (input sensitivity)
    [ G | -G]   -->'R'  (G*input sensitivity)
P = [ 0 |  I]   -->'T'  (K*G*input sensitivity)
    [-------]
    [ G | -G]
```

is obtained by the command `[P,r]=augment(G,flag,'i')`. For state-space `G`, this `P` is calculated by: `[I,-I;0,0;0,I;0,0]+[0;I;0;I]*G*[I,-I]` and is thus generically minimal.

Note that weighting functions can be introduced by left-multiplying `P` by a diagonal system of appropriate dimension, e.g., `P = sysdiag(W1,W2,W3,eye(G))*P`.

Sensitivity functions can be calculated by `lft`. One has:

For output sensitivity functions [P,r]=augment(P,'SRT'): lft(P,r,K)=[inv(eye()+G*K);K*inv(eye()+G*K);G*K*inv(eye()+G*K)];

For input sensitivity functions [P,r]=augment(P,'SRT','i'): lft(P,r,K)=[inv(eye()+K*G);G*inv(eye()+K*G);K*G*inv(eye()+G*K)];

Examples

```
G=ssrand(2,3,2); //Plant
K=ssrand(3,2,2); //Compensator
[P,r]=augment(G,'T');
T=lft(P,r,K);   //Complementary sensitivity function
Ktf=ss2tf(K);Gtf=ss2tf(G);
Ttf=ss2tf(T);T11=Ttf(1,1);
Oloop=Gtf*Ktf;
Tn=Oloop*inv(eye(Oloop)+Oloop);
clean(T11-Tn(1,1));
//
[Pi,r]=augment(G,'T','i');
T1=lft(Pi,r,K);T1tf=ss2tf(T1); //Input Complementary sensitivity function
Oloop=Ktf*Gtf;
T1n=Oloop*inv(eye(Oloop)+Oloop);
clean(T1tf(1,1)-T1n(1,1))
```

See Also

lft , sensi

Name

balreal — balanced realization

```
[slb [,U] ] = balreal(sl)
```

Parameters

sl,slb
: linear systems (`syslin` lists)

Description

Balanced realization of linear system `sl=[A,B,C,D]`. `sl` can be a continuous-time or discrete-time state-space system. sl is assumed stable.

```
slb=[inv(U)*A*U ,inv(U)*B , C*U , D]
```

is the balanced realization.

`slb` is returned as a `syslin` list.

Examples

```
A=diag([-1,-2,-3,-4,-5]);B=rand(5,2);C=rand(1,5);
sl=syslin('c',A,B,C);
[slb,U]=balreal(sl);
Wc=clean(ctr_gram(slb))
W0=clean(obs_gram(slb))
```

See Also

ctr_gram , obs_gram , hankelsv , equil , equil1

Name

bilin — general bilinear transform

```
[sl1]=bilin(sl,v)
```

Parameters

sl,sl1
: linear systems (`syslin` lists)

v
: real vector with 4 entries (`v=[a,b,c,d]`)

Description

Given a linear system in state space form, `sl=syslin(dom,A,B,C,D)` (`syslin` list), `sl1=bilin(sl,v)` returns in `sl1` a linear system with matrices `[A1,B1,C1,D1]` such that the transfer function `H1(s)=C1*inv(s*eye()-A1)*B1+D1` is obtained from `H(z)=C*inv(z*eye()-A)*B+D` by replacing `z` by `z=(a*s+b)/(c*s+d)`. One has `w=bilin(bilin(w,[a,b,c,d]),[d,-b,-c,a])`

Examples

```
s=poly(0,'s');z=poly(0,'z');
w=ssrand(1,1,3);
wtf=ss2tf(w);v=[2,3,-1,4];a=v(1);b=v(2);c=v(3);d=v(4);
[horner(wtf,(a*z+b)/(c*z+d)),ss2tf(bilin(w,[a,b,c,d]))]
clean(ss2tf(bilin(bilin(w,[a,b,c,d]),[d,-b,-c,a]))-wtf)
```

See Also

horner , cls2dls

Name

black — Black's diagram (Nichols chart)

```
black( sl,[fmin,fmax] [,step] [,comments] )
black( sl,frq [,comments] )
black(frq,db,phi [,comments])
black(frq,repf [,comments])
```

Parameters

sl
: list (linear system `syslin`)

fmin,fmax
: real scalars (frequency bounds)

frq
: row vector or matrix (frequencies)

db,phi
: row vectors or matrices (modulus, phase)

repf
: row vectors or matrices (complex frequency response)

step
: real

comments
: string

Description

Black's diagram (Nichols'chart) for a linear system `sl`. `sl` can be a continuous-time or discrete-time SIMO system (see `syslin`). In case of multi-output the outputs are plotted with different symbols.

The frequencies are given by the bounds `fmin`,`fmax` (in Hz) or by a row-vector (or a matrix for multi-output) `frq`.

`step` is the (logarithmic) discretization step. (see `calfrq` for the choice of default value).

`comments` is a vector of character strings (captions).

`db,phi` are the matrices of modulus (in Db) and phases (in degrees). (One row for each response).

`repf` matrix of complex numbers. One row for each response.

To plot the grid of iso-gain and iso-phase of `y/(1+y)` use chart().

Default values for `fmin` and `fmax` are `1.d-3`, `1.d+3` if `sl` is continuous-time or `1.d-3`, `0.5`/sl.dt (nyquist frequency) if `sl` is discrete-time.

Examples

```


s=poly(0,'s')
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
```

```
clf();black(h,0.01,100);
chart(list(1,0));

h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
clf()
black([h1;h],0.01,100,['h1';'h'])
chart(list(1,0));
```

See Also

bode, nyquist, chart, freq, repfreq, calfrq, phasemag

Name

bode — Bode plot

```
bode(sl,[fmin,fmax] [,step] [,comments] )
bode(sl,frq [,comments] )
bode(frq,db,phi [,comments])
bode(frq, repf [,comments])
```

Parameters

sl
: `syslin` list (SISO or SIMO linear system) in continuous or discrete time.

fmin,fmax
: real (frequency bounds (in Hz))

step
: real (logarithmic step.)

comments
: vector of character strings (captions).

frq
: row vector or matrix (frequencies (in Hz)) (one row for each SISO subsystem).

db
: row vector or matrix (magnitudes (in Db)). (one row for each SISO subsystem).

phi
: row vector or matrix (phases (in degree)) (one row for each SISO subsystem).

repf
: row vector or matrix of complex numbers (complex frequency response).

Description

Bode plot, i.e magnitude and phase of the frequency response of `sl`.

`sl` can be a continuous-time or discrete-time SIMO system (see `syslin`). In case of multi-output the outputs are plotted with different symbols.

The frequencies are given by the bounds `fmin,fmax` (in Hz) or by a row-vector (or a matrix for multi-output) `frq`.

`step` is the (logarithmic) discretization step. (see `calfrq` for the choice of default value).

`comments` is a vector of character strings (captions).

`db,phi` are the matrices of modulus (in Db) and phases (in degrees). (One row for each response).

`repf` matrix of complex numbers. One row for each response.

Default values for `fmin` and `fmax` are `1.d-3`, `1.d+3` if `sl` is continuous-time or `1.d-3`, `0.5`/sl.dt (nyquist frequency) if `sl` is discrete-time. Automatic discretization of frequencies is made by `calfrq`.

Examples

```
s=poly(0,'s')
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
tit='(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)';
bode(h,0.01,100,tit);
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
clf()
bode([h1;h],0.01,100,['h1';'h'])
```

See Also

black, nyquist, gainplot, repfreq, g_margin, p_margin, calfrq, phasemag

Name

bstap — hankel approximant

```
[Q]=bstap(Sl)
```

Parameters

sl
: linear system (`syslin` list) assumed continuous-time and anti-stable.

Q
: best stable approximation of `Sl` (`syslin` list).

Description

Computes the best approximant `Q` of the linear system `Sl`

where

||T||

is the H-infinity norm of the Hankel operator associated with `Sl`.

See Also

syslin

Name

cainv — Dual of abinv

```
[X,dims,J,Y,k,Z]=cainv(Sl,alfa,beta,flag)
```

Parameters

Sl
: `syslin` list containing the matrices `[A,B,C,D]`.

alfa
real number or vector (possibly complex, location of closed loop poles)

beta
real number or vector (possibly complex, location of closed loop poles)

flag
(optional) character string `'ge'` (default) or `'st'` or `'pp'`

X
orthogonal matrix of size nx (dim of state space).

dims
integer row vector `dims=[nd1,nu1,dimS,dimSg,dimN]` (5 entries, nondecreasing order).If `flag='st'`, (resp. `'pp'`), `dims` has 4 (resp. 3) components.

J
real matrix (output injection)

Y
orthogonal matrix of size ny (dim of output space).

k
integer (normal rank of `Sl`)

Z
non-singular linear system (`syslin` list)

Description

`cainv` finds a bases `(X,Y)` (of state space and output space resp.) and output injection matrix `J` such that the matrices of Sl in bases (X,Y) are displayed as:

```
                [A11,*,*,*,*,*]                 [*]
                [0,A22,*,*,*,*]                 [*]
X'*(A+J*C)*X =  [0,0,A33,*,*,*]   X'*(B+J*D) =  [*]
                [0,0,0,A44,*,*]                 [0]
                [0,0,0,0,A55,*]                 [0]
                [0,0,0,0,0,A66]                 [0]

       Y*C*X =  [0,0,C13,*,*,*]          Y*D =  [*]
                [0,0,0,0,0,C26]                 [0]
```

The partition of X is defined by the vector dims=[nd1,nu1,dimS,dimSg,dimN] and the partition of Y is determined by k.

Eigenvalues of A11 (nd1 x nd1) are unstable. Eigenvalues of A22 (nu1-nd1 x nu1-nd1) are stable.

The pair (A33, C13) (dimS-nu1 x dimS-nu1, k x dimS-nu1) is observable, and eigenvalues of A33 are set to alfa.

Matrix A44 (dimSg-dimS x dimSg-dimS) is unstable. Matrix A55 (dimN-dimSg,dimN-dimSg) is stable

The pair (A66,C26) (nx-dimN x nx-dimN) is observable, and eigenvalues of A66 set to beta.

The dimS first columns of X span S= smallest (C,A) invariant subspace which contains Im(B), dimSg first columns of X span Sg the maximal "complementary detectability subspace" of Sl

The dimN first columns of X span the maximal "complementary observability subspace" of Sl. (dimS=0 if B(ker(D))=0).

If flag='st' is given, a five blocks partition of the matrices is returned and dims has four components. If flag='pp' is given a four blocks partition is returned (see abinv).

This function can be used to calculate an unknown input observer:

```
// DDEP: dot(x)=A x + Bu + Gd
//           y= Cx   (observation)
//           z= Hx    (z=variable to be estimated, d=disturbance)
//  Find: dot(w) = Fw + Ey + Ru such that
//          zhat = Mw + Ny
//           z-Hx goes to zero at infinity
//  Solution exists iff Ker H contains Sg(A,C,G) inter KerC (assuming detectabi
//i.e. H is such that:
// For any W which makes a column compression of [Xp(1:dimSg,:);C]
// with Xp=X' and [X,dims,J,Y,k,Z]=cainv(syslin('c',A,G,C));
// [Xp(1:dimSg,:);C]*W = [0 | *] one has
// H*W = [0 | *]  (with at least as many aero columns as above).
```

See Also

abinv , dt_ility , ui_observer

Name

calfrq — frequency response discretization

```
[frq,bnds,split]=calfrq(h,fmin,fmax)
```

Parameters

h
: Linear system in state space or transfer representation (`see syslin`)

fmin,fmax
: real scalars (min and max frequencies in Hz)

frq
: row vector (discretization of the frequency interval)

bnds
: vector `[Rmin Rmax Imin Imax]` where `Rmin` and `Rmax` are the lower and upper bounds of the frequency response real part, `Imin` and `Imax` are the lower and upper bounds of the frequency response imaginary part,

split
: vector of frq splitting points indexes

Description

frequency response discretization; `frq` is the discretization of `[fmin,fmax]` such that the peaks in the frequency response are well represented.

Singularities are located between `frq(split(k)-1)` and `frq(split(k))` for `k>1`.

Examples

```
s=poly(0,'s')
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
[f1,bnds,spl]=calfrq(h1,0.01,1000);
rf=repfreq(h1,f1);
plot2d(real(rf)',imag(rf)')
```

See Also

bode, black, nyquist, freq, repfreq, logspace

Name

canon — canonical controllable form

```
[Ac,Bc,U,ind]=canon(A,B)
```

Parameters

Ac,Bc
: canonical form

U
: current basis (square nonsingular matrix)

ind
: vector of integers, controllability indices

Description

gives the canonical controllable form of the pair `(A,B)`.

`Ac=inv(U)*A*U, Bc=inv(U)*B`

The vector `ind` is made of the `epsilon_i`'s indices of the pencil `[sI - A , B]` (decreasing order). For example with `ind=[3,2]`, `Ac` and `Bc` are as follows:

```
      [*,*,*,*,*]           [*]
      [1,0,0,0,0]           [0]
Ac=   [0,1,0,0,0]        Bc=[0]
      [*,*,*,*,*]           [*]
      [0,0,0,1,0]           [0]
```

If `(A,B)` is controllable, by an appropriate choice of `F` the `*` entries of `Ac+Bc*F` can be arbitrarily set to desired values (pole placement).

Examples

```
A=[1,2,3,4,5;
   1,0,0,0,0;
   0,1,0,0,0;
   6,7,8,9,0;
   0,0,0,1,0];
B=[1,2;
   0,0;
   0,0;
   2,1;
   0,0];
X=rand(5,5);A=X*A*inv(X);B=X*B;    //Controllable pair
[Ac,Bc,U,ind]=canon(A,B);  //Two indices --> ind=[3.2];
index=1;for k=1:size(ind,'*')-1,index=[index,1+sum(ind(1:k))];end
Acstar=Ac(index,:);Bcstar=Bc(index,:);
s=poly(0,'s');
```

```
p1=s^3+2*s^2-5*s+3;p2=(s-5)*(s-3);
//p1 and p2 are desired closed-loop polynomials with degrees 3,2
c1=coeff(p1);c1=c1($-1:-1:1);c2=coeff(p2);c2=c2($-1:-1:1);
Acstardesired=[-c1,0,0;0,0,0,-c2];
//Acstardesired(index,:) is companion matrix with char. pol=p1*p2
F=Bcstar\(Acstardesired-Acstar);   //Feedbak gain
Ac+Bc*F         // Companion form
spec(A+B*F/U)   // F/U is the gain matrix in original basis.
```

See Also

obsv_mat , cont_mat , ctr_gram , contrss , ppol , contr , stabil

Authors

F.D.

Name

ccontrg — central H-infinity controller

```
[K]=ccontrg(P,r,gamma);
```

Parameters

P
: `syslin` list (linear system in state-space representation)

r
: 1x2 row vector, dimension of the 2,2 part of `P`

gamma
: real number

Description

returns a realization `K` of the central controller for the general standard problem in state-space form.

Note that gamma must be > gopt (ouput of `gamitg`)

P contains the parameters of plant realization `(A,B,C,D)` (`syslin` list) with

```
B = ( B1 , B2 ) ,        C= ( C1 ) ,    D = ( D11  D12)
                            ( C2 )          ( D21  D22)
```

`r(1)` and `r(2)` are the dimensions of `D22` (rows x columns)

See Also

gamitg , h_inf

Authors

P. Gahinet (INRIA);

Name

chart — Nichols chart

```
chart([flags])
chart(gain [,flags])
chart(gain,phase [,flags])
```

Parameters

gain
: real vector (gains (in DB))

phase
: real vector (phases (in degree))

flags
: a list of at most 4 flags list(sup [,leg [,cm [,cphi]]])

  sup
  : 1 indicates superposition on the previous plot 0 no superposition is done

  leg
  : 1 indicates that legends are drawn, o: no legends

  cm
  : color index for gain curves

  cphi
  : color index for phase curves

Description

plot the Nichols'chart: iso-gain and iso-phase contour of y/(1+y) in phase/gain plane

`chart` may be used in cunjunction with black.

The default values for `gain` and `phase` are respectively :

```
[-12 -8 -6 -5 -4 -3 -2 -1.4 -1 -.5 0.25 0.5 0.7 1 1.4 2 2.3 3 4 5 6 8 12]
```

```
[-(1:10) , -(20:10:160)]
```

Examples

```
s=poly(0,'s')
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
black(h,0.01,100)
chart(list(1,0,2,3));

clf()
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
black([h1;h],0.01,100,['h1';'h'])
set(gca(),'data_bounds',[-180 -30;180 30]) //enlarge the frame
chart(list(1,0));
```

See Also

nyquist , black

Name

cls2dls — bilinear transform

```
[sl1]=cls2dls(sl,T [,fp])
```

Parameters

sl,sl1
: linear systems (`syslin` lists)

T
: real number, the sampling period

fp
: prevarping frequency in hertz

Description

given `sl=[A,B,C,D]` (`syslin` list),a continuous time system `cls2dls` returns the sampled system obtained by the bilinear transform `s=(2/T)*(z-1)/(z+1)`.

Examples

```
s=poly(0,'s');z=poly(0,'z');
sl=syslin('c',(s+1)/(s^2-5*s+2));  //Continuous-time system in transfer form
slss=tf2ss(sl);  //Now in state-space form
sl1=cls2dls(slss,0.2);  //sl1= output of cls2dls
sl1t=ss2tf(sl1) // Converts in transfer form
sl2=horner(sl,(2/0.2)*(z-1)/(z+1))   //Compare sl2 and sl1
```

See Also

horner

Name

colinout — inner-outer factorization

```
[Inn,X,Gbar]=colinout(G)
```

Parameters

G
: linear system (`syslin` list) `[A,B,C,D]`

Inn
: inner factor (`syslin` list)

Gbar
: outer factor (`syslin` list)

X
: row-compressor of `G` (`syslin` list)

Description

Inner-outer factorization (and column compression) of (lxp) `G =[A,B,C,D]` with l<=p.

`G` is assumed to be fat (`l<=p`) without zero on the imaginary axis and with a `D` matrix which is full row rank.

`G` must also be stable for having `Gbar` stable.

Dual of `rowinout`.

See Also

syslin , rowinout

Name

colregul — removing poles and zeros at infinity

```
[Stmp,Ws]=colregul(Sl,alfa,beta)
```

Parameters

Sl,Stmp
: `syslin` lists

alfa,beta
: reals (new pole and zero positions)

Description

computes a prefilter `Ws` such that `Stmp=Sl*Ws` is proper and with full rank `D` matrix.

Poles at infinity of `Sl` are moved to `alfa`;

Zeros at infinity of `Sl` are moved to `beta`;

`Sl` is a assumed to be a left invertible linear system (`syslin` list) in state-space representation with possibly a polynomial `D` matrix.

See Also

invsyslin , inv , rowregul , rowshuff

Authors

F. D. , R. N. ;

Name

cont_frm — transfer to controllable state-space

```
[sl]=cont_frm(NUM,den)
```

Parameters

NUM
: polynomial matrix

den
: polynomial

sl
: `syslin` list, `sl=[A,B,C,D]`.

Description

controllable state-space form of the transfer `NUM/den`.

Examples

```
s=poly(0,'s');NUM=[1+s,s];den=s^2-5*s+1;
sl=cont_frm(NUM,den);
slss=ss2tf(sl);       //Compare with NUM/den
```

See Also

tf2ss , canon , contr

Name

cont_mat — controllability matrix

```
Cc=cont_mat(A,B)
Cc=cont_mat(sl)
```

Parameters

a,b
: two real matrices of appropriate dimensions

sl
: linear system (`syslin` list)

Description

`cont_mat` returns the controllability matrix of the pair `A,B` (resp. of the system `sl=[A,B,C,D]`).

```
Cc=[B, AB, A^2 B,..., A^(n-1) B]
```

See Also

ctr_gram , contr , canon , st_ility

Name

contr — controllability, controllable subspace, staircase

```
n=contr(A,B [,tol])
[n,U]=contr(A,B [,tol])
[n,U,ind,V,Ac,Bc]=contr(A,B,[,tol])
```

Parameters

A, B
: real matrices

tol
: tolerance parameter

n
: dimension of controllable subspace.

U
: orthogonal change of basis which puts `(A,B)` in canonical form.

V
: orthogonal matrix, change of basis in the control space.

Ac
: block Hessenberg matrix `Ac=U'*A*U`

Bc
: is `U'*B*V`.

ind
: p integer vector associated with controllability indices (dimensions of subspaces `B, B +A*B,...=ind(1),ind(1)+ind(2),...`)

Description

`[n,[U]]=contr(A,B,[tol])` gives the controllable form of an `(A,B)` pair.(`dx/dt = A x + B u` or `x(n+1) = A x(n) +b u(n)`). The `n` first columns of `U` make a basis for the controllable subspace.

If `V=U(:,1:n)`, then `V'*A*V` and `V'*B` give the controllable part of the `(A,B)` pair.

The pair `(Bc, Ac)` is in staircase controllable form.

```
                     |B |sI-A      *  . . .  *      *       |
                     | 1|    11       .      .      .       |
                     |  |  A    sI-A    .    .      .       |
                     |  |   21      22    .  .      .       |
                     |  |         .     .    *      *       |
    [U'BV|sI - U'AU] = |0 |     0    .     .                |
                     |  |            A     sI-A     *       |
                     |  |             p,p-1    pp           |
                     |  |                                   |
                     |0 |         0          0   sI-A       |
                     |  |                            p+1,p+1|
```

Reference

Slicot library (see ab01od in SCIDIR/routines/slicot).

Examples

```
W=ssrand(2,3,5,list('co',3));  //cont. subspace has dim 3.
A=W("A");B=W("B");
[n,U]=contr(A,B);n
A1=U'*A*U;
spec(A1(n+1:$,n+1:$))  //uncontrollable modes
spec(A+B*rand(3,5))
```

See Also

canon , cont_mat , unobs , stabil , st_ility

Name

contrss — controllable part

```
[slc]=contrss(sl [,tol])
```

Parameters

sl
: linear system (`syslin` list)

tol
: is a threshold for controllability (see `contr`). default value is `sqrt(%eps)`.

Description

returns the controllable part of the linear system `sl = (A,B,C,D)` in state-space form.

Examples

```
A=[1,1;0,2];B=[1;0];C=[1,1];sl=syslin('c',A,B,C);  //Non minimal
slc=contrss(sl);
sl1=ss2tf(sl);sl2=ss2tf(slc);      //Compare sl1 and sl2
```

See Also

cont_mat , ctr_gram , cont_frm , contr

Name

copfac — right coprime factorization

```
[N,M,XT,YT]=copfac(G [,polf,polc,tol])
```

Parameters

G
: `syslin` list (continuous-time linear system)

polf, polc
: respectively the poles of `XT` and `YT` and the poles of `n` and `M` (default values =-1).

tol
: real threshold for detecting stable poles (default value `100*%eps`)

N,M,XT,YT
: linear systems represented by `syslin` lists

Description

`[N,M,XT,YT]=copfac(G,[polf,polc,[tol]])` returns a right coprime factorization of `G`.

`G = N*M^-1` where `N` and `M` are stable, proper and right coprime. (i.e. `[N M]` left-invertible with stability)

`XT` and `YT` satisfy:

`[XT -YT].[M N]' = eye` (Bezout identity)

`G` is assumed stabilizable and detectable.

See Also

syslin , lcf

Name

csim — simulation (time response) of linear system

```
[y [,x]]=csim(u,t,sl,[x0 [,tol]])
```

Parameters

u
: function, list or string (control)

t
: real vector specifying times with, `t(1)` is the initial time (`x0=x(t(1))`).

sl
: list (`syslin`)

y
: a matrix such that `y=[y(t(i)]`, i=1,..,n

x
: a matrix such that `x=[x(t(i)]`, i=1,..,n

tol
: a 2 vector [atol rtol] defining absolute and relative tolerances for ode solver (see ode)

Description

simulation of the controlled linear system `sl`. `sl` is assumed to be a continuous-time system represented by a `syslin` list.

`u` is the control and `x0` the initial state.

`y` is the output and `x` the state.

The control can be:

1. a function : `[inputs]=u(t)`

2. a list : `list(ut,parameter1,....,parametern)` such that: `inputs=ut(t,parameter1,....,parametern)` (`ut` is a function)

3. the string `"impuls"` for impulse response calculation (here `sl` is assumed SISO without direct feed through and `x0=0`)

4. the string `"step"` for step response calculation (here `sl` is assumed SISO without direct feed-through and `x0=0`)

5. a vector giving the values of u correponding to each t value.

Examples

```
s=poly(0,'s');rand('seed',0);w=ssrand(1,1,3);w('A')=w('A')-2*eye();
t=0:0.05:5;
//impulse(w) = step (s * w)
xbasc(0);xset("window",0);xselect();
plot2d([t',t'],[(csim('step',t,tf2ss(s)*w))',0*t'])
```

```
xbasc(1);xset("window",1);xselect();
plot2d([t',t'],[(csim('impulse',t,w))',0*t'])
//step(w) = impulse (s^-1 * w)
xbasc(3);xset("window",3);xselect();
plot2d([t',t'],[(csim('step',t,w))',0*t'])
xbasc(4);xset("window",4);xselect();
plot2d([t',t'],[(csim('impulse',t,tf2ss(1/s)*w))',0*t'])

//input defined by a time function
deff('u=input(t)','u=abs(sin(t))')
xbasc();plot2d([t',t'],[(csim(input,t,w))',0*t'])
```

See Also

syslin , dsimul , flts , ltitr , rtitr , ode , impl

Name

ctr_gram — controllability gramian

```
[Gc]=ctr_gram(A,B [,dom])
[Gc]=ctr_gram(sl)
```

Parameters

A,B
: two real matrices of appropriate dimensions

dom
: character string (`'c'` (default value) or `'d'`)

sl
: linear system, `syslin` list

Description

Controllability gramian of `(A,B)` or `sl` (a `syslin` linear system).

`dom` character string giving the time domain : `"d"` for a discrete time system and `"c"` for continuous time (default case).

Examples

```
A=diag([-1,-2,-3]);B=rand(3,2);
Wc=ctr_gram(A,B)
U=rand(3,3);A1=U*A/U;B1=U*B;
Wc1=ctr_gram(A1,B1)    //Not invariant!
```

See Also

equil1 , obs_gram , contr , cont_mat , cont_frm , contrss

Authors

S. Steer INRIA 1988

Name

dbphi — frequency response to phase and magnitude representation

```
[db,phi] =dbphi(repf)
```

Parameters

db,phi
: vector of gains (db) and phases (degrees)

repf
: vector of complex frequency response

Description

`db(k)` is the magnitude of `repf(k)` expressed in dB i.e. `db(k)=20*log(abs(repf(k)))/log(10)` and `phi(k)` is the phase of `repf(k)` expressed in degrees.

See Also

repfreq , bode

Name

dcf — double coprime factorization

```
[N,M,X,Y,NT,MT,XT,YT]=dcf(G,[polf,polc,[tol]])
```

Parameters

G
: `syslin` list (continuous-time linear system)

polf, polc
: respectively the poles of `XT` and `YT` and the poles of `N` and `M` (default values =`-1`).

tol
: real threshold for detecting stable poles (default value `100*%eps`).

N,M,XT,YT,NT,MT,X,Y
: linear systems represented by `syslin` lists

Description

returns eight stable systems `(N,M,X,Y,NT,MT,XT,YT)` for the doubly coprime factorization

`G` must be stabilizable and detectable.

See Also

copfac

Name

ddp — disturbance decoupling

```
[Closed,F,G]=ddp(Sys,zeroed,B1,D1)
[Closed,F,G]=ddp(Sys,zeroed,B1,D1,flag,alfa,beta)
```

Parameters

Sys
: `syslin` list containing the matrices `(A,B2,C,D2)`.

zeroed
: integer vector, indices of outputs of `Sys` which are zeroed.

B1
: real matrix

D1
: real matrix. `B1` and `D1` have the same number of columns.

flag
: string `'ge'` or `'st'` (default) or `'pp'`.

alpha
: real or complex vector (loc. of closed loop poles)

beta
: real or complex vector (loc. of closed loop poles)

Description

Exact disturbance decoupling (output nulling algorithm). Given a linear system, and a subset of outputs, z, which are to be zeroed, characterize the inputs w of Sys such that the transfer function from w to z is zero. `Sys` is a linear system {A,B2,C,D2} with one input and two outputs (i.e. Sys: u-->(z,y)), part the following system defined from `Sys` and `B1,D1`:

```
xdot =  A x + B1  w + B2  u
   z = C1 x + D11 w + D12 u
   y = C2 x + D21 w + D22 u
```

outputs of Sys are partitioned into (z,y) where z is to be zeroed, i.e. the matrices C and D2 are:

```
   C=[C1;C2]          D2=[D12;D22]
   C1=C(zeroed,:)     D12=D2(zeroed,:)
```

The matrix `D1` is partitioned similarly as `D1=[D11;D21]` with `D11=D1(zeroed,:)`. The control is u=Fx+Gw and one looks for matriced `F,G` such that the closed loop system: w-->z given by

```
  xdot= (A+B2*F)  x + (B1 + B2*G) w
    z = (C1+D12F) x + (D11+D12*G) w
```

has zero transfer transfer function.

`flag='ge'`no stability constraints. `flag='st'` : look for stable closed loop system (A+B2*F stable). `flag='pp'` : eigenvalues of A+B2*F are assigned to `alfa` and `beta`.

Closed is a realization of the `w-->y` closed loop system

```
  xdot= (A+B2*F)  x + (B1 + B2*G) w
    y = (C2+D22*F) x + (D21+D22*G) w
```

Stability (resp. pole placement) requires stabilizability (resp. controllability) of (A,B2).

Examples

```
rand('seed',0);nx=6;nz=3;nu=2;ny=1;
A=diag(1:6);A(2,2)=-7;A(5,5)=-9;B2=[1,2;0,3;0,4;0,5;0,0;0,0];
C1=[zeros(nz,nz),eye(nz,nz)];D12=[0,1;0,2;0,3];
Sys12=syslin('c',A,B2,C1,D12);
C=[C1;rand(ny,nx)];D2=[D12;rand(ny,size(D12,2))];
Sys=syslin('c',A,B2,C,D2);
[A,B2,C1,D12]=abcd(Sys12);  //The matrices of Sys12.
my_alpha=-1;my_beta=-2;flag='ge';
[X,dims,F,U,k,Z]=abinv(Sys12,my_alpha,my_beta,flag);
clean(X'*(A+B2*F)*X)
clean(X'*B2*U)
clean((C1+D12*F)*X)
clean(D12*U);
//Calculating an ad-hoc B1,D1
G1=rand(size(B2,2),3);
B1=-B2*G1;
D11=-D12*G1;
D1=[D11;rand(ny,size(B1,2))];

[Closed,F,G]=ddp(Sys,1:nz,B1,D1,'st',my_alpha,my_beta);
closed=syslin('c',A+B2*F,B1+B2*G,C1+D12*F,D11+D12*G);
ss2tf(closed)
```

See Also

abinv , ui_observer

Authors

F.D.

Name

des2ss — descriptor to state-space

```
[Sl]=des2ss(A,B,C,D,E [,tol])
[Sl]=des2ss(Des)
```

Parameters

A,B,C,D,E
: real matrices of appropriate dimensions

Des
: list

Sl
: `syslin` list

tol
: real parameter (threshold) (default value `100*%eps`).

Description

Descriptor to state-space transform.

`Sl=des2ss(A,B,C,D,E)` returns a linear system `Sl` equivalent to the descriptor system `(E,A,B,C,D)`.

For index one `(E,A)` pencil, explicit formula is used and for higher index pencils `rowshuff` is used.

`Sl=des2ss(Des)` with `Des=list('des',A,B,C,D,E)` returns a linear system `Sl` in state-space form with possibly a polynomial `D` matrix.

A generalized Leverrier algorithm is used.

Examples

```
s=poly(0,'s');G=[1/(s-1),s;1,2/s^3];
S1=tf2des(G);S2=tf2des(G,"withD");
W1=des2ss(S1);W2=des2ss(S2);
clean(ss2tf(W1))
clean(ss2tf(W2))
```

See Also

des2tf , glever , rowshuff

Name

des2tf — descriptor to transfer function conversion

```
[S]=des2tf(sl)
[Bfs,Bis,chis]=des2tf(sl)
```

Parameters

sl
: list (linear system in descriptor form)

Bfs, Bis
: two polynomial matrices

chis
: polynomial

S
: rational matrix

Description

Given the linear system in descriptor form i.e. `Sl=list('des',A,B,C,D,E)`, `des2tf` converts `sl` into its transfer function representation:

```
S=C*(s*E-A)^(-1)*B+D
```

Called with 3 outputs arguments `des2tf` returns `Bfs` and `Bis` two polynomial matrices, and `chis` polynomial such that:

```
S=Bfs/chis - Bis
```

`chis` is the determinant of `(s*E-A)` (up to a xcative constant);

Examples

```
s=poly(0,'s');
G=[1/(s+1),s;1+s^2,3*s^3];
Descrip=tf2des(G);Tf1=des2tf(Descrip)
Descrip2=tf2des(G,"withD");Tf2=des2tf(Descrip2)
[A,B,C,D,E]=Descrip2(2:6);Tf3=C*inv(s*E-A)*B+D
```

See Also

glever , pol2des , tf2des , ss2tf , des2ss , rowshuff

Authors

F. D.

Name

dhinf — H_infinity design of discrete-time systems

```
[AK,BK,CK,DK,(RCOND)] = dishin(A,B,C,D,ncon,nmeas,gamma)
```

Parameters

A
: the n-by-n system state matrix A.

B
: the n-by-m system input matrix B.

C
: the p-by-n system output matrix C.

D
: the p-by-m system matrix D.

ncon
: the number of control inputs. m >= ncon >= 0, p-nmeas >= ncon.

nmeas
: the number of measurements. p >= nmeas >= 0, m-ncon >= nmeas.

gamma
: the parameter gamma used in `H_infinity` design. It is assumed that gamma is sufficiently large so that the controller is admissible. gamma >= 0.

AK
: the n-by-n controller state matrix AK.

BK
: the n-by-nmeas controller input matrix BK.

CK
: the ncon-by-n controller output matrix CK.

DK
: the ncon-by-nmeas controller matrix DK.

RCOND
: a vector containing estimates of the reciprocal condition numbers of the matrices which are to be inverted and estimates of the reciprocal condition numbers of the Riccati equations which have to be solved during the computation of the controller. (See the description of the algorithm in [1].)

  RCOND
  : (1) contains the reciprocal condition number of the matrix R3,

  RCOND
  : (2) contains the reciprocal condition number of the matrix R1 - R2'*inv(R3)*R2

  RCOND
  : (3) contains the reciprocal condition number of the matrix V21,

  RCOND
  : (4) contains the reciprocal condition number of the matrix St3,

  RCOND
  : (5) contains the reciprocal condition number of the matrix V12,

RCOND
(6) contains the reciprocal condition number of the matrix Im2 + DKHAT*D22,

RCOND
(7) contains the reciprocal condition number of the X-Riccati equation,

RCOND
(8) contains the reciprocal condition number of the Z-Riccati equation.

Description

`[AK,BK,CK,DK,(RCOND)] = dhinf(A,B,C,D,ncon,nmeas, gamma)` To compute the matrices of an H-infinity (sub)optimal n-state controller

```
     | AK | BK |
K =  |----|----|,
     | CK | DK |
```

for the discrete-time system

```
     | A  | B1  B2  |   | A | B |
P =  |----|---------| = |---|---|,
     | C1 | D11 D12 |   | C | D |
     | C2 | D21 D22 |
```

and for a given value of gamma, where B2 has column size of the number of control inputs (ncon) and C2 has row size of the number of measurements (nmeas) being provided to the controller.

References

[1] P.Hr. Petkov, D.W. Gu and M.M. Konstantinov. Fortran 77 routines for Hinf and H2 design of linear discrete-time control systems. Report99-8, Department of Engineering, Leicester University, April 1999.

Examples

```
//example from Niconet report SLWN1999-12
//Hinf
A=[-0.7  0    0.3  0   -0.5 -0.1
   -0.6  0.2 -0.4 -0.3  0    0
   -0.5  0.7 -0.1  0    0   -0.8
   -0.7  0    0   -0.5 -1    0
    0    0.3  0.6 -0.9  0.1 -0.4
    0.5 -0.8  0    0    0.2 -0.9];
B=[-1 -2 -2  1  0
    1  0  1 -2  1
   -3 -4  0  2 -2
    1 -2  1  0 -1
```

```
    0  1 -2  0  3
    1  0  3 -1 -2];
C=[ 1 -1  2 -2  0 -3
   -3  0  1 -1  1  0
    0  2  0 -4  0 -2
    1 -3  0  0  3  1
    0  1 -2  1  0 -2];
D=[1 -1 -2  0  0
   0  1  0  1  0
   2 -1 -3  0  1
   0  1  0  1 -1
   0  0  1  2  1];

ncon=2
nmeas=2
gam=111.30;
[AK,BK,CK,DK] = dhinf(A,B,C,D,ncon,nmeas,gam)
```

See Also

hinf , h_inf

Name

dhnorm — discrete H-infinity norm

```
hinfnorm=dhnorm(sl,[tol],[normax])
```

Parameters

sl
: the state space system (`syslin` list) (discrete-time)

tol
: tolerance in bisection step, default value `0.01`

normax
: upper bound for the norm , default value is `1000`

hinfnorm
: the discrete infinity norm of `Sl`

Description

produces the discrete-time infinity norm of a state-space system (the maximum over all frequencies on the unit circle of the maximum singular value).

See Also

h_norm , linfn

Name

dscr — discretization of linear system

```
[sld [,r]]=dscr(sl,dt [,m])
```

Parameters

sl
: `syslin` list containing `[A,B,C,D]`.

dt
real number, sampling period

m
covariance of the input noise (continuous time)(default value=0)

r
covariance of the output noise (discrete time) given if `m` is given as input

sld
sampled (discrete-time) linear system, `syslin` list

Description

Discretization of linear system. `sl` is a continuous-time system:

`dx/dt=A*x+B*u (+ noise)`.

`sld` is the discrete-time system obtained by sampling `sl` with the sampling period `dt`.

Examples

```
s=poly(0,'s');
Sys=syslin('c',[1,1/(s+1);2*s/(s^2+2),1/s])
ss2tf(dscr(tf2ss(Sys),0.1))
```

See Also

syslin , flts , dsimul

Name

dsimul — state space discrete time simulation

```
y=dsimul(sl,u)
```

Parameters

sl
: `syslin` list describing a discrete time linear system

u
real matrix of appropriate dimension

y
output of `sl`

Description

Utility function. If `[A,B,C,D]=abcd(sl)` and `x0=sl('X0')`, `dsimul` returns `y=C*ltitr(A,B,u,x0)+D*u` i.e. the time response of `sl` to the input `u`. `sl` is assumed to be in state space form (`syslin` list).

Examples

```
z=poly(0,'z');
h=(1-2*z)/(z^2-0.2*z+1);
sl=tf2ss(h);
u=zeros(1,20);u(1)=1;
x1=dsimul(sl,u)   //Impulse response
u=ones(1,20);
x2=dsimul(sl,u);  //Step response
```

See Also

syslin , flts , ltitr

Name

dt_ility — detectability test

```
[k, [n [,U [,Sld ] ] ]]=dt_ility(Sl [,tol])
```

Parameters

Sl
: linear system (`syslin` list)

n
: dimension of unobservable subspace

k
: dimension of unstable, unobservable subspace (`k<=n`).

U
: orthogonal matrix

Sld
: linear system (`syslin` list)

tol
: threshold for controllability test.

Description

Detectability test for `sl`, a linear system in state-space representation. `U` is a basis whose `k` first columns span the unstable, unobservable subspace of `Sl` (intersection of unobservable subspace of `(A,C)` and unstable subspace of `A`). Detectability means `k=0`.

`Sld = (U'*A*U,U'*B,C*U,D)` displays the "detectable part" of `Sl=(A,B,C,D)`, i.e.

```
         [*,*,*]
U'*A*U = [0,*,*]
         [0,0,*]

C*U = [0,0,*]
```

with `(A33,C3)` observable (dimension `nx-n`), `A22` stable (dimension `n-k`) and `A11` unstable (dimension `k`).

Examples

```
A=[2,1,1;0,-2,1;0,0,3];
C=[0,0,1];
X=rand(3,3);A=inv(X)*A*X;C=C*X;
W=syslin('c',A,[],C);
[k,n,U,W1]=dt_ility(W);
W1("A")
W1("C")
```

See Also

contr , st_ility , unobs , stabil

Name

dtsi — stable anti-stable decomposition

```
[Ga,Gs,Gi]=dtsi(G,[tol])
```

Parameters

G
: linear system (`syslin` list)

Ga
: linear system (`syslin` list) antistable and strictly proper

Gs
: linear system (`syslin` list) stable and strictly proper

Gi
: real matrix (or polynomial matrix for improper systems)

tol
: optional parameter for detecting stables poles. Default value: `100*%eps`

Description

returns the stable-antistable decomposition of `G`:

`G = Ga + Gs + Gi`, `(Gi = G(oo))`

`G` can be given in state-space form or in transfer form.

See Also

syslin , pbig , psmall , pfss

Name

equil — balancing of pair of symmetric matrices

```
T=equil(P,Q)
```

Parameters

P, Q
: two positive definite symmetric matrices

T
: nonsingular matrix

Description

`equil` returns `t` such that:

`T*P*T'` and `inv(T)'*Q*inv(T)` are both equal to a same diagonal and positive matrix.

Examples

```
P=rand(4,4);P=P*P';
Q=rand(4,4);Q=Q*Q';
T=equil(P,Q)
clean(T*P*T')
clean(inv(T)'*Q*inv(T))
```

See Also

equil1 , balanc , ctr_gram

Name

equil1 — balancing (nonnegative) pair of matrices

```
[T [,siz]]=equil1(P,Q [,tol])
```

Parameters

P, Q
: two non-negative symmetric matrices

T
: nonsingular matrix

siz
: vector of three integers

tol
: threshold

Description

`equil1` computes `t` such that:

`P1=T*P*T'` and `Q1=inv(T)'*Q*inv(T)` are as follows:

`P1 = diag(S1,S2,0,0)` and `Q1 = diag(S1,0,S3,0)` with `S1,S2,S3` positive and diagonal matrices with respective dimensions `siz=[n1,n2,n3]`

`tol` is a threshold for rank determination in SVD

Examples

```
S1=rand(2,2);S1=S1*S1';
S2=rand(2,2);S2=S2*S2';
S3=rand(2,2);S3=S3*S3';
P=sysdiag(S1,S2,zeros(4,4));
Q=sysdiag(S1,zeros(2,2),S3,zeros(2,2));
X=rand(8,8);
P=X*P*X';Q=inv(X)'*Q*inv(X);
[T,siz]=equil1(P,Q);
P1=clean(T*P*T')
Q1=clean(inv(T)'*Q*inv(T))
```

See Also

balreal , minreal , equil , hankelsv

Authors

S. Steer 1987

Name

evans — Evans root locus

```
evans(H [,kmax])
```

Parameters

H
: list (linear system `syslin`)

kmax
: real (maximum gain desired for the plot)

Description

Gives the Evans root locus for a linear system in state-space or transfer form H(s) (`syslin` list). This is the locus of the roots of `1+k*H(s)=1+k*N(s)/D(s)`, in the complex plane. For a selected sample of gains `k <= kmax`, the imaginary part of the roots of `D(s)+k*N(s)` is plotted vs the real part.

To obtain the gain at a given point of the locus you can simply execute the following instruction : `k=-1/real(horner(h,[1,%i]*locate(1)))` and click the desired point on the root locus. If the coordinates of the selected point are in the real 2 x 1 vector `P=locate(1)` this `k` solves the equation `k*N(w) + D(w) =0` with `w=P(1)+%i*P(2)=[1,%i]*P`.

Examples

```
H=syslin('c',352*poly(-5,'s')/poly([0,0,2000,200,25,1],'s','c'));
evans(H,100)
P=3.0548543 - 8.8491842*%i;    //P=selected point
k=-1/real(horner(H,P));
Ns=H('num');Ds=H('den');
roots(Ds+k*Ns)     //contains P as particular root
// Another one
clf();s=poly(0,'s');n=1+s;
d=real(poly([-1 -2 -%i %i],'s'));
evans(n,d,100);
//
clf();n=real(poly([0.1-%i 0.1+%i,-10],'s'));
evans(n,d,80);
```

See Also

kpure , krac2 , locate

Name

feedback — feedback operation

```
Sl=Sl1/.Sl2
```

Parameters

Sl1,Sl2
: linear systems (`syslin` list) in state-space or transfer form, or ordinary gain matrices.

Sl
: linear system (`syslin` list) in state-space or transfer form

Description

The feedback operation is denoted by `/.` (slashdot). This command returns `Sl=Sl1*(I+Sl2*Sl1)^-1`, i.e the (negative) feedback of `Sl1` and `Sl2`. `Sl` is the transfer `v -> y` for `y = Sl1 u, u = v - Sl2 y`.

The result is the same as `Sl=LFT([0,I;I,-Sl2],Sl1)`.

Caution: do not use with decimal point (e.g. `1/.1` is ambiguous!)

Examples

```
S1=ssrand(2,2,3);S2=ssrand(2,2,2);
W=S1/.S2;
ss2tf(S1/.S2)
//Same operation by LFT:
ss2tf(lft([zeros(2,2),eye(2,2);eye(2,2),-S2],S1))
//Other approach: with constant feedback
BigS=sysdiag(S1,S2); F=[zeros(2,2),eye(2,2);-eye(2,2),zeros(2,2)];
Bigclosed=BigS/.F;
W1=Bigclosed(1:2,1:2);   //W1=W (in state-space).
ss2tf(W1)
//Inverting
ss2tf(S1*inv(eye()+S2*S1))
```

See Also

lft , sysdiag , augment , obscont

Name

findABCD — discrete-time system subspace identification

```
[SYS,K] = findABCD(S,N,L,R,METH,NSMPL,TOL,PRINTW)
SYS = findABCD(S,N,L,R,METH)

[SYS,K,Q,Ry,S,RCND] = findABCD(S,N,L,R,METH,NSMPL,TOL,PRINTW)
[SYS,RCND] = findABCD(S,N,L,R,METH)
```

Parameters

S
: integer, the number of block rows in the block-Hankel matrices

N
: integer, the system order

L
: integer, the number of output

R
: matrix, relevant part of the R factor of the concatenated block-Hankel matrices computed by a call to findr.

METH
: integer, an option for the method to use

  = 1
  : MOESP method with past inputs and outputs;

  = 2
  : N4SID method;

  = 3
  : combined method: A and C via MOESP, B and D via N4SID.

  Default: METH = 3.

NSMPL
: integer, the total number of samples used for calculating the covariance matrices and the Kalman predictor gain. This parameter is not needed if the covariance matrices and/or the Kalman predictor gain matrix are not desired. If NSMPL = 0, then K, Q, Ry, and S are not computed. Default: NSMPL = 0.

TOL
: the tolerance used for estimating the rank of matrices. If TOL > 0, then the given value of TOL is used as a lower bound for the reciprocal condition number. Default: prod(size(matrix))*epsilon_machine where epsilon_machine is the relative machine precision.

PRINTW
: integer, switch for printing the warning messages.

  PRINTW
  : = 1: print warning messages;

  PRINTW
  : = 0: do not print warning messages.

  Default: PRINTW = 0.

SYS
: computes a state-space realization SYS = (A,B,C,D) (an syslin object)

K
: the Kalman predictor gain K (if NSMPL > 0)

Q
: state covariance

Ry
: output covariance

S
: state-output cross-covariance

RCND
: vector, reciprocal condition numbers of the matrices involved in rank decisions, least squares or Riccati equation solutions

Description

Finds the system matrices and the Kalman gain of a discrete-time system, given the system order and the relevant part of the R factor of the concatenated block-Hankel matrices, using subspace identification techniques (MOESP and/or N4SID).

- [SYS,K] = findABCD(S,N,L,R,METH,NSMPL,TOL,PRINTW) computes a state- space realization SYS = (A,B,C,D) (an ss object), and the Kalman predictor gain K (if NSMPL > 0). The model structure is:

```
x(k+1) = Ax(k) + Bu(k) + Ke(k),   k >= 1,
y(k)   = Cx(k) + Du(k) + e(k),
```

  where x(k) and y(k) are vectors of length N and L, respectively.

- [SYS,K,Q,Ry,S,RCND] = findABCD(S,N,L,R,METH,NSMPL,TOL,PRINTW) also returns the state, output, and state-output (cross-)covariance matrices Q, Ry, and S (used for computing the Kalman gain), as well as the vector RCND of length lr containing the reciprocal condition numbers of the matrices involved in rank decisions, least squares or Riccati equation solutions, where

```
lr = 4,  if Kalman gain matrix K is not required, and
lr = 12, if Kalman gain matrix K is required.
```

Matrix R, computed by findR, should be determined with suitable arguments METH and JOBD. METH = 1 and JOBD = 1 must be used in findR, for METH = 1 in findABCD; METH = 1 must be used in findR, for METH = 3 in findABCD.

Examples

```
//generate data from a given linear system
```

```
A = [ 0.5, 0.1,-0.1, 0.2;
      0.1, 0,  -0.1,-0.1;
     -0.4,-0.6,-0.7,-0.1;
      0.8, 0,  -0.6,-0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));


// Compute R
S=15;
[R,N1,SVAL] = findR(S,Y',U');
N=3;
SYS1 = findABCD(S,N,1,R) ;SYS1.dt=0.1;

SYS1.X0 = inistate(SYS1,Y',U');

Y1=flts(U,SYS1);
xbasc();plot2d((1:nsmp)',[Y',Y1'])
```

See Also

findAC , findBD , findBDK , findR , sorder , sident

Name

findAC — discrete-time system subspace identification

```
[A,C] = findAC(S,N,L,R,METH,TOL,PRINTW)
[A,C,RCND] = findAC(S,N,L,R,METH,TOL,PRINTW)
```

Parameters

S
: integer, the number of block rows in the block-Hankel matrices

N
: integer

L
: integer

R
: matrix, relevant part of the R factor of the concatenated block-Hankel matrices computed by a call to findr.

METH
: integer, an option for the method to use

  = 1
  : MOESP method with past inputs and outputs;

  = 2
  : N4SID method;

  Default: METH = 3.

TOL
: the tolerance used for estimating the rank of matrices. If TOL > 0, then the given value of TOL is used as a lower bound for the reciprocal condition number. Default: prod(size(matrix))*epsilon_machine where epsilon_machine is the relative machine precision.

PRINTW
: integer, switch for printing the warning messages.

  PRINTW
  : = 1: print warning messages;

  = 0
  : do not print warning messages.

  Default: PRINTW = 0.

A
: matrix, state system matrix

C
: matrix, output system matrix

RCND
: vector of length 4, condition numbers of the matrices involved in rank decision

Description

finds the system matrices A and C of a discrete-time system, given the system order and the relevant part of the R factor of the concatenated block-Hankel matrices, using subspace identification techniques (MOESP or N4SID).

- [A,C] = findAC(S,N,L,R,METH,TOL,PRINTW) computes the system matrices A and C. The model structure is: x(k+1) = Ax(k) + Bu(k) + Ke(k), k >= 1, y(k) = Cx(k) + Du(k) + e(k), where x(k) and y(k) are vectors of length N and L, respectively.
- [A,C,RCND] = findAC(S,N,L,R,METH,TOL,PRINTW) also returns the vector RCND of length 4 containing the condition numbers of the matrices involved in rank decisions.

Matrix R, computed by findR, should be determined with suitable arguments METH and JOBD.

Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1,-0.1, 0.2;
      0.1, 0,  -0.1,-0.1;
     -0.4,-0.6,-0.7,-0.1;
      0.8, 0,  -0.6,-0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));

// Compute R
S=15;L=1;
[R,N,SVAL] = findR(S,Y',U');

N=3;
METH=3;TOL=-1;
[A,C] = findAC(S,N,L,R,METH,TOL);
```

See Also

findABCD , findBD , findBDK , findR , sorder , sident

Name

findBD — initial state and system matrices B and D of a discrete-time system

```
[(x0) (,B (,D)) (,V) (,rcnd)] = findBD(jobx0,comuse (,job),A (,B),C (,D),Y
(,U,tol,printw,ldwork))
```

Parameters

jobx0
: integer option to specify whether or not the initial state should be computed:

  =
  : 1 : compute the initial state x0;

  =
  : 2 : do not compute the initial state (possibly, because x0 is known to be zero).

comuse
: integer option to specify whether the system matrices B and D should be computed or used:

  =
  : 1 : compute the matrices B and D, as specified by job;

  =
  : 2 : use the matrices B and D, as specified by job;

  =
  : 3 : do not compute/use the matrices B and D.

job
: integer option to determine which of the system matrices B and D should be computed or used:

  =
  : 1 : compute/use the matrix B only (D is known to be zero);

  =
  : 2 : compute/use the matrices B and D.

  job must not be specified if jobx0 = 2 and comuse = 2, or if comuse = 3.

A
: state matrix of the given system

B
: optionnal, input matrix of the given system

C
: output matrix of the given system

D
: optionnal, direct feedthrough of the given system

Y
: the t-by-l output-data sequence matrix. Column j of Y contains the t values of the j-th output component for consecutive time increments.

U
: the t-by-m input-data sequence matrix (input when jobx0 = 1 and comuse = 2, or comuse = 1). Column j of U contains the t values of the j-th input component for consecutive time increments.

tol
: optionnal, tolerance used for estimating the rank of matrices. If tol > 0, then the given value of tol is used as a lower bound for the reciprocal condition number; an m-by-n matrix whose estimated condition number is less than 1/tol is considered to be of full rank. Default: m*n*epsilon_machine where epsilon_machine is the relative machine precision.

printw
: :optionnal, switch for printing the warning messages.

  =
  : 1: print warning messages;

  =
  : 0: do not print warning messages.

  Default: printw = 0.

ldwork
: (optional) the workspace size. Default : computed by the formula LDWORK = MAX(minimum workspace size needed, 2*CSIZE/3, CSIZE - (m + l)*t - 2*n*(n + m + l) - l*m) where CSIZE is the cache size in double precision words.

x0
: initial state vector

Br
: system input matrix

Dr
: system direct feedthrough matrix

V
: the n-by-n orthogonal matrix which reduces A to a real Schur form (output when jobx0 = 1 or comuse = 1).

rcnd
: (optional) the reciprocal condition numbers of the matrices involved in rank decisions.

Description

findBD function for estimating the initial state and the system matrices B and D of a discrete-time system, using SLICOT routine IB01CD.

```
  [x0,Br,V,rcnd] = findBD(1,1,1,A,C,Y,U)
[x0,Br,Dr,V,rcnd] = findBD(1,1,2,A,C,Y,U)
     [Br,V,rcnd] = findBD(2,1,1,A,C,Y,U)
   [B,Dr,V,rcnd] = findBD(2,1,2,A,C,Y,U)
    [x0,V,rcnd] = findBD(1,2,1,A,B,C,Y,U)
    [x0,V,rcnd] = findBD(1,2,2,A,B,C,D,Y,U)
      [x0,rcnd] = findBD(2,2)      // (Set x0 = 0, rcnd = 1)
    [x0,V,rcnd] = findBD(1,3,A,C,Y)
```

Note: the example lines above may contain at the end the parameters tol, printw, ldwork.

FINDBD estimates the initial state and/or the system matrices Br and Dr of a discrete-time system, given the system matrices A, C, and possibly B, D, and the input and output trajectories of the system.

The model structure is :

```
        x(k+1) = Ax(k) + Bu(k),    k >= 1,
        y(k)   = Cx(k) + Du(k),
```

where x(k) is the n-dimensional state vector (at time k),

u(k) is the m-dimensional input vector,

y(k) is the l-dimensional output vector,

and A, B, C, and D are real matrices of appropriate dimensions.

Comments

1. The n-by-m system input matrix B is an input parameter when jobx0 = 1 and comuse = 2, and it is an output parameter when comuse = 1.

2. The l-by-m system matrix D is an input parameter when jobx0 = 1, comuse = 2 and job = 2, and it is an output parameter when comuse = 1 and job = 2.

3. The n-vector of estimated initial state x(0) is an output parameter when jobx0 = 1, but also when jobx0 = 2 and comuse <= 2, in which case it is set to 0.

4. If ldwork is specified, but it is less than the minimum workspace size needed, that minimum value is used instead.

Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1,-0.1, 0.2;
      0.1, 0,  -0.1,-0.1;
     -0.4,-0.6,-0.7,-0.1;
      0.8, 0,  -0.6,-0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));

// Compute R
S=15;L=1;
[R,N,SVAL] = findR(S,Y',U');

N=3;
METH=3;TOL=-1;
[A,C] = findAC(S,N,L,R,METH,TOL);
[X0,B,D] = findBD(1,1,2,A,C,Y',U')
```

```
SYS1=syslin(1,A,B,C,D,X0);

Y1=flts(U,SYS1);
xbasc();plot2d((1:nsmp)',[Y',Y1'])
```

See Also

inistate , findx0BD , findABCD , findAC , findBD

Authors

V. Sima, Katholieke Univ. Leuven, Belgium, May 2000. (Revisions: V. Sima, July 2000)

Name

findBDK — Kalman gain and B D system matrices of a discrete-time system

```
[B,D,K] = findBDK(S,N,L,R,A,C,METH,JOB,NSMPL,TOL,PRINTW)
[B,D,RCND] = findBDK(S,N,L,R,A,C,METH,JOB)
[B,D,K,Q,Ry,S,RCND] = findBDK(S,N,L,R,A,C,METH,JOB,NSMPL,TOL,PRINTW)
```

Parameters

S
: integer, the number of block rows in the block-Hankel matrices

N
: integer

L
: integer

R
: matrix, relevant part of the R factor of the concatenated block-Hankel matrices computed by a call to findR.

A
: square matrix

C
: matrix

METH
: integer, an option for the method to use

  = 1
  : MOESP method with past inputs and outputs;

  = 2
  : N4SID method;

  Default: METH = 2.

JOB
: an option specifying which system matrices should be computed:

  = 1
  : compute the matrix B;

  = 2
  : compute the matrices B and D.

  Default: JOB = 2.

NSMPL
: integer, the total number of samples used for calculating the covariance matrices and the Kalman predictor gain. This parameter is not needed if the covariance matrices and/or the Kalman predictor gain matrix are not desired. If NSMPL = 0, then K, Q, Ry, and S are not computed. Default: NSMPL = 0.

TOL
: the tolerance used for estimating the rank of matrices. If TOL > 0, then the given value of TOL is used as a lower bound for the reciprocal condition number. Default: prod(size(matrix))*epsilon_machine where epsilon_machine is the relative machine precision.

PRINTW
integer, switch for printing the warning messages.

PRINTW
= 1: print warning messages;

PRINTW
= 0: do not print warning messages.

Default: PRINTW = 0.

SYS
computes a state-space realization SYS = (A,B,C,D) (an syslin object)

K
the Kalman predictor gain K (if NSMPL > 0)

Q
state covariance

Ry
output covariance

S
state-output cross-covariance

RCND
he vector of length 12 containing the reciprocal condition numbers of the matrices involved in rank decisions, least squares or Riccati equation solutions.

Description

finds the system matrices B and D and the Kalman gain of a discrete-time system, given the system order, the matrices A and C, and the relevant part of the R factor of the concatenated block-Hankel matrices, using subspace identification techniques (MOESP or N4SID).

- [B,D,K] = findBDK(S,N,L,R,A,C,METH,JOB,NSMPL,TOL,PRINTW) computes the system matrices B (if JOB = 1), B and D (if JOB = 2), and the Kalman predictor gain K (if NSMPL > 0). The model structure is:

```
x(k+1) = Ax(k) + Bu(k) + Ke(k),    k >= 1,
y(k)   = Cx(k) + Du(k) + e(k),
```

  where x(k) and y(k) are vectors of length N and L, respectively.

- [B,D,RCND] = findBDK(S,N,L,R,A,C,METH,JOB) also returns the vector RCND of length 4 containing the reciprocal condition numbers of the matrices involved in rank decisions.

- [B,D,K,Q,Ry,S,RCND] = findBDK(S,N,L,R,A,C,METH,JOB,NSMPL,TOL,PRINTW) also returns the state, output, and state-output (cross-)covariance matrices Q, Ry, and S (used for computing the Kalman gain), as well as the vector RCND of length 12 containing the reciprocal condition numbers of the matrices involved in rank decisions, least squares or Riccati equation solutions.

Matrix R, computed by findR, should be determined with suitable arguments METH and JOBD. METH = 1 and JOBD = 1 must be used in findR, for METH = 1 in findBDK. Using METH = 1 in FINDR and METH = 2 in findBDK is allowed.

The number of output arguments may vary, but should correspond to the input arguments, e.g.,

```
           B = findBDK(S,N,L,R,A,C,METH,1)  or
       [B,D] = findBDK(S,N,L,R,A,C,METH,2)  or
  [B,D,RCND] = findBDK(S,N,L,R,A,C,METH,2)
```

Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1,-0.1, 0.2;
      0.1, 0,  -0.1,-0.1;
     -0.4,-0.6,-0.7,-0.1;
      0.8, 0,  -0.6,-0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));

// Compute R
S=15;L=1;
[R,N,SVAL] = findR(S,Y',U');

N=3;
METH=3;TOL=-1;
[A,C] = findAC(S,N,L,R,METH,TOL);
[B,D,K] = findBDK(S,N,L,R,A,C);
SYS1=syslin(1,A,B,C,D);

SYS1.X0 = inistate(SYS1,Y',U');

Y1=flts(U,SYS1);
xbasc();plot2d((1:nsmp)',[Y',Y1'])
```

See Also

findABCD , findAC , findBD , findR , sorder , sident

Name

findR — Preprocessor for estimating the matrices of a linear time-invariant dynamical system

```
[R,N [,SVAL,RCND]] = findR(S,Y,U,METH,ALG,JOBD,TOL,PRINTW)
[R,N] = findR(S,Y)
```

Parameters

S
: the number of block rows in the block-Hankel matrices.

Y
: :

U
: :

METH
: an option for the method to use:

  1
  : MOESP method with past inputs and outputs;

  2
  : N4SI15 0 1 1 1000D method.

  Default: METH = 1.

ALG
: an option for the algorithm to compute the triangular factor of the concatenated block-Hankel matrices built from the input-output data:

  1
  : Cholesky algorithm on the correlation matrix;

  2
  : fast QR algorithm;

  3
  : standard QR algorithm.

  Default: ALG = 1.

JOBD
: an option to specify if the matrices B and D should later be computed using the MOESP approach:

  =
  : 1 : the matrices B and D should later be computed using the MOESP approach;

  =
  : 2 : the matrices B and D should not be computed using the MOESP approach.

  Default: JOBD = 2. This parameter is not relevant for METH = 2.

TOL
: a vector of length 2 containing tolerances:

  TOL
  : (1) is the tolerance for estimating the rank of matrices. If TOL(1) > 0, the given value of TOL(1) is used as a lower bound for the reciprocal condition number.

Default: TOL(1) = prod(size(matrix))*epsilon_machine where epsilon_machine is the relative machine precision.

TOL
(2) is the tolerance for estimating the system order. If TOL(2) >= 0, the estimate is indicated by the index of the last singular value greater than or equal to TOL(2). (Singular values less than TOL(2) are considered as zero.)

When TOL(2) = 0, then S*epsilon_machine*sval(1) is used instead TOL(2), where sval(1) is the maximal singular value. When TOL(2) < 0, the estimate is indicated by the index of the singular value that has the largest logarithmic gap to its successor. Default: TOL(2) = -1.

PRINTW
a switch for printing the warning messages.

=
1: print warning messages;

=
0: do not print warning messages.

Default: PRINTW = 0.

R
:

N
the order of the discrete-time realization

SVAL
singular values SVAL, used for estimating the order.

RCND
vector of length 2 containing the reciprocal condition numbers of the matrices involved in rank decisions or least squares solutions.

Description

findR Preprocesses the input-output data for estimating the matrices of a linear time-invariant dynamical system, using Cholesky or (fast) QR factorization and subspace identification techniques (MOESP or N4SID), and estimates the system order.

[R,N] = findR(S,Y,U,METH,ALG,JOBD,TOL,PRINTW) returns the processed upper triangular factor R of the concatenated block-Hankel matrices built from the input-output data, and the order N of a discrete-time realization. The model structure is:

```
x(k+1) = Ax(k) + Bu(k) + w(k),    k >= 1,
y(k)   = Cx(k) + Du(k) + e(k).
```

The vectors y(k) and u(k) are transposes of the k-th rows of Y and U, respectively.

[R,N,SVAL,RCND] = findR(S,Y,U,METH,ALG,JOBD,TOL,PRINTW) also returns the singular values SVAL, used for estimating the order, as well as, if meth = 2, the vector RCND of length 2 containing the reciprocal condition numbers of the matrices involved in rank decisions or least squares solutions.

[R,N] = findR(S,Y) assumes U = [] and default values for the remaining input arguments.

Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1,-0.1, 0.2;
      0.1, 0,  -0.1,-0.1;
     -0.4,-0.6,-0.7,-0.1;
      0.8, 0,  -0.6,-0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
U=(ones(1,1000)+rand(1,1000,'normal'));
Y=(flts(U,SYS)+0.5*rand(1,1000,'normal'));
// Compute R

[R,N,SVAL] = findR(15,Y',U');
SVAL
N
```

See Also

findABCD , findAC , findBD , findBDK , sorder , sident

Name

findx0BD — Estimates state and B and D matrices of a discrete-time linear system

```
[X0,B,D] = findx0BD(A,C,Y,U,WITHX0,WITHD,TOL,PRINTW)
[x0,B,D,V,rcnd] = findx0BD(A,C,Y,U)
```

Parameters

A
: state matrix of the system

C
: C matrix of the system

Y
: system output

U
: system input

WITHX0
: a switch for estimating the initial state x0.

  =
  : 1: estimate x0;

  =
  : 0: do not estimate x0.

  Default: WITHX0 = 1.

WITHD
: a switch for estimating the matrix D.

  =
  : 1: estimate the matrix D;

  =
  : 0: do not estimate the matrix D.

  Default: WITHD = 1.

TOL
: the tolerance used for estimating the rank of matrices. If TOL > 0, then the given value of TOL is used as a lower bound for the reciprocal condition number. Default: prod(size(matrix))*epsilon_machine where epsilon_machine is the relative machine precision.

PRINTW
: a switch for printing the warning messages.

  =
  : 1: print warning messages;

  =
  : 0: do not print warning messages.

  Default: PRINTW = 0.

X0
: intial state of the estimated linear system.

B
B matrix of the estimated linear system.

D
D matrix of the estimated linear system.

V
orthogonal matrix which reduces the system state matrix A to a real Schur form

rcnd
estimates of the reciprocal condition numbers of the matrices involved in rank decisions.

Description

findx0BD Estimates the initial state and/or the matrices B and D of a discrete-time linear system, given the (estimated) system matrices A, C, and a set of input/output data.

[X0,B,D] = findx0BD(A,C,Y,U,WITHX0,WITHD,TOL,PRINTW) estimates the initial state X0 and the matrices B and D of a discrete-time system using the system matrices A, C, output data Y and the input data U. The model structure is :

```
    x(k+1) = Ax(k) + Bu(k),   k >= 1,
    y(k)   = Cx(k) + Du(k),
```

The vectors y(k) and u(k) are transposes of the k-th rows of Y and U, respectively.

[x0,B,D,V,rcnd] = findx0BD(A,C,Y,U) also returns the orthogonal matrix V which reduces the system state matrix A to a real Schur form, as well as some estimates of the reciprocal condition numbers of the matrices involved in rank decisions.

```
   B = findx0BD(A,C,Y,U,0,0)  returns B only, and
[B,D] = findx0BD(A,C,Y,U,0)    returns B and D only.
```

Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1,-0.1, 0.2;
      0.1, 0,  -0.1,-0.1;
     -0.4,-0.6,-0.7,-0.1;
      0.8, 0,  -0.6,-0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));

// Compute R
S=15;L=1;
```

```
[R,N,SVAL] = findR(S,Y',U');

N=3;
METH=3;TOL=-1;
[A,C] = findAC(S,N,L,R,METH,TOL);

[X0,B,D,V,rcnd] = findx0BD(A,C,Y',U');
SYS1=syslin(1,A,B,C,D,X0);


Y1=flts(U,SYS1);
xbasc();plot2d((1:nsmp)',[Y',Y1'])
```

See Also

findBD , inistate

Name

flts — time response (discrete time, sampled system)

```
[y [,x]]=flts(u,sl [,x0])
[y]=flts(u,sl [,past])
```

Parameters

u
: matrix (input vector)

sl
: list (linear system `syslin`)

x0
: vector (initial state ; default value=`0`)

past
: matrix (of the past ; default value=`0`)

x,y
: matrices (state and output)

Description

- State-space form:

`sl` is a discrete linear system given by its state space representation (see syslin):

`sl=syslin('d',A,B,C,D)`:

```
x[t+1] = A x[t] + B u[t]
y[t] = C x[t] + D u[t]
```

or, more generally, if `D` is a polynomial matrix (`p = degree(D(z))`):

```
D(z)=D_0 + z D_1 + z^2 D_2 +..+ z^p D_p
y[t] = C x[t] + D_0 u[t] + D_1 u[t+1] +..+ D_[p] u[t+p]
```

- Transfer form:

`y=flts(u,sl[,past])`. Here `sl` is a linear system in transfer matrix representation i.e

`sl=syslin('d',transfer_matrix)` (see `syslin`).

```
past = [u     ,...,  u   ]
       [ -nd           -1]
       [y     ,...,  y   ]
       [ -nd           -1]
```

is the matrix of past values of u and y.

`nd` is the maximum of degrees of lcm's of each row of the denominator matrix of `sl`.

```
u=[u0 u1 ... un]  (input)
y=[y0 y1 ... yn]  (output)
```

p is the difference between maximum degree of numerator and maximum degree of denominator

Examples

```
sl=syslin('d',1,1,1);u=1:10;
y=flts(u,sl);
plot2d(y)
[y1,x1]=flts(u(1:5),sl);y2=flts(u(6:10),sl,x1);
y-[y1,y2]

//With polynomial D:
z=poly(0,'z');
D=1+z+z^2; p =degree(D);
sl=syslin('d',1,1,1,D);
y=flts(u,sl);[y1,x1]=flts(u(1:5),sl);
y2=flts(u(5-p+1:10),sl,x1);  // (update)
y-[y1,y2]

//Delay (transfer form): flts(u,1/z)
// Usual responses
z=poly(0,'z');
h=syslin(0.1,(1-2*z)/(z^2+0.3*z+1))
imprep=flts(eye(1,20),tf2ss(h));   //Impulse response
clf();
plot(imprep,'b')
u=ones(1,20);
stprep=flts(ones(1,20),tf2ss(h));   //Step response
plot(stprep,'g')

// Other examples
A=[1 2 3;0 2 4;0 0 1];B=[1 0;0 0;0 1];C=eye(3,3);Sys=syslin('d',A,B,C);
H=ss2tf(Sys); u=[1;-1]*(1:10);
//
yh=flts(u,H); ys=flts(u,Sys);
norm(yh-ys,1)
//hot restart
[ys1,x]=flts(u(:,1:4),Sys);ys2=flts(u(:,5:10),Sys,x);
norm([ys1,ys2]-ys,1)
//
yh1=flts(u(:,1:4),H);yh2=flts(u(:,5:10),H,[u(:,2:4);yh(:,2:4)]);
norm([yh1,yh2]-yh,1)
//with D<>0
D=[-3 8;4 -0.5;2.2 0.9];
Sys=syslin('d',A,B,C,D);
H=ss2tf(Sys); u=[1;-1]*(1:10);
rh=flts(u,H); rs=flts(u,Sys);
norm(rh-rs,1)
//hot restart
[ys1,x]=flts(u(:,1:4),Sys);ys2=flts(u(:,5:10),Sys,x);
norm([ys1,ys2]-rs,1)
//With H:
yh1=flts(u(:,1:4),H);yh2=flts(u(:,5:10),H,[u(:,2:4); yh1(:,2:4)]);
```

```
norm([yh1,yh2]-rh)
```

See Also

ltitr, dsimul, rtitr

Name

fourplan — augmented plant to four plants

```
[P11,P12,P21,P22]=fourplan(P,r)
```

Parameters

P
: `syslin` list (linear system)

r
: 1x2 row vector, dimension of `P22`

P11,P12,P21,P22
: `syslin` lists.

Description

Utility function.

`P` being partitioned as follows:

```
P=[ P11 P12;
    P21 P22]
```

with `size(P22)=r` this function returns the four linear systems `P11,P12,P21,P22`.

See Also

lqg , lqg2stan , lqr , lqe , lft

Name

frep2tf — transfer function realization from frequency response

```
[h [,err]]=frep2tf(frq,repf,dg [,dom,tols,weight])
```

Parameters

frq
: vector of frequencies in Hz.

repf
: vector of frequency response

dg
: degree of linear system

dom
: time domain (`'c'` or `'d'` or `dt`)

tols
: a vector of size 3 giving the relative and absolute tolerance and the maximum number of iterations (default values are `rtol=1.e-2; atol=1.e-4, N=10`).

weight
: vector of weights on frequencies

h
: SISO transfer function

err
: error (for example if `dom='c' sum(abs(h(2i*pi*frq) - rep)^2)/size(frq,*)`)

Description

Frequency response to transfer function conversion. The order of `h` is a priori given in `dg` which must be provided. The following linear system is solved in the least square sense.

```
weight(k)*(n( phi_k) - d(phi_k)*rep_k)=0, k=1,..,n
```

where `phi_k= 2*%i*%pi*frq` when `dom='c'` and `phi_k=exp(2*%i*%pi*dom*frq` if not. If the `weight` vector is not given a default penalization is used (when `dom='c'`).

A stable and minimum phase system can be obtained by using function `factors`.

Examples

```
s=poly(0,'s');
h=syslin('c',(s-1)/(s^3+5*s+20))
frq=0:0.05:3;repf=repfreq(h,frq);
clean(frep2tf(frq,repf,3))
```

```
Sys=ssrand(1,1,10);
frq=logspace(-3,2,200);
[frq,rep]=repfreq(Sys,frq);  //Frequency response of Sys
[Sys2,err]=frep2tf(frq,rep,10);Sys2=clean(Sys2)//Sys2 obtained from freq. resp
[frq,rep2]=repfreq(Sys2,frq); //Frequency response of Sys2
xbasc();bode(frq,[rep;rep2])   //Responses of Sys and Sys2
[sort(spec(Sys('A'))),sort(roots(Sys2('den')))] //poles

dom=1/1000; // Sampling time
z=poly(0,'z');
h=syslin(dom,(z^2+0.5)/(z^3+0.1*z^2-0.5*z+0.08))
frq=(0:0.01:0.5)/dom;repf=repfreq(h,frq);
[Sys2,err]=frep2tf(frq,repf,3,dom);
[frq,rep2]=repfreq(Sys2,frq); //Frequency response of Sys2
xbasc();plot2d1("onn",frq',abs([repf;rep2])');
```

See Also

imrep2ss , arl2 , time_id , armax , frfit

Name

freq — frequency response

```
[x]=freq(A,B,C [,D],f)
[x]=freq(NUM,DEN,f)
```

Parameters

A, B, C, D
: real matrices of respective dimensions `nxn, nxp, mxn, mxp`.

NUM,DEN
: polynomial matrices of dimension `mxp`

x
: real or complex matrix

Description

`x=freq(A,B,C [,D],f)` returns a real or complex `mxp*t` matrix such that:

`x(:,k*p:(k+1)*p)= C*inv(f(k)*eye()-A)*B + D`.

Thus, for `f` taking values along the imaginary axis or on the unit circle `x` is the continuous or discrete time frequency response of `(A,B,C,D)`.

`x=freq(NUM,DEN,f)` returns a real or complex matrix `x` such that columns `k*(p-1)+1` to `k*p` of `x` contain the matrix `NUM(f(k))./DEN(f(k))`

Examples

```
s=poly(0,'s');
sys=(s+1)/(s^3-5*s+4)
rep=freq(sys("num"),sys("den"),[0,0.9,1.1,2,3,10,20])
[horner(sys,0),horner(sys,20)]
//
Sys=tf2ss(sys);
[A,B,C,D]=abcd(Sys);
freq(A,B,C,[0,0.9,1.1,2,3,10,20])
```

See Also

repfreq , horner

Name

freson — peak frequencies

```
fr=freson(h)
```

Parameters

h
: `syslin` list

fr
: vector of peak frequencies in Hz

Description

returns the vector of peak frequencies in Hz for the SISO plant h

Examples

```
h=syslin('c',-1+%s,(3+2*%s+%s^2)*(50+0.1*%s+%s^2))
fr=freson(h)
bode(h)
g=20*log(abs(repfreq(h,fr)))/log(10)
```

See Also

frep2tf , zgrid , h_norm

Name

fspecg — stable factorization

```
[gm]=fspecg(g).
```

Parameters

g,gm
: `syslin` lists (linear systems in state-space representation)

Description

returns `gm` with `gm` and `gm^-1` stable such that:

```
gtild(g)*g = gtild(gm)*gm
```

`g` and `gm` are continuous-time linear systems in state-space form.

Imaginary-axis poles are forbidden.

Name

fstabst — Youla's parametrization

```
[J]=fstabst(P,r)
```

Parameters

P
: `syslin` list (linear system)

r
: 1x2 row vector, dimension of `P22`

J
: `syslin` list (linear system in state-space representation)

Description

Parameterization of all stabilizing feedbacks.

`P` is partitioned as follows:

```
P=[ P11 P12;
    P21 P22]
```

(in state-space or transfer form: automatic conversion in state-space is done for the computations)

`r` = size of `P22` subsystem, (2,2) block of `P`

```
         J =[ J11 J12;
              J21 J22]
```

`K` is a stabilizing controller for `P` (i.e. `P22`) iff `K=lft(J,r,Q)` with `Q` stable.

The central part of `J` , `J11` is the lqg regulator for `P`

This `J` is such that defining `T` as the 2-port `lft` of `P` and `J` : `[T,rt]=lft(P,r,J,r)` one has that `T12` is inner and `T21` is co-inner.

Examples

```
ny=2;nu=3;nx=4;
P22=ssrand(ny,nu,nx);
bigQ=rand(nx+nu,nx+nu);bigQ=bigQ*bigQ';
bigR=rand(nx+ny,nx+ny);bigR=bigR*bigR';
[P,r]=lqg2stan(P22,bigQ,bigR);
J=fstabst(P,r);
```

```
Q=ssrand(nu,ny,1);Q('A')=-1;  //Stable Q
K=lft(J,r,Q);
A=h_cl(P,r,K); spec(A)
```

See Also

obscont , lft , lqg , lqg2stan

Name

g_margin — gain margin and associated crossover frequency

```
gm=g_margin(h)
[gm,fr]=g_margin(h)
```

Parameters

h
: a SISO linear system (see :syslin).

gm
: a number, the gain margin (in dB) if any of `Inf`

fr
: a number, the associated frequency in hertz, or an empty matrix if the gain margin does not exist.

Description

Given a SISO linear system in continuous or discrete time, `g_margin` returns `gm`, the gain margin in dB of `h` and `fr`, the achieved corresponding frequency in hz.

The gain margin, if it exists, is the minimal value of the system gain at points where the nyquist plot crosses the negative real axis. In other words the gain margin is `20*log10(1/g)` where `g` is the open loop gain of `h` when the frequency response phase of `h` equals -180°

The algorithm uses polynomial root finder to solve the equations:

h(s)=h(-s)
: for the continuous time case.

h(z)=h(1/z)
: for the discrete time case.

Examples

```
h=syslin('c',-1+%s,3+2*%s+%s^2) //continuous time case
[g,fr]=g_margin(h)
[g,fr]=g_margin(h-10)
nyquist(h-10)


h = syslin(0.1,0.04798*%z+0.0464,%z^2-1.81*%z+0.9048);//discrete time case
[g ,fr]=g_margin(h);
show_margins(h)
```

See Also

p_margin, show_margins, repfreq, black, bode, chart, nyquist

Authors

Serge Steer, INRIA

Name

gainplot — magnitude plot

```
gainplot(sl,fmin,fmax [,step] [,comments] )
gainplot(frq,db,phi [,comments])
gainplot(frq, repf [,comments])
```

Parameters

sl
: list (`syslin` SIMO linear system).

fmin,fmax
: real scalars (frequency interval).

step
: real (discretization step (logarithmic scale))

comments
: string

frq
: matrix (row by row frequencies)

db,phi
: matrices (magnitudes and phases corresponding to `frq`)

repf
: complex matrix. One row for each frequency response.

Description

Same as Bode but plots only the magnitude.

Examples

```
s=poly(0,'s')
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
gainplot(h,0.01,100,'(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)')
clf()
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
gainplot([h1;h],0.01,100,['h1';'h'])
```

See Also

bode , black , nyquist , freq , repfreq , g_margin , p_margin

Name

gamitg — H-infinity gamma iterations

```
[gopt]=gamitg(G,r,prec [,options]);
```

Parameters

G
: `syslin` list (plant realization)

r
: 1x2 row vector (dimension of `G22`)

prec
: desired relative accuracy on the norm

option
: string `'t'`

gopt
: real scalar, optimal H-infinity gain

Description

`gopt=gamitg(G,r,prec [,options])` returns the H-infinity optimal gain `gopt`.

`G` contains the state-space matrices `[A,B,C,D]` of the plant with the usual partitions:

```
B = ( B1 , B2 ) ,    C = ( C1 ) ,    D = ( D11  D12)
                         ( C2 )          ( D21  D22)
```

These partitions are implicitly given in `r`: `r(1)` and `r(2)` are the dimensions of `D22` (rows x columns)

With `option='t'`, `gamitg` traces each bisection step, i.e., displays the lower and upper bounds and the current test point.

See Also

ccontrg , h_inf

Authors

P. Gahinet

Name

gcare — control Riccati equation

```
[X,F]=gcare(Sl)
```

Parameters

Sl
: linear system (`syslin` list)

X
: symmetric matrix

F
: real matrix

Description

Generalized Control Algebraic Riccati Equation (GCARE). `X` = solution , `F` = gain.

The GCARE for `Sl=[A,B,C,D]` is:

```
(A-B*Si*D'*C)'*X+X*(A-B*Si*D'*C)-X*B*Si*B'*X+C'*Ri*C=0
```

where `S=(eye()+D'*D)`, `Si=inv(S)`, `R=(eye()+D*D')`, `Ri=inv(R)` and `F=-Si*(D'*C+B'*X)` is such that `A+B*F` is stable.

See Also

gfare

Name

gfare — filter Riccati equation

```
[Z,H]=gfare(Sl)
```

Parameters

Sl
: linear system (`syslin` list)

Z
: symmetric matrix

H
: real matrix

Description

Generalized Filter Algebraic Riccati Equation (GFARE). `Z` = solution, `H` = gain.

The GFARE for `Sl=[A,B,C,D]` is:

```
(A-B*D'*Ri*C)*Z+Z*(A-B*D'*Ri*C)'-Z*C'*Ri*C*Z+B*Si*B'=0
```

where `S=(eye()+D'*D)`, `Si=inv(S)`, `R=(eye()+D*D')`, `Ri=inv(R)` and `H=-(B*D'+Z*C')*Ri` is such that `A+H*C` is stable.

See Also

gcare

Name

gfrancis — Francis equations for tracking

```
[L,M,T]=gfrancis(Plant,Model)
```

Parameters

Plant
: `syslin` list

Model
: `syslin` list

L,M,T
: real matrices

Description

Given the the linear plant:

```
x'= F*x + G*u
y = H*x + J*u
```

and the linear model

```
xm'= A*xm + B*um
ym = C*xm + D*um
```

the goal is for the plant to track the model i.e. `e = y - ym ---> 0` while keeping stable the state x(t) of the plant. `u` is given by feedforward and feedback

```
u = L*xm + M*um + K*(x-T*xm) = [K , L-K*T] *(x,xm) + M*um
```

The matrices T,L,M satisfy generalized Francis equations

```
F*T + G*L = T*A
H*T + J*L = C
      G*M = T*B
      J*M = D
```

The matrix `K` must be chosen as stabilizing the pair `(F,G)` See example of use in directory `demos/tracking`.

Examples

```
Plant=ssrand(1,3,5);
[F,G,H,J]=abcd(Plant);
nw=4;nuu=2;A=rand(nw,nw);
st=maxi(real(spec(A)));A=A-st*eye(A);
B=rand(nw,nuu);C=2*rand(1,nw);D=0*rand(C*B);
Model=syslin('c',A,B,C,D);
[L,M,T]=gfrancis(Plant,Model);
norm(F*T+G*L-T*A,1)
norm(H*T+J*L-C,1)
norm(G*M-T*B,1)
norm(J*M-D,1)
```

See Also

lqg , ppol

Name

gtild — tilde operation

```
Gt=gtild(G)
Gt=gtild(G,flag)
```

Parameters

G
: either a polynomial or a linear system (`syslin` list) or a rational matrix

Gt
: same as G

flag
: character string: either `'c'` or `'d'` (optional parameter).

Description

If `G` is a polynomial matrix (or a polynomial), `Gt=gtild(G,'c')` returns the polynomial matrix `Gt(s)=G(-s)'`.

If `G` is a polynomial matrix (or a polynomial), `Gt=gtild(G,'d')` returns the polynomial matrix `Gt=G(1/z)*z^n` where n is the maximum degree of `G`.

For continuous-time systems represented in state-space by a `syslin` list, `Gt = gtild(G,'c')` returns a state-space representation of `G(-s)'` i.e the `ABCD` matrices of `Gt` are `A',-C', B', D'`. If `G` is improper (`D= D(s)`) the `D` matrix of `Gt` is `D(-s)'`.

For discrete-time systems represented in state-space by a `syslin` list, `Gt = gtild(G,'d')` returns a state-space representation of `G(-1/z)'` i.e the (possibly improper) state-space representation of `-z*C*inv(z*A-B)*C + D(1/z)` .

For rational matrices, `Gt = gtild(G,'c')` returns the rational matrix `Gt(s)=G(-s)` and `Gt = gtild(G,'d')` returns the rational matrix `Gt(z)= G(1/z)'`.

The parameter `flag` is necessary when `gtild` is called with a polynomial argument.

Examples

```
//Continuous time
s=poly(0,'s');G=[s,s^3;2+s^3,s^2-5]
Gt=gtild(G,'c')
Gt-horner(G,-s)'   //continuous-time interpretation
Gt=gtild(G,'d');
Gt-horner(G,1/s)'*s^3  //discrete-time interpretation
G=ssrand(2,2,3);Gt=gtild(G);   //State-space (G is cont. time by default)
clean((horner(ss2tf(G),-s))'-ss2tf(Gt))   //Check
// Discrete-time
z=poly(0,'z');
Gss=ssrand(2,2,3);Gss('dt')='d'; //discrete-time
Gss(5)=[1,2;0,1];   //With a constant D matrix
G=ss2tf(Gss);Gt1=horner(G,1/z)';
Gt=gtild(Gss);
Gt2=clean(ss2tf(Gt)); clean(Gt1-Gt2)  //Check
```

```
//Improper systems
z=poly(0,'z');
Gss=ssrand(2,2,3);Gss(7)='d'; //discrete-time
Gss(5)=[z,z^2;1+z,3];    //D(z) is polynomial
G=ss2tf(Gss);Gt1=horner(G,1/z)';  //Calculation in transfer form
Gt=gtild(Gss);    //..in state-space
Gt2=clean(ss2tf(Gt));clean(Gt1-Gt2)  //Check
```

See Also

syslin , horner , factors

Name

h2norm — H2 norm

```
[n]=h2norm(Sl [,tol])
```

Parameters

Sl
: linear system (`syslin` list)

n
: real scalar

Description

produces the H2 norm of a linear continuous time system `Sl`.

(For `Sl` in state-space form `h2norm` uses the observability gramian and for `Sl` in transfer form `h2norm` uses a residue method)

Name

h_cl — closed loop matrix

```
[Acl]=h_cl(P,r,K)
[Acl]=h_cl(P22,K)
```

Parameters

P, P22
: linear system (`syslin` list), augmented plant or nominal plant respectively

r
: 1x2 row vector, dimensions of 2,2 part of `P` (`r=[rows,cols]=size(P22)`)

K
: linear system (`syslin` list), controller

Acl
: real square matrix

Description

Given the standard plant `P` (with `r=size(P22)`) and the controller `K`, this function returns the closed loop matrix `Acl`.

The poles of `Acl` must be stable for the internal stability of the closed loop system.

`Acl` is the A-matrix of the linear system `[I -P22;-K I]^-1` i.e. the A-matrix of `lft(P,r,K)`

See Also

lft

Authors

F. D.

Name

h_inf — H-infinity (central) controller

```
[Sk,ro]=h_inf(P,r,romin,romax,nmax)
[Sk,rk,ro]=h_inf(P,r,romin,romax,nmax)
```

Parameters

P
: `syslin` list : continuous-time linear system (``augmented'' plant given in state-space form or in transfer form)

r
: size of the `P22` plant i.e. 2-vector `[#outputs,#inputs]`

romin,romax
: a priori bounds on `ro` with `ro=1/gama^2`; (`romin=0` usually)

nmax
: integer, maximum number of iterations in the gama-iteration.

Description

`h_inf` computes H-infinity optimal controller for the continuous-time plant `P`.

The partition of `P` into four sub-plants is given through the 2-vector `r` which is the size of the `22` part of `P`.

`P` is given in state-space e.g. `P=syslin('c',A,B,C,D)` with `A,B,C,D` = constant matrices or `P=syslin('c',H)` with `H` a transfer matrix.

`[Sk,ro]=H_inf(P,r,romin,romax,nmax)` returns `ro` in `[romin,romax]` and the central controller `Sk` in the same representation as `P`.

(All calculations are made in state-space, i.e conversion to state-space is done by the function, if necessary).

Invoked with three LHS parameters,

`[Sk,rk,ro]=H_inf(P,r,romin,romax,nmax)` returns `ro` and the Parameterization of all stabilizing controllers:

a stabilizing controller `K` is obtained by `K=lft(Sk,r,PHI)` where `PHI` is a linear system with dimensions `r'` and satisfy:

`H_norm(PHI) < gamma`. `rk (=r)` is the size of the `Sk22` block and `ro = 1/gama^2` after `nmax` iterations.

Algorithm is adapted from Safonov-Limebeer. Note that `P` is assumed to be a continuous-time plant.

See Also

gamitg , ccontrg , leqr

Authors

F.Delebecque INRIA (1990)

Name

h_inf_st — static H_infinity problem

```
[Kopt,gamaopt]=h_inf_stat(D,r)
```

Parameters

D
: real matrix

r
: 1x2 vector

Kopt
: matrix

Description

computes a matrix `Kopt` such that largest singular value of:

`lft(D,r,K)=D11+D12*  K*inv(I-D22*K)*  D21` is minimal (Static `H_infinity` four blocks problem).

D is partionned as `D=[D11 D12; D21 D22]` where `size(D22)=r=[r1 r2]`

Authors

F.D. ;

Name

h_norm — H-infinity norm

```
[hinfnorm [,frequency]]=h_norm(sl [,rerr])
```

Parameters

sl
: the state space system (`syslin` list)

rerr
: max. relative error, default value `1e-8`

hinfnorm
: the infinity norm of `Sl`

frequency
: frequency at which maximum is achieved

Description

produces the infinity norm of a state-space system (the maximum over all frequencies of the maximum singular value).

See Also

linfn , linf , svplot

Name

hankelsv — Hankel singular values

```
[nk2,W]=hankelsv(sl [,tol])
[nk2]=hankelsv(sl [,tol])
```

Parameters

sl
: `syslin` list representing the linear system (state-space).

tol
tolerance parameter for detecting imaginary axis modes (default value is `1000*%eps`).

Description

returns `nk2`, the squared Hankel singular values of `sl` and `W = P*Q` = controllability gramian times observability gramian.

`nk2` is the vector of eigenvalues of `W`.

Examples

```
A=diag([-1,-2,-3]);
sl=syslin('c',A,rand(3,2),rand(2,3));[nk2,W]=hankelsv(sl)
[Q,M]=pbig(W,nk2(2)-%eps,'c');
slr=projsl(sl,Q,M);hankelsv(slr)
```

See Also

balreal , equil , equil1

Name

hinf — H_infinity design of continuous-time systems

```
[AK,BK,CK,DK,(RCOND)] = hinf(A,B,C,D,ncon,nmeas,gamma)
```

Parameters

A
: the n-by-n system state matrix A.

B
: the n-by-m system input matrix B.

C
: the p-by-n system output matrix C.

D
: the p-by-m system matrix D.

ncon
: the number of control inputs. m >= ncon >= 0, p-nmeas >= ncon.

nmeas
: the number of measurements. p >= nmeas >= 0, m-ncon >= nmeas.

gamma
: the parameter gamma used in `H_infinity` design. It is assumed that gamma is sufficiently large so that the controller is admissible. gamma >= 0.

AK
: the n-by-n controller state matrix AK.

BK
: the n-by-nmeas controller input matrix BK.

CK
: the ncon-by-n controller output matrix CK.

DK
: the ncon-by-nmeas controller matrix DK.

RCOND
: a vector containing estimates of the reciprocal condition numbers of the matrices which are to be inverted and estimates of the reciprocal condition numbers of the Riccati equations which have to be solved during the computation of the controller. (See the description of the algorithm in [1].)

  RCOND
  : (1) contains the reciprocal condition number of the control transformation matrix TU,

  RCOND
  : (2) contains the reciprocal condition number of the measurement transformation matrix TY,

  RCOND
  : (3) contains an estimate of the reciprocal condition number of the X-Riccati equation,

  RCOND
  : (4) contains an estimate of the reciprocal condition number of the Y-Riccati equation.

Description

`[AK,BK,CK,DK,(RCOND)] = hinf(A,B,C,D,ncon,nmeas,gamma)` To compute the matrices of an H-infinity (sub)optimal n-state controller

```
          | AK | BK |
      K = |----|----|,
          | CK | DK |
```

for the continuous-time system

```
          | A  | B1  B2  |   | A | B |
      P = |----|---------| = |---|---|,
          | C1 | D11 D12 |   | C | D |
          | C2 | D21 D22 |
```

and for a given value of gamma, where B2 has column size of the number of control inputs (ncon) and C2 has row size of the number of measurements (nmeas) being provided to the controller.

References

[1] P.Hr. Petkov, D.W. Gu and M.M. Konstantinov. Fortran 77 routines for Hinf and H2 design of continuous-time linear control systems. Report98-14, Department of Engineering, Leicester University, August 1998.

Examples

```
//example from Niconet report SLWN1999-12
//Hinf
A=[-1  0  4  5 -3 -2
   -2  4 -7 -2  0  3
   -6  9 -5  0  2 -1
   -8  4  7 -1 -3  0
    2  5  8 -9  1 -4
    3 -5  8  0  2 -6];

B=[-3 -4 -2  1  0
    2  0  1 -5  2
   -5 -7  0  7 -2
    4 -6  1  1 -2
   -3  9 -8  0  5
    1 -2  3 -6 -2];

C=[ 1 -1  2 -4  0 -3
   -3  0  5 -1  1  1
   -7  5  0 -8  2 -2
    9 -3  4  0  3  7
```

```
    0  1 -2  1 -6 -2];

D=[ 1 -2 -3  0  0
    0  4  0  1  0
    5 -3 -4  0  1
    0  1  0  1 -3
    0  0  1  7  1];
Gamma=10.18425636157899;
[AK,BK,CK,DK] = hinf(A,B,C,D,2,2,Gamma)
```

See Also

dhinf

Name

imrep2ss — state-space realization of an impulse response

```
[sl]=imrep2ss(v [,deg])
```

Parameters

v
: vector coefficients of impulse response, `v(:,k)` is the kth sample

deg
: integer (order required)

sl
: `syslin` list

Description

Impulse response to linear system conversion (one input). `v` must have an even number of columns.

Examples

```
s=poly(0,'s');
H=[1/(s+0.5);2/(s-0.4)]   //strictly proper
np=20;w=ldiv(H('num'),H('den'),np);
rep=[w(1:np)';w(np+1:2*np)'];   //The impulse response
H1=ss2tf(imrep2ss(rep))
z=poly(0,'z');
H=(2*z^2-3.4*z+1.5)/(z^2-1.6*z+0.8)     //Proper transfer function
u=zeros(1,20);u(1)=1;
rep=rtitr(H('num'),H('den'),u);   //Impulse rep.
//   <=> rep=ldiv(H('num'),H('den'),20)
w=z*imrep2ss(rep)   //Realization with shifted impulse response
// i.e strictly proper to proper
H2=ss2tf(w);
```

See Also

frep2tf , arl2 , time_id , armax , markp2ss , ldiv

Name

inistate — Estimates the initial state of a discrete-time system

```
X0 = inistate(SYS,Y,U,TOL,PRINTW)
X0 = inistate(A,B,C,Y,U);
X0 = inistate(A,C,Y);

[x0,V,rcnd] = inistate(SYS,Y,U,TOL,PRINTW)
```

Parameters

SYS
: given system, syslin(dt,A,B,C,D)

Y
: the output of the system

U
: the input of the system

TOL
: TOL is the tolerance used for estimating the rank of matrices. If TOL > 0, then the given value of TOL is used as a lower bound for the reciprocal condition number.

  Default: prod(size(matrix))*epsilon_machine where epsilon_machine is the relative machine precision.

PRINTW
: PRINTW is a switch for printing the warning messages.

  =
  : 1: print warning messages;

  =
  : 0: do not print warning messages.

  Default: PRINTW = 0.

X0
: the estimated initial state vector

V
: orthogonal matrix which reduces the system state matrix A to a real Schur form

rcnd
: estimate of the reciprocal condition number of the coefficient matrix of the least squares problem solved.

Description

inistate Estimates the initial state of a discrete-time system, given the (estimated) system matrices, and a set of input/output data.

X0 = inistate(SYS,Y,U,TOL,PRINTW) estimates the initial state X0 of the discrete-time system SYS = (A,B,C,D), using the output data Y and the input data U. The model structure is :

```
    x(k+1) = Ax(k) + Bu(k),    k >= 1,
    y(k)   = Cx(k) + Du(k),
```

The vectors y(k) and u(k) are transposes of the k-th rows of Y and U, respectively.

Instead of the first input parameter SYS (an syslin object), equivalent information may be specified using matrix parameters, for instance, X0 = inistate(A,B,C,Y,U); or X0 = inistate(A,C,Y);

[x0,V,rcnd] = inistate(SYS,Y,U,TOL,PRINTW) returns, besides x0, the orthogonal matrix V which reduces the system state matrix A to a real Schur form, as well as an estimate of the reciprocal condition number of the coefficient matrix of the least squares problem solved.

See Also

findBD , findx0BD

Name

invsyslin — system inversion

```
[sl2]=invsyslin(sl1)
```

Parameters

sl1,sl2
: `syslin` lists (linear systems in state space representation)

Description

Utility function. Computes the state form of the inverse `sl2` of the linear system `sl1` (which is also given in state form).

The `D`-matrix is supposed to be full rank. Old stuff used by `inv(S)` when `S` is a `syslin` list.

See Also

rowregul , inv

Name

kpure — continuous SISO system limit feedback gain

```
K=kpure(sys [,tol])
[K,R]=kpure(sys [,tol])
```

Parameters

sys
: SISO linear system (syslin)

tol
: vector with 2 elements `[epsK epsI]`. `epsK` is a tolerance used to determine if two values of `K` can be considered as equal `epsI` is a tolerance used to determine if a root is imaginary or not. The default value is `[1e-6 1e-6 ]`

K
: Real vector, the vector of gains for which at least one closed loop pole is imaginary.

R
: Complex vector, the imaginary closed loop poles associated with the values of `K`.

Description

`K=kpure(sys)` computes the gains `K` such that the system `sys` feedback by `K(i)` (`sys/.K(i)`) has poles on imaginary axis.

Examples

```
s=poly(0,'s');
h=syslin('c',(s-1)/(1+5*s+s^2+s^3))
xbasc();evans(h)
K=kpure(h)
hf=h/.K(1)
roots(denom(hf))
```

See Also

evans , krac2

Name

krac2 — continuous SISO system limit feedback gain

```
g=krac2(sys)
```

Parameters

sys
: SISO linear system (syslin)

g
: constant

Description

`krac2(sys)` computes the gains `g` such that the system `sys` fedback by g (`sys/.g`) has 2 real equal poles.

Examples

```
h=syslin('c',352*poly(-5,'s')/poly([0,0,2000,200,25,1],'s','c'));
xbasc();evans(h,100)
g=krac2(h)
hf1=h/.g(1);roots(denom(hf1))
hf2=h/.g(2);roots(denom(hf2))
```

See Also

evans , kpure

Name

lcf — normalized coprime factorization

```
[N,M]=lcf(sl)
```

Parameters

sl
: linear system given in state space or transfer function (`syslin` list)

N,M
: two linear systems (`syslin` list)

Description

Computes normalized coprime factorization of the linear dynamic system `sl`.

```
sl = M^-1 N
```

Authors

F. D.; ;

Name

leqr — H-infinity LQ gain (full state)

```
[K,X,err]=leqr(P12,Vx)
```

Parameters

P12
: `syslin` list

Vx
: symmetric nonnegative matrix (should be small enough)

K,X
: two real matrices

err
: a real number (l1 norm of LHS of Riccati equation)

Description

`leqr` computes the linear suboptimal H-infinity LQ full-state gain for the plant `P12=[A,B2,C1,D12]` in continuous or discrete time.

`P12` is a `syslin` list (e.g. `P12=syslin('c',A,B2,C1,D12)`).

```
    [C1' ]               [Q  S]
    [    ]  * [C1 D12] = [    ]
    [D12']               [S' R]
```

`Vx` is related to the variance matrix of the noise `w` perturbing `x`; (usually `Vx=gama^-2*B1*B1'`).

The gain `K` is such that `A + B2*K` is stable.

`X` is the stabilizing solution of the Riccati equation.

For a continuous plant:

```
(A-B2*inv(R)*S')'*X+X*(A-B2*inv(R)*S')-X*(B2*inv(R)*B2'-Vx)*X+Q-S*inv(R)*S'=0
```

```
K=-inv(R)*(B2'*X+S)
```

For a discrete time plant:

```
X-(Abar'*inv((inv(X)+B2*inv(R)*B2'-Vx))*Abar+Qbar=0
```

```
K=-inv(R)*(B2'*inv(inv(X)+B2*inv(R)*B2'-Vx)*Abar+S')
```

with `Abar=A-B2*inv(R)*S'` and `Qbar=Q-S*inv(R)*S'`

The 3-blocks matrix pencils associated with these Riccati equations are:

```
              discrete                          continuous
  |I  -Vx  0|   | A    0    B2|       |I   0   0|   | A    Vx    B2|
z |0   A'  0| - |-Q    I    -S|     s |0   I   0| - |-Q   -A'   -S |
  |0   B2' 0|   | S'   0     R|       |0   0   0|   | S'   -B2'   R|
```

See Also

lqr

Authors

F.D.;

Name

lft — linear fractional transformation

```
[P1]=LFT(P,K)
[P1]=LFT(P,r,K)
[P1,r1]=LFT(P,r,Ps,rs)
```

Parameters

P
: linear system (`syslin` list), the ``augmented'' plant, implicitly partitioned into four blocks (two input ports and two output ports).

K
: linear system (`syslin` list), the controller (possibly an ordinary gain).

r
: 1x2 row vector, dimension of `P22`

Ps
: linear system (`syslin` list), implicitly partitioned into four blocks (two input ports and two output ports).

rs
: 1x2 row vector, dimension of `Ps22`

Description

Linear fractional transform between two standard plants `P` and `Ps` in state space form or in transfer form (`syslin` lists).

`r= size(P22) rs=size(P22s)`

`LFT(P,r, K)` is the linear fractional transform between `P` and a controller `K` (`K` may be a gain or a controller in state space form or in transfer form);

`LFT(P,K)` is `LFT(P,r,K)` with `r`=size of `K` transpose;

`P1= P11+P12*K* (I-P22*K)^-1 *P21`

`[P1,r1]=LFT(P,r,Ps,rs)` returns the generalized (2 ports) lft of `P` and `Ps`.

`P1` is the pair two-port interconnected plant and the partition of `P1` into 4 blocks in given by `r1` which is the dimension of the `22` block of `P1`.

`P` and `R` can be PSSDs i.e. may admit a polynomial `D` matrix.

Examples

```
s=poly(0,'s');
P=[1/s, 1/(s+1); 1/(s+2),2/s]; K= 1/(s-1);
lft(P,K)
lft(P,[1,1],K)
P(1,1)+P(1,2)*K*inv(1-P(2,2)*K)*P(2,1)   //Numerically dangerous!
ss2tf(lft(tf2ss(P),tf2ss(K)))
```

```
lft(P,-1)
f=[0,0;0,1];w=P/.f; w(1,1)
//Improper plant (PID control)
W=[1,1;1,1/(s^2+0.1*s)];K=1+1/s+s
lft(W,[1,1],K); ss2tf(lft(tf2ss(W),[1,1],tf2ss(K)))
```

See Also

sensi , augment , feedback , sysdiag

Name

lin — linearization

```
[A,B,C,D]=lin(sim,x0,u0)
[sl]=lin(sim,x0,u0)
```

Parameters

sim
: function

x0, u0
: vectors of compatible dimensions

A,B,C,D
: real matrices

sl
: `syslin` list

Description

linearization of the non-linear system `[y,xdot]=sim(x,u)` around `x0,u0`.

`sim` is a function which computes `y` and `xdot`.

The output is a linear system (`syslin` list) `sl` or the four matrices `(A,B,C,D)`

For example, if `ftz` is the function passed to ode e.g.

```
[zd]=ftz(t,z,u)
```

and if we assume that `y=x`

`[z]=ode(x0,t0,tf,list(ftz,u))` compute `x(tf)`.

If `simula` is the following function:

```
deff('[y,xd]=simula(x,u)','xd=ftz(tf,x,u); y=x;');
```

the tangent linear system `sl` can be obtained by:

```
[A,B,C,D]=lin(simula,z,u)
sl = syslin('c',A,B,C,D,x0)
```

Examples

```
deff('[y,xdot]=sim(x,u)','xdot=[u*sin(x);-u*x^2];y=xdot(1)+xdot(2)')
sl=lin(sim,1,2);
```

See Also

external , derivat

Name

linf — infinity norm

```
linf(g [,eps],[tol])
```

Parameters

g
: is a `syslin` linear system.

eps
: is error tolerance on n.

tol
: threshold for imaginary axis poles.

Description

returns the L_infinity norm of `g`.

```
n=sup [sigmax(g(jw)]
   w
```

(sigmax largest singular value).

See Also

h_norm , linfn

Name

linfn — infinity norm

```
[x,freq]=linfn(G,PREC,RELTOL,options);
```

Parameters

G
: is a `syslin` list

PREC
: desired relative accuracy on the norm

RELTOL
: relative threshold to decide when an eigenvalue can be considered on the imaginary axis.

options
: available options are `'trace'` or `'cond'`

x
: is the computed norm.

freq
: vector

Description

Computes the Linf (or Hinf) norm of `G` This norm is well-defined as soon as the realization `G=(A,B,C,D)` has no imaginary eigenvalue which is both controllable and observable.

`freq` is a list of the frequencies for which `||G||` is attained,i.e., such that `||G (j om)|| = ||G||`.

If -1 is in the list, the norm is attained at infinity.

If -2 is in the list, `G` is all-pass in some direction so that `||G (j omega)|| = ||G||` for all frequencies omega.

The algorithm follows the paper by G. Robel (AC-34 pp. 882-884, 1989). The case `D=0` is not treated separately due to superior accuracy of the general method when `(A,B,C)` is nearly non minimal.

The `'trace'` option traces each bisection step, i.e., displays the lower and upper bounds and the current test point.

The `'cond'` option estimates a confidence index on the computed value and issues a warning if computations are ill-conditioned

In the general case (`A` neither stable nor anti-stable), no upper bound is prespecified.

If by contrast `A` is stable or anti stable, lower and upper bounds are computed using the associated Lyapunov solutions.

See Also

h_norm

Authors

P. Gahinet

Name

linmeq — Sylvester and Lyapunov equations solver

```
[X(,sep)] = linmeq(task,A,(B,)C,flag,trans(,schur))
```

Parameters

task
: integer option to determine the equation type:

  =1
  : solve the Sylvester equation (1a) or (1b);

  =2
  : solve the Lyapunov equation (2a) or (2b);

  =3
  : solve for the Cholesky factor op(X) the Lyapunov equation (3a) or (3b).

A
: real matrix

B
: real matrix

C
: real matrix

flag
: (optional) integer vector of length 3 or 2 containing options.

  task
  : = 1 : flag has length 3

    flag(1)
    : = 0 : solve the continuous-time equation (1a); otherwise, solve the discrete-time equation (1b).

    flag(2)
    : = 1 : A is (quasi) upper triangular;

    flag(2)
    : = 2 : A is upper Hessenberg;

    otherwise
    : A is in general form.

    flag(3)
    : = 1 : B is (quasi) upper triangular;

    flag(3)
    : = 2 : B is upper Hessenberg;

    otherwise,
    : B is in general form.

  task
  : = 2 : flag has length 2

flag(1)
if 0 solve continuous-time equation (2a), otherwise, solve discrete-time equation (2b).

flag(2)
= 1 : A is (quasi) upper triangular otherwise, A is in general form.

task
= 3 : flag has length 2

flag(1)
= 0 : solve continuous-time equation (3a); otherwise, solve discrete-time equation (3b).

flag(2)
= 1 : A is (quasi) upper triangular; otherwise, A is in general form.

Default: flag(1) = 0, flag(2) = 0 (, flag(3) = 0).

trans
(optional) integer specifying a transposition option.

=
0 : solve the equations (1) - (3) with op(M) = M.

=
1 : solve the equations (1) - (3) with op(M) = M'.

=
2 : solve the equations (1) with op(A) = A'; op(B) = B;

=
3 : solve the equations (1) with op(A) = A; op(B) = B'.

Default: trans = 0.

schur
(optional) integer specifying whether the Hessenberg-Schur or Schur method should be used. Available for task = 1.

= 1 : Hessenberg-Schur method (one matrix is reduced
to Schur form).

= 2 : Schur method (two matrices are reduced to Schur
form).

Default: schur = 1.

X
:

sep
(optional) estimator of Sep(op(A),-op(A)') for (2.a) or Sepd(A,A') for (2.b).

Description

linmeq function for solving Sylvester and Lyapunov equations using SLICOT routines SB04MD, SB04ND, SB04PD, SB04QD, SB04RD, SB03MD, and SB03OD.

```
    [X] = linmeq(1,A,B,C,flag,trans,schur)
[X,sep] = linmeq(2,A,C,flag,trans)
```

```
      [X] = linmeq(2,A,C,flag,trans)
      [X] = linmeq(3,A,C,flag,trans)
```

linmeq solves various Sylvester and Lyapunov matrix equations:

```
      op(A)*X + X*op(B) = C,                                (1a)

      op(A)*X*op(B) + X = C,                                (1b)

      op(A)'*X + X*op(A) = C,                               (2a)

      op(A)'*X*op(A) - X = C,                               (2b)

      op(A)'*(op(X)'*op(X)) + (op(X)'*op(X))*op(A) =
                           -  op(C)'*op(C),                 (3a)

      op(A)'*(op(X)'*op(X))*op(A) - op(X)'*op(X) =
                                  - op(C)'*op(C),           (3b)
```

where op(M) = M, or M'.

Comments

1. For equation (1a) or (1b), when schur = 1, the Hessenberg-Schur method is used, reducing one matrix to Hessenberg form and the other one to a real Schur form. Otherwise, both matrices are reduced to real Schur forms. If one or both matrices are already reduced to Schur/Hessenberg forms, this could be specified by flag(2) and flag(3). For general matrices, the Hessenberg-Schur method could be significantly more efficient than the Schur method.

2. For equation (2a) or (2b), matrix C is assumed symmetric.

3. For equation (3a) or (3b), matrix A must be stable or convergent, respectively.

4. For equation (3a) or (3b), the computed matrix X is the Cholesky factor of the solution, i.e., the real solution is op(X)'*op(X), where X is an upper triangular matrix.

Revisions

V. Sima, Katholieke Univ. Leuven, Belgium, May 1999, May, Sep. 2000. V. Sima, University of Bucharest, Romania, May 2000.

Examples

```
//(1a)
n=40;m=30;
A=rand(n,n);C=rand(n,m);B=rand(m,m);
```

```
X = linmeq(1,A,B,C);
norm(A*X+X*B-C,1)
//(1b)
flag=[1,0,0]
X = linmeq(1,A,B,C,flag);
norm(A*X*B+X-C,1)
//(2a)
A=rand(n,n);C=rand(A);C=C+C';
X = linmeq(2,A,C);
norm(A'*X + X*A -C,1)
//(2b)
X = linmeq(2,A,C,[1 0]);
norm(A'*X*A -X-C,1)
//(3a)
A=rand(n,n);
A=A-(max(real(spec(A)))+1)*eye(); //shift eigenvalues
C=rand(A);
X=linmeq(3,A,C);
norm(A'*X'*X+X'*X*A +C'*C,1)
//(3b)
A = [-0.02, 0.02,-0.10, 0.02,-0.03, 0.12;
      0.02, 0.14, 0.12,-0.10,-0.02,-0.14;
     -0.10, 0.12, 0.05, 0.03,-0.04,-0.04;
      0.02,-0.10, 0.03,-0.06, 0.08, 0.11;
     -0.03,-0.02,-0.04, 0.08, 0.14,-0.07;
      0.12,-0.14,-0.04, 0.11,-0.07, 0.04]

C=rand(A);
X=linmeq(3,A,C,[1 0]);
norm(A'*X'*X*A - X'*X +C'*C,1)
```

See Also

sylv , lyap

Authors

H. Xu, TU Chemnitz, FR Germany, Dec. 1998.

Name

lqe — linear quadratic estimator (Kalman Filter)

```
[K,X]=lqe(P21)
```

Parameters

P21
: `syslin` list

K, X
: real matrices

Description

`lqe` returns the Kalman gain for the filtering problem in continuous or discrete time.

`P21` is a `syslin` list representing the system `P21=[A,B1,C2,D21]` `P21=syslin('c',A,B1,C2,D21)` or `P21=syslin('d',A,B1,C2,D21)`

The input to `P21` is a white noise with variance:

```
     [B1 ]                [Q  S]
BigV=[   ] [ B1' D21'] = [    ]
     [D21]                [S' R]
```

`X` is the solution of the stabilizing Riccati equation and `A+K*C2` is stable.

In continuous time:

```
(A-S*inv(R)*C2)*X+X*(A-S*inv(R)*C2)'-X*C2'*inv(R)*C2*X+Q-S*inv(R)*S'=0
```

```
K=-(X*C2'+S)*inv(R)
```

In discrete time:

```
X=A*X*A'-(A*X*C2'+B1*D21')*pinv(C2*X*C2'+D21*D21')*(C2*X*A'+D21*B1')+B1*B1'
```

```
K=-(A*X*C2'+B1*D21')*pinv(C2*X*C2'+D21*D21')
```

`xhat(t+1)= E(x(t+1)| y(0),...,y(t))` (one-step predicted `x`) satisfies the recursion:

```
xhat(t+1)=(A+K*C2)*xhat(t) - K*y(t).
```

Examples

```
//Assume the equations
//.
//x = Ax + Ge
//y = Cx + v
//with
//E ee' = Q_e,    Evv' = R,    Eev' = N
//
//This is equivalent to
//.
//x = Ax  + B1 w
//y = C2x + D21 w
//with E { [Ge ]  [Ge v]' } = E { [B1w ] [B1w D21w]' } = bigR =
//         [ v ]                  [D21w]
//
//[B1*B1'  B1*D21';
// D21*B1'  D21*D21']
//=
//[G*Q_e*G' G*N;
// N*G' R]

//To find (B1,D21) given (G,Q_e,R,N) form bigR =[G*Q_e*G' G*N;N'*G' R].
//Then [W,Wt]=fullrf(bigR);  B1=W(1:size(G,1),:);
//D21=W(($+1-size(C2,1)):$,:)
//
//P21=syslin('c',A,B1,C2,D21);
//[K,X]=lqe(P21);

//Example:
nx=5;ne=2;ny=3;
A=-diag(1:nx);G=ones(nx,ne);
C=ones(ny,nx); Q_e(ne,ne)=1; R=diag(1:ny); N=zeros(ne,ny);
bigR =[G*Q_e*G' G*N;N'*G' R];
[W,Wt]=fullrf(bigR);B1=W(1:size(G,1),:);
D21=W(($+1-size(C,1)):$,:);
C2=C;
P21=syslin('c',A,B1,C2,D21);
[K,X]=lqe(P21);
//Riccati check:
S=G*N;Q=B1*B1';
(A-S*inv(R)*C2)*X+X*(A-S*inv(R)*C2)'-X*C2'*inv(R)*C2*X+Q-S*inv(R)*S'

//Stability check:
spec(A+K*C)
```

See Also

lqr , observer

Authors

F. D.

Name

lqg — LQG compensator

```
[K]=lqg(P,r)
```

Parameters

P
: `syslin` list (augmented plant) in state-space form

r
: 1x2 row vector = (number of measurements, number of inputs) (dimension of the 2,2 part of `P`)

K
: `syslin` list (controller)

Description

`lqg` computes the linear optimal LQG (H2) controller for the "augmented" plant `P=syslin('c',A,B,C,D)` (continuous time) or `P=syslin('d',A,B,C,D)` (discrete time).

The function `lqg2stan` returns `P` and `r` given the nominal plant, weighting terms and variances of noises.

`K` is given by the following ABCD matrices: `[A+B*Kc+Kf*C+Kf*D*Kc,-Kf,Kc,0]` where `Kc=lqr(P12)` is the controller gain and `Kf=lqe(P21)` is the filter gain. See example in `lqg2stan`.

See Also

lqg2stan , lqr , lqe , h_inf , obscont

Authors

F.D.

Name

lqg2stan — LQG to standard problem

```
[P,r]=lqg2stan(P22,bigQ,bigR)
```

Parameters

P22
: `syslin` list (nominal plant) in state-space form

bigQ
: `[Q,S;S',N]` (symmetric) weighting matrix

bigR
: `[R,T;T',V]` (symmetric) covariance matrix

r
: `1x2` row vector = (number of measurements, number of inputs) (dimension of the 2,2 part of `P`)

P
: `syslin` list (augmented plant)

Description

`lqg2stan` returns the augmented plant for linear LQG (H2) controller design.

`P22=syslin(dom,A,B2,C2)` is the nominal plant; it can be in continuous time (`dom='c'`) or discrete time (`dom='d'`).

```
.
x = Ax + w1 + B2u
y = C2x + w2
```

for continuous time plant.

```
x[n+1]= Ax[n] + w1 + B2u
    y = C2x + w2
```

for discrete time plant.

The (instantaneous) cost function is `[x' u'] bigQ [x;u]`.

The covariance of `[w1;w2]` is `E[w1;w2] [w1',w2'] = bigR`

If `[B1;D21]` is a factor of `bigQ`, `[C1,D12]` is a factor of `bigR` and `[A,B2,C2,D22]` is a realization of P22, then `P` is a realization of `[A,[B1,B2],[C1,-C2],[0,D12;D21,D22]`. The (negative) feedback computed by `lqg` stabilizes `P22`, i.e. the poles of `cl=P22/.K` are stable.

Examples

```
ny=2;nu=3;nx=4;
P22=ssrand(ny,nu,nx);
bigQ=rand(nx+nu,nx+nu);bigQ=bigQ*bigQ';
bigR=rand(nx+ny,nx+ny);bigR=bigR*bigR';
[P,r]=lqg2stan(P22,bigQ,bigR);K=lqg(P,r);  //K=LQG-controller
spec(h_cl(P,r,K))      //Closed loop should be stable
//Same as Cl=P22/.K; spec(Cl('A'))
s=poly(0,'s')
lqg2stan(1/(s+2),eye(2,2),eye(2,2))
```

See Also

lqg , lqr , lqe , obscont , h_inf , augment , fstabst , feedback

Authors

F.D.

Name

lqg_ltr — LQG with loop transform recovery

```
[kf,kc]=lqg_ltr(sl,mu,ro)
```

Parameters

sl
: linear system in state-space form (`syslin` list)

mu,ro
: real positive numbers chosen ``small enough''

kf,kc
: controller and observer Kalman gains.

Description

returns the Kalman gains for:

```
          x = a*x + b*u + l*w1
 (sl)
          y = c*x + mu*I*w2

          z = h*x
```

Cost function:

```
     /+oo
                |
         J    = E(| [z(t)'*z(t) + ro^2*u(t)'*u(t)]dt)
          lqg     |
                  / 0
```

The lqg/ltr approach looks for `L,mu,H,ro` such that: J(lqg) = J(freq) where

```
                 /+oo          *  *             *
         J    =  | tr[S  W  W  S ] + tr[T  T]dw
          freq   |
                 /0
```

and

```
S = (I + G*K)^(-1)
T = G*K*(I+G*K)^(-1)
```

See Also

syslin

Name

lqr — LQ compensator (full state)

```
[K,X]=lqr(P12)
```

Parameters

P12
: `syslin` list (state-space linear system)

K,X
: two real matrices

Description

`lqr` computes the linear optimal LQ full-state gain for the plant `P12=[A,B2,C1,D12]` in continuous or discrete time.

`P12` is a `syslin` list (e.g. `P12=syslin('c',A,B2,C1,D12)`).

The cost function is l2-norm of `z'*z` with `z=C1 x + D12 u` i.e. `[x,u]' * BigQ * [x;u]` where

```
      [C1' ]                [Q  S]
BigQ= [    ]  * [C1 D12] = [    ]
      [D12']                [S' R]
```

The gain `K` is such that `A + B2*K` is stable.

`X` is the stabilizing solution of the Riccati equation.

For a continuous plant:

```
(A-B2*inv(R)*S')'*X+X*(A-B2*inv(R)*S')-X*B2*inv(R)*B2'*X+Q-S*inv(R)*S'=0
```

```
K=-inv(R)*(B2'*X+S)
```

For a discrete plant:

```
X=A'*X*A-(A'*X*B2+C1'*D12)*pinv(B2'*X*B2+D12'*D12)*(B2'*X*A+D12'*C1)+C1'*C1;
```

```
K=-pinv(B2'*X*B2+D12'*D12)*(B2'*X*A+D12'*C1)
```

An equivalent form for X is

```
X=Abar'*inv(inv(X)+B2*inv(r)*B2')*Abar+Qbar
```

with `Abar=A-B2*inv(R)*S'` and `Qbar=Q-S*inv(R)*S'`

The 3-blocks matrix pencils associated with these Riccati equations are:

```
              discrete                          continuous
  |I   0    0|   | A    0    B2|         |I   0   0|   | A    0    B2|
z|0   A'   0| - |-Q    I    -S|        s|0   I   0| - |-Q   -A'   -S|
  |0   B2'  0|   | S'   0     R|         |0   0   0|   | S'  -B2'   R|
```

Caution: It is assumed that matrix R is non singular. In particular, the plant must be tall (number of outputs >= number of inputs).

Examples

```
A=rand(2,2);B=rand(2,1);   //two states, one input
Q=diag([2,5]);R=2;     //Usual notations x'Qx + u'Ru
Big=sysdiag(Q,R);    //Now we calculate C1 and D12
[w,wp]=fullrf(Big);C1=wp(:,1:2);D12=wp(:,3:$);   //[C1,D12]'*[C1,D12]=Big
P=syslin('c',A,B,C1,D12);    //The plant (continuous-time)
[K,X]=lqr(P)
spec(A+B*K)    //check stability
norm(A'*X+X*A-X*B*inv(R)*B'*X+Q,1)  //Riccati check
P=syslin('d',A,B,C1,D12);    // Discrete time plant
[K,X]=lqr(P)
spec(A+B*K)   //check stability
norm(A'*X*A-(A'*X*B)*pinv(B'*X*B+R)*(B'*X*A)+Q-X,1) //Riccati check
```

See Also

lqe , gcare , leqr

Authors

F.D.;

Name

ltitr — discrete time response (state space)

```
[X]=ltitr(A,B,U,[x0])
[xf,X]=ltitr(A,B,U,[x0])
```

Parameters

A,B
: real matrices of appropriate dimensions

U,X
: real matrices

x0,xf
: real vectors (default value=0 for `x0`))

Description

calculates the time response of the discrete time system

```
x[t+1] = Ax[t] + Bu[t].
```

The inputs `ui`'s are the columns of the `U` matrix

```
U=[u0,u1,...,un];
```

`x0` is the vector of initial state (default value : 0) ;

`X` is the matrix of outputs (same number of columns as `U`).

```
X=[x0,x1,x2,...,xn]
```

`xf` is the vector of final state `xf=X[n+1]`

Examples

```
A=eye(2,2);B=[1;1];
x0=[-1;-2];
u=[1,2,3,4,5];
x=ltitr(A,B,u,x0)
x1=A*x0+B*u(1)
x2=A*x1+B*u(2)
```

```
x3=A*x2+B*u(3) //....
```

See Also

rtitr , flts

Name

m_circle — plots the complex plane iso-gain contours of y/(1+y)

```
m_circle()
m_circle(gain)
```

Parameters

gain
: vector of gains (in DB). The default value is

  gain
  : =[-12 -8 -6 -5 -4 -3 -2 -1.4 -1 -.5 0.25 0.5 0.7 1 1.4 2 2.3 3 4 5 6 8 12]

Description

`m_circle` draws the iso-gain contours given by then `gain` argument in the complex plane (Re,Im).

The default value for `gain` is:

```
[-12 -8 -6 -5 -4 -3 -2 -1.4 -1 -.5 0.25 0.5 0.7 1 1.4 2 2.3 3 4 5 6 8 12]
```

`m_circle` is used with `nyquist`.

Examples

```
//Example 1 :
  s=poly(0,'s')
  h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
  nyquist(h,0.01,100,'(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)')
  m_circle();
//Example 2:
  xbasc();
  h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
  nyquist([h1;h],0.01,100,['h1';'h'])
  m_circle([-8 -6 -4]);
```

See Also

nyquist , chart , black

Authors

S.Steer.;

Name

macglov — Mac Farlane Glover problem

```
[P,r]=macglov(Sl)
```

Parameters

Sl
: linear system (`syslin` list)

P
: linear system (`syslin` list), ``augmented" plant

r
: 1x2 vector, dimension of `P22`

Description

`[P,r]=macglov(Sl)` returns the standard plant `P` for the Glover-McFarlane problem.

For this problem ro_optimal = 1-hankel_norm(`[N,M]`) with `[N,M]=lcf(sl)` (Normalized coprime factorization) i.e.

gama_optimal = `1/sqrt(ro_optimal)`

Authors

F. Delebecque INRIA

Name

markp2ss — Markov parameters to state-space

```
[sl]=markp2ss(markpar,n,nout,nin)
```

Parameters

markpar
: matrix

n,nout,nin
: integers

Sl
: `syslin` list

Description

given a set of `n` Markov parameters stacked in the (row)-matrix `markpar` of size `nout`X`(n*nin)` `markp2ss` returns a state-space linear system `sl` (`syslin` list) such that with `[A,B,C,D]=abcd(sl)`:

```
C*B = markpar(1:nout,1:nin),
C*A*B =markpar(1:nout,nin+1:2*nin),....
```

Examples

```
W=ssrand(2,3,4);   //random system with 2 outputs and 3 inputs
[a,b,c,d]=abcd(W);
markpar=[c*b,c*a*b,c*a^2*b,c*a^3*b,c*a^4*b];
S=markp2ss(markpar,5,2,3);
[A,B,C,D]=abcd(S);
Markpar=[C*B,C*A*B,C*A^2*B,C*A^3*B,C*A^4*B];
norm(markpar-Markpar,1)
//Caution... c*a^5*b is not C*A^5*B !
```

See Also

frep2tf , tf2ss , imrep2ss

Name

minreal — minimal balanced realization

```
slb=minreal(sl [,tol])
```

Parameters

sl,slb
: `syslin` lists

tol
: real (threshold)

Description

`[ae,be,ce]=minreal(a,b,c,domain [,tol])` returns the balanced realization of linear system `sl` (`syslin` list).

`sl` is assumed stable.

`tol` threshold used in `equil1`.

Examples

```
A=[-eye(2,2),rand(2,2);zeros(2,2),-2*eye(2,2)];
B=[rand(2,2);zeros(2,2)];C=rand(2,4);
sl=syslin('c',A,B,C);
slb=minreal(sl);
ss2tf(sl)
ss2tf(slb)
ctr_gram(sl)
clean(ctr_gram(slb))
clean(obs_gram(slb))
```

See Also

minss , balreal , arhnk , equil , equil1

Authors

S. Steer INRIA 1987

Name

minss — minimal realization

```
[slc]=minss( sl [,tol])
```

Parameters

sl,slc
: `syslin` lists (linear systems in state-space form)

tol
: real (threshold for rank determination (see `contr`))

Description

`minss` returns in `slc` a minimal realization of `sl`.

Examples

```
sl=syslin('c',[1 0;0 2],[1;0],[2 1]);
ssprint(sl);
ssprint(minss(sl))
```

See Also

contr , minreal , arhnk , contrss , obsvss , balreal

Name

mucomp — mu (structured singular value) calculation

```
[BOUND, D, G] = mucomp(Z, K, T)
```

Parameters

Z
: the complex n-by-n matrix for which the structured singular value is to be computed

K
: the vector of length m containing the block structure of the uncertainty.

T
: the vector of length m indicating the type of each block. T(I) = 1 if the corresponding block is real T(I) = 2 if the corresponding block is complex.

BOUND
: the upper bound on the structured singular value.

D, G
: vectors of length n containing the diagonal entries of the diagonal matrices D and G, respectively, such that the matrix `Z'*D^2*Z + sqrt(-1)*(G*Z-Z'*G) - bound^2*D^2` is negative semidefinite.

Description

To compute an upper bound on the structured singular value for a given square complex matrix and given block structure of the uncertainty.

Reference

Slicot routine AB13MD.

Name

narsimul — armax simulation (using rtitr)

```
[z]=narsimul(a,b,d,sig,u,[up,yp,ep])
[z]=narsimul(ar,u,[up,yp,ep])
```

Description

ARMAX simulation. Same as arsimul but the method is different the simulation is made with rtitr

Authors

J-Ph. Chancelier ENPC Cergrene; ;

Name

nehari — Nehari approximant

```
[x]=nehari(R [,tol])
```

Parameters

R
: linear system (`syslin` list)

x
: linear system (`syslin` list)

tol
: optional threshold

Description

`[x]=nehari(R [,tol])returns` the Nehari approximant of `R`.

`R` = linear system in state-space representation (`syslin` list).

`R` is strictly proper and `- R~` is stable (i.e. `R` is anti stable).

```
|| R - X ||oo = min || R - Y ||oo
              Y in Hoo
```

Name

noisegen — noise generation

```
b=noisegen(pas,Tmax,sig)
```

Description

generates a Scilab function `[b]=Noise(t)` where `Noise(t)` is a piecewise constant function (constant on `[k*pas,(k+1)*pas]`). The value on each constant interval are random values from i.i.d Gaussian variables of standard deviation sig. The function is constant for `t<=0` and `t>=Tmax`.

Examples

```
noisegen(0.5,30,1.0);
x=-5:0.01:35;
y=feval(x,Noise);
plot(x,y);
```

Name

nyquist — nyquist plot

```
nyquist( sl,[fmin,fmax] [,step] [,comments] )
nyquist( sl, frq [,comments] )
nyquist(frq,db,phi [,comments])
nyquist(frq, repf [,comments])
```

Parameters

sl
: `syslin` list (SIMO linear system in continuous or discrete time)

fmin,fmax
: real scalars (frequency bounds (in Hz))

step
: real (logarithmic discretization step)

comments
: string vector (captions).

frq
: vector or matrix of frequencies (in Hz) (one row for each output of `sl`).

db,phi
: real matrices of modulus (in Db) and phases (in degree) (one row for each output of `sl`).

repf
: matrix of complex numbers. Frequency response (one row for aech output of `sl`)

Description

Nyquist plot i.e Imaginary part versus Real part of the frequency response of `sl`.

For continous time systems `sl(2*%i*%pi*w)` is plotted. For discrete time system or discretized systems `sl(exp(2*%i*%pi*w*fd)` is used (`fd=1` for discrete time systems and `fd=sl('dt')` for discretized systems)

`sl` can be a continuous-time or discrete-time SIMO system (see `syslin`). In case of multi-output the outputs are plotted with different symbols.

The frequencies are given by the bounds `fmin,fmax` (in Hz) or by a row-vector (or a matrix for multi-output) `frq`.

`step` is the (logarithmic) discretization step. (see `calfrq` for the choice of default value).

`comments` is a vector of character strings (captions).

`db,phi` are the matrices of modulus (in Db) and phases (in degrees). (One row for each response).

`repf` is a matrix of complex numbers. One row for each response.

Default values for `fmin` and `fmax` are `1.d-3`, `1.d+3` if `sl` is continuous-time or `1.d-3`, `0.5`/sl.dt (nyquist frequency) if `sl` is discrete-time.

Automatic discretization of frequencies is made by `calfrq`.

Examples

```
clf();
s=poly(0,'s');
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01));
comm='(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)';
nyquist(h,0.01,100,comm);
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
clf();
nyquist([h1;h],0.01,100,['h1';'h'])
clf();nyquist([h1;h])
```

See Also

bode, black, calfrq, freq, repfreq, phasemag

Name

obs_gram — observability gramian

```
Go=obs_gram(A,C [,dom])
Go=obs_gram(sl)
```

Parameters

A,C
: real matrices (of appropriate dimensions)

dom
: string (`"d'` or `"c"` (default value))

sl
: : `syslin` list

Description

Observability gramian of the pair `(A,C)` or linear system `sl` (`syslin` list). `dom` is the domain which can be

"c"
: continuous system (default)

"d"
: discrete system

Examples

```
A=-diag(1:3);C=rand(2,3);
Go=obs_gram(A,C,'c');     // <=> w=syslin('c',A,[],C); Go=obs_gram(w);
norm(Go*A+A'*Go+C'*C,1)
norm(lyap(A,-C'*C,'c')-Go,1)
A=A/4; Go=obs_gram(A,C,'d');    //discrete time case
norm(lyap(A,-C'*C,'d')-Go,1)
```

See Also

ctr_gram , obsvss , obsv_mat , lyap

Name

obscont — observer based controller

```
[K]=obscont(P,Kc,Kf)
[J,r]=obscont(P,Kc,Kf)
```

Parameters

P
: `syslin` list (nominal plant) in state-space form, continuous or discrete time

Kc
: real matrix, (full state) controller gain

Kf
: real matrix, filter gain

K
: `syslin` list (controller)

J
: `syslin` list (extended controller)

r
: 1x2 row vector

Description

`obscont` returns the observer-based controller associated with a nominal plant `P` with matrices `[A,B,C,D]` (`syslin` list).

The full-state control gain is `Kc` and the filter gain is `Kf`. These gains can be computed, for example, by pole placement.

`A+B*Kc` and `A+Kf*C` are (usually) assumed stable.

`K` is a state-space representation of the compensator `K: y->u` in:

`xdot = A x + B u, y=C x + D u, zdot= (A + Kf C)z -Kf y +B u, u=Kc z`

`K` is a linear system (`syslin` list) with matrices given by: `K=[A+B*Kc+Kf*C+Kf*D*Kc,Kf,-Kc]`.

The closed loop feedback system `Cl: v ->y` with (negative) feedback `K` (i.e. `y = P u, u = v - K y`, or

```
xdot = A x + B u,
y = C x + D u,
zdot = (A + Kf C) z - Kf y + B u,
u = v -F z
```

) is given by `Cl = P/.(-K)`

The poles of `Cl( spec(cl('A')) )` are located at the eigenvalues of `A+B*Kc` and `A+Kf*C`.

Invoked with two output arguments `obscont` returns a (square) linear system `K` which parametrizes all the stabilizing feedbacks via a LFT.

Let `Q` an arbitrary stable linear system of dimension `r(2)xr(1)` i.e. number of inputs x number of outputs in `P`. Then any stabilizing controller `K` for `P` can be expressed as `K=lft(J,r,Q)`. The controller which corresponds to `Q=0` is `K=J(1:nu,1:ny)` (this `K` is returned by `K=obscont(P,Kc,Kf)`). `r` is `size(P)` i.e the vector [number of outputs, number of inputs];

Examples

```
ny=2;nu=3;nx=4;P=ssrand(ny,nu,nx);[A,B,C,D]=abcd(P);
Kc=-ppol(A,B,[-1,-1,-1,-1]);  //Controller gain
Kf=-ppol(A',C',[-2,-2,-2,-2]);Kf=Kf';    //Observer gain
cl=P/.(-obscont(P,Kc,Kf));spec(cl('A'))   //closed loop system
[J,r]=obscont(P,Kc,Kf);
Q=ssrand(nu,ny,3);Q('A')=Q('A')-(maxi(real(spec(Q('A'))))+0.5)*eye(Q('A'))
//Q is a stable parameter
K=lft(J,r,Q);
spec(h_cl(P,K))  // closed-loop A matrix (should be stable);
```

See Also

ppol , lqg , lqr , lqe , h_inf , lft , syslin , feedback , observer

Authors

F.D. ; ;

Name

observer — observer design

```
Obs=observer(Sys,J)
[Obs,U,m]=observer(Sys [,flag,alfa])
```

Parameters

Sys
: `syslin` list (linear system)

J
: nx x ny constant matrix (output injection matrix)

flag
: character strings (`'pp'` or `'st'` (default))

alfa
: location of closed-loop poles (optional parameter, default=-1)

Obs
: linear system (`syslin` list), the observer

U
: orthogonal matrix (see `dt_ility`)

m
: integer (dimension of unstable unobservable (`st`) or unobservable (`pp`) subspace)

Description

`Obs=observer(Sys,J)` returns the observer `Obs=syslin(td,A+J*C,[B+J*D,-J],eye(A))` obtained from `Sys` by a `J` output injection. (td is the time domain of `Sys`). More generally, `observer` returns in `Obs` an observer for the observable part of linear system `Sys`: `dotx=A x + Bu, y=Cx + Du` represented by a `syslin` list. `Sys` has `nx` state variables, `nu` inputs and `ny` outputs. `Obs` is a linear system with matrices `[Ao,Bo,Identity]`, where `Ao` is `no x no`, `Bo` is `no x (nu+ny)`, `Co` is `no x no` and `no=nx-m`.

Input to `Obs` is `[u,y]` and output of `Obs` is:

xhat=estimate of x modulo unobservable subsp. (case `flag='pp'`) or

xhat=estimate of x modulo unstable unobservable subsp. (case `flag='st'`)

case `flag='st'`: z=H*x can be estimated with stable observer iff `H*U(:,1:m)=0` and assignable poles of the observer are set to `alfa(1),alfa(2),...`

case `flag='pp'`: z=H*x can be estimated with given error spectrum iff `H*U(:,1:m)=0` all poles of the observer are assigned and set to `alfa(1),alfa(2),...`

If H satifies the constraint: `H*U(:,1:m)=0` (ker(H) contains unobs-subsp. of Sys) one has H*U=[0,H2] and the observer for z=H*x is H2*Obs with H2=H*U(:,m+1:nx) i.e. Co, the C-matrix of the observer for H*x, is Co=H2.

In the particular case where the pair `(A,C)` of `Sys` is observable, one has `m=0` and the linear system `U*Obs` (resp. `H*U*Obs`) is an observer for `x` (resp. `Hx`). The error spectrum is `alpha(1),alpha(2),...,alpha(nx)`.

Examples

```
nx=5;nu=1;ny=1;un=3;us=2;Sys=ssrand(ny,nu,nx,list('dt',us,us,un));
//nx=5 states, nu=1 input, ny=1 output,
//un=3 unobservable states, us=2 of them unstable.
[Obs,U,m]=observer(Sys);  //Stable observer (default)
W=U';H=W(m+1:nx,:);[A,B,C,D]=abcd(Sys);  //H*U=[0,eye(no,no)];
Sys2=ss2tf(syslin('c',A,B,H))  //Transfer u-->z
Idu=eye(nu,nu);Sys3=ss2tf(H*U(:,m+1:$)*Obs*[Idu;Sys])
//Transfer u-->[u;y=Sys*u]-->Obs-->xhat-->HUxhat=zhat  i.e. u-->output of Obs
//this transfer must equal Sys2, the u-->z transfer  (H2=eye).

//Assume a Kalman model
//dotx = A x + B u + G w
// y   = C x + D u + H w + v
//with Eww' = QN, Evv' = RN, Ewv' = NN
//To build a Kalman observer:
//1-Form BigR = [G*QN*G'         G*QN*H'+G*NN;
//               H*QN*G'+NN*G'   H*QN*H'+RN];
//the covariance matrix of the noise vector [Gw;Hw+v]
//2-Build the plant P21 : dotx = A x + B1 e ; y = C2 x + D21 e
//with e a unit white noise.
// [W,Wt]=fullrf(BigR);
//B1=W(1:size(G,1),:);D21=W(($+1-size(C,1)):$,:);
//C2=C;
//P21=syslin('c',A,B1,C2,D21);
//3-Compute the Kalman gain
//L = lqe(P21);
//4- Build an observer for the plant [A,B,C,D];
//Plant = syslin('c',A,B,C,D);
//Obs = observer(Plant,L);
//Test example:
A=-diag(1:4);
B=ones(4,1);
C=B'; D= 0; G=2*B; H=-3; QN=2;
RN=5; NN=0;
BigR = [G*QN*G'         G*QN*H'+G*NN;
        H*QN*G'+NN*G'   H*QN*H'+RN];
[W,Wt]=fullrf(BigR);
B1=W(1:size(G,1),:);D21=W(($+1-size(C,1)):$,:);
C2=C;
P21=syslin('c',A,B1,C2,D21);
L = lqe(P21);
Plant = syslin('c',A,B,C,D);
Obs = observer(Plant,L);
spec(Obs.A)
```

See Also

dt_ility , unobs , stabil

Authors

F.D.

Name

obsv_mat — observability matrix

```
[O]=obsv_mat(A,C)
[O]=obsv_mat(sl)
```

Parameters

A,C,O
: real matrices

sl
: `syslin` list

Description

`obsv_mat` returns the observability matrix:

```
O=[C; CA; CA^2;...; CA^(n-1) ]
```

See Also

contrss , obsvss , obs_gram

Name

obsvss — observable part

```
[Ao,Bo,Co]=obsvss(A,B,C [,tol])
[slo]=obsvss(sl [,tol])
```

Parameters

A,B,C,Ao,Bo,Co
: real matrices

sl,slo
: `syslin` lists

tol
: real (threshold) (default value `100*%eps`)

Description

`slo=(Ao,Bo,Co)` is the observable part of linear system `sl=(A,B,C)` (`syslin` list)

`tol` threshold to test controllability (see `contr`); default value = `100*%eps`

See Also

contr , contrss , obsv_mat , obs_gram

Name

p_margin — phase margin and associated crossover frequency

```
[phm,fr] = p_margin(h)
phm=p_margin(h)
```

Parameters

h
: a SISO linear system (see :syslin).

phm
: a number, the phase margin in degree if it exists or an empty matrix.

fr
: a number, the corresponding frequency (in hz) or an empty matrix.

Description

Given a SISO linear system in continuous or discrete time, `p_margin` returns `phm`, the phase margin in degree of `h` and `fr`, the achieved corresponding frequency in hz.

The phase margin is the values of the phase at frequency points where the nyquist plot of `h` crosses the unit circle. In other words the phase margin is the difference between the phase of the frequency response of `h` and -180° when the gain of `h` is 1.

The algorithm uses polynomial root finder to solve the equations:

h(s)*h(-s)=1
: for the continuous time case.

h(z)*h(1/z)=1
: for the discrete time case.

Examples

```
//continuous case
h=syslin('c',-1+%s,3+2*%s+%s^2)
[p,fr]=p_margin(h)
[p,fr]=p_margin(h+0.7)
show_margins(h+0.7,'nyquist')

//discrete case
h = syslin(0.1,0.04798*%z+0.0464,%z^2-1.81*%z+0.9048);//ok
[p ,f]=p_margin(h)
show_margins(h,'nyquist')
```

See Also

p_margin, show_margins, repfreq, black, bode, chart, nyquist

Authors

Serge Steer, INRIA

Name

parrot — Parrot's problem

```
K=parrot(D,r)
```

Parameters

D,K
: matrices

r
: 1X2 vector (dimension of the 2,2 part of `D`)

Description

Given a matrix `D` partionned as `[D11 D12; D21 D22]` where `size(D22)=r=[r1,r2]` compute a matrix `K` such that largest singular value of `[D11 D12; D21 D22+K]` is minimal (Parrot's problem)

See Also

h_inf_st

Name

pfss — partial fraction decomposition

```
elts=pfss(Sl)
elts=pfss(Sl,rmax)
elts=pfss(Sl,'cord')
elts=pfss(Sl,rmax,'cord')
```

Parameters

Sl

`syslin` list (state-space or transfer linear system) rmax : real number controlling the conditioning of block diagoanalization cord : character string `'c'` or `'d'`.

Description

Partial fraction decomposition of the linear system `Sl` (in state-space form, transfer matrices are automatically converted to state-space form by `tf2ss`):

`elts` is the list of linear systems which add up to `Sl` i.e. `elts=list(S1,S2,S3,...,Sn)` with:

`Sl = S1 + S2 +... +Sn`.

Each `Si` contains some poles of S according to the block-diagonalization of the `A` matrix of `S`.

For non proper systems the polynomial part of `Sl` is put in the last entry of `elts`.

If `Sl` is given in transfer form, it is first converted into state-space and each subsystem `Si` is then converted in transfer form.

The A matrix is of the state-space is put into block diagonal form by function `bdiag`. The optional parameter `rmax` is sent to `bdiag`. If `rmax` should be set to a large number to enforce block-diagonalization.

If the optional flag `cord='c'` is given the elements in `elts` are sorted according to the real part (resp. magnitude if `cord='d'`) of the eigenvalues of A matrices.

Examples

```
W=ssrand(1,1,6);
elts=pfss(W);
W1=0;for k=1:size(elts), W1=W1+ss2tf(elts(k));end
clean(ss2tf(W)-W1)
```

See Also

pbig , bdiag , coffg , dtsi

Authors

F.D.;

Name

phasemag — phase and magnitude computation

```
[phi,db]=phasemag(z [,mod])
```

Parameters

z
: matrix or row vector of complex numbers.

mod
: character string

  mod='c'
  : "continuous" representation between -infinity and +360 degrees (default)

  mod='m'
  : representation between -360 and 0 degrees

phi
: phases (in degree) of `z`.

db
: magnitude (in Db)

Description

`phasemag` computes the phases and magnitudes of the entries of a complex matrix. For `mod='c'phasemag` computes `phi(:,i+1)` to minimize the distance with `phi(:,i)`, i.e. it tries to obtain a "continuous representation" of the phase.

To obtain the phase between -%pi and %pi use `phi=atan(imag(z),real(z))`

Examples

```
s=poly(0,'s');
h=syslin('c',1/((s+5)*(s+10)*(100+6*s+s*s)*(s+.3)));
[frq,rf]=repfreq(h,0.1,20,0.005);
scf();
plot2d(frq',phasemag(rf,'c')');
scf();
plot2d(frq',phasemag(rf,'m')');
```

See Also

repfreq , gainplot , atan , bode

Name

ppol — pole placement

```
[K]=ppol(A,B,poles)
```

Parameters

A,B
: real matrices of dimensions `nxn` and `nxm`.

poles
: real or complex vector of dimension `n`.

K
: real matrix (negative feedback gain)

Description

`K=ppol(A,B,poles)` returns a mxn gain matrix `K` such that the eigenvalues of `A-B*K` are `poles`. The pair `(A,B)` must be controllable. Complex number in `poles` must appear in conjugate pairs.

An output-injection gain `F` for `(A,C)` is obtained as follows:

```
Ft=ppol(A',C',poles); F=Ft'
```

The algorithm is by P.H. Petkov.

Examples

```
A=rand(3,3);B=rand(3,2);
F=ppol(A,B,[-1,-2,-3]);
spec(A-B*F)
```

See Also

canon , stabil

Name

prbs_a — pseudo random binary sequences generation

```
[u]=prbs_a(n,nc,[ids])
```

Description

generation of pseudo random binary sequences `u=[u0,u1,...,u_(n-1)]` u takes values in {-1,1} and changes at most nc times its sign. ids can be used to fix the date at which u must change its sign ids is then an integer vector with values in `[1:n]`.

Examples

```
u=prbs_a(50,10);
plot2d2("onn",(1:50)',u',1,"151",' ',[0,-1.5,50,1.5]);
```

Name

projsl — linear system projection

```
[slp]=projsl(sl,Q,M)
```

Parameters

sl,slp
: `syslin` lists

Q,M
matrices (projection factorization)

Description

`slp`= projected model of `sl` where `Q*M` is the full rank factorization of the projection.

If `(A,B,C,D)` is the representation of `sl`, the projected model is given by `(M*A*Q,M*B,C*Q,D)`.

Usually, the projection `Q*M` is obtained as the spectral projection of an appropriate auxiliary matrix `W` e.g. `W` = product of (weighted) gramians or product of Riccati equations.

Examples

```
rand('seed',0);sl=ssrand(2,2,5);[A,B,C,D]=abcd(sl);poles=spec(A)
[Q,M]=pbig(A,0,'c');  //keeping unstable poles
slred=projsl(sl,Q,M);spec(slred('A'))
sl('D')=rand(2,2);  //making proper system
trzeros(sl)  //zeros of sl
wi=inv(sl);  //wi=inverse in state-space
[q,m]=psmall(wi('A'),2,'d');  //keeping small zeros (poles of wi) i.e. abs(z)<2
slred2=projsl(sl,q,m);
trzeros(slred2)  //zeros of slred2 = small zeros of sl
//  Example keeping second order modes
A=diag([-1,-2,-3]);
sl=syslin('c',A,rand(3,2),rand(2,3));[nk2,W]=hankelsv(sl)
[Q,M]=pbig(W,nk2(2)-%eps,'c');    //keeping 2 eigenvalues of W
slr=projsl(sl,Q,M);  //reduced model
hankelsv(slr)
```

See Also

pbig

Authors

F. D.;

Name

reglin — Linear regression

```
[a,b,sig]=reglin(x,y)
```

Description

solve the regression problem y=a*x+ b in the least square sense. `sig` is the standard deviation of the residual. `x` and `y` are two matrices of size `x(p,n)` and `y(q,n)`, where `n` is the number of samples.

The estimator `a` is a matrix of size `(q,p)` and `b` is a vector of size `(q,1)`

```
// simulation of data for a(3,5) and b(3,1)
x=rand(5,100);
aa=testmatrix('magi',5);aa=aa(1:3,:);
bb=[9;10;11]
y=aa*x +bb*ones(1,100)+ 0.1*rand(3,100);
// identification
[a,b,sig]=reglin(x,y);
maxi(abs(aa-a))
maxi(abs(bb-b))
// an other example : fitting a polynom
f=1:100; x=[f.*f; f];
y= [ 2,3]*x+ 10*ones(f) + 0.1*rand(f);
[a,b]=reglin(x,y)
```

See Also

pinv , leastsq , qr

Name

repfreq — frequency response

```
[ [frq,] repf]=repfreq(sys,fmin,fmax [,step])
[ [frq,] repf]=repfreq(sys [,frq])
[ frq,repf,splitf]=repfreq(sys,fmin,fmax [,step])
[ frq,repf,splitf]=repfreq(sys [,frq])
```

Parameters

sys
: `syslin` list : SIMO linear system

fmin,fmax
: two real numbers (lower and upper frequency bounds)

frq
: real vector of frequencies (Hz)

step
: logarithmic discretization step

splitf
: vector of indexes of critical frequencies.

repf
: vector of the complex frequency response

Description

`repfreq` returns the frequency response calculation of a linear system. If `sys(s)` is the transfer function of `Sys`, `repf(k)` equals `sys(s)` evaluated at `s= %i*frq(k)*2*%pi` for continuous time systems and at `exp(2*%i*%pi*dt*frq(k))` for discrete time systems (`dt` is the sampling period).

`db(k)` is the magnitude of `repf(k)` expressed in dB i.e. `db(k)=20*log10(abs(repf(k)))` and `phi(k)` is the phase of `repf(k)` expressed in degrees.

If `fmin,fmax,step` are input parameters, the response is calculated for the vector of frequencies `frq` given by: `frq=[10.^((log10(fmin)):step:(log10(fmax))) fmax];`

If `step` is not given, the output parameter `frq` is calculated by `frq=calfrq(sys,fmin,fmax)`.

Vector `frq` is splitted into regular parts with the `split` vector. `frq(splitf(k):splitf(k+1)-1)` has no critical frequency. `sys` has a pole in the range `[frq(splitf(k)),frq(splitf(k)+1)]` and no poles outside.

Examples

```
A=diag([-1,-2]);B=[1;1];C=[1,1];
Sys=syslin('c',A,B,C);
frq=0:0.02:5;w=frq*2*%pi; //frq=frequencies in Hz ;w=frequencies in rad/sec;
[frq1,rep] =repfreq(Sys,frq);
[db,phi]=dbphi(rep);
Systf=ss2tf(Sys)    //Transfer function of Sys
```

```
x=horner(Systf,w(2)*sqrt(-1))    // x is Systf(s) evaluated at s = i w(2)
rep=20*log(abs(x))/log(10)   //magnitude of x in dB
db(2)    // same as rep
ang=atan(imag(x),real(x));   //in rad.
ang=ang*180/%pi              //in degrees
phi(2)
repf=repfreq(Sys,frq);
repf(2)-x
```

See Also

bode , freq , calfrq , horner , nyquist , dbphi

Authors

S. S.

Name

ric_desc — Riccati equation

```
X=ric_desc(H [,E))
[X1,X2,zero]=ric_desc(H [,E])
```

Parameters

H,E
: real square matrices

X1,X2
: real square matrices

zero
: real number

Description

Riccati solver with hamiltonian matrices as inputs.

In the continuous time case calling sequence is `ric_descr(H)` (one input):

Riccati equation is:

```
 (Ec)   A'*X + X*A + X*R*X -Q = 0.
```

Defining the hamiltonian matrix `H` by:

```
H = [A  R;
     Q -A']
```

with the calling sequence `[X1,X2,zero]=ric_descr(H)`, the solution `X` is given by `X=X1/X2`.

`zero` = L1 norm of rhs of (`Ec`)

The solution `X` is also given by `X=riccati(A,Q,R,'c'))`

In the discrete-time case calling sequence is `ric_descr(H,E)` (two inputs):

The Riccati equation is:

```
  (Ed)  A'*X*A-(A'*X*B*(R+B'*X*B)^-1)*(B'*X*A)+C-X = 0.
```

Defining `G=B/R*B'` and the hamiltonian pencil `(E,H)` by:

```
E=[eye(n,n),G;                H=[A, 0*ones(n,n);
   0*ones(n,n),A']              -C, eye(n,n)];
```

with the calling sequence `[X1,X2,err]=ric_descr(H,E)`, the solution `X` is given by `X=X1/X2`.

`zero`= L1 norm of rhs of (`Ed`)

The solution `X` is also given by `X=riccati(A,G,C,'d')` with `G=B/R*B'`

See Also

riccati

Name

ricc — Riccati equation

```
[X,RCOND,FERR]=ricc(A,B,C,"cont""method")
[X,RCOND,FERR]=ricc(F,G,H,"disc","method")
```

Parameters

A,B,C
: real matrices of appropriate dimensions

F,G,H
: real matrices of appropriate dimensions

X
: real matrix

"cont","disc"'
: imposed string (flag for continuous or discrete)

method
: 'schr' or 'sign' for continuous-time systems and 'schr' or 'invf' for discrete-tyme systems

Description

Riccati solver.

Continuous time:

```
X=ricc(A,B,C,'cont')
```

gives a solution to the continuous time ARE

```
A'*X+X*A-X*B*X+C=0 .
```

`B` and `C` are assumed to be nonnegative definite. `(A,G)` is assumed to be stabilizable with `G*G'` a full rank factorization of `B`.

`(A,H)` is assumed to be detectable with `H*H'` a full rank factorization of `C`.

Discrete time:

```
X=ricc(F,G,H,'disc')
```

gives a solution to the discrete time ARE

```
X=F'*X*F-F'*X*G1*((G2+G1'*X*G1)^-1)*G1'*X*F+H
```

`F` is assumed invertible and `G = G1*inv(G2)*G1'`.

One assumes `(F,G1)` stabilizable and `(C,F)` detectable with `C'*C` full rank factorization of `H`. Use preferably `ric_desc`.

C, D are symmetric .It is assumed that the matrices A, C and D are such that the corresponding matrix pencil has N eigenvalues with moduli less than one.

Error bound on the solution and a condition estimate are also provided. It is assumed that the matrices A, C and D are such that the corresponding Hamiltonian matrix has N eigenvalues with negative real parts.

Examples

```
//Standard formulas to compute Riccati solutions
A=rand(3,3);B=rand(3,2);C=rand(3,3);C=C*C';R=rand(2,2);R=R*R'+eye();
B=B*inv(R)*B';
X=ricc(A,B,C,'cont');
norm(A'*X+X*A-X*B*X+C,1)
H=[A -B;-C -A'];
[T,d]=schur(eye(H),H,'cont');T=T(:,1:d);
X1=T(4:6,:)/T(1:3,:);
norm(X1-X,1)
[T,d]=schur(H,'cont');T=T(:,1:d);
X2=T(4:6,:)/T(1:3,:);
norm(X2-X,1)
//       Discrete time case
F=A;B=rand(3,2);G1=B;G2=R;G=G1/G2*G1';H=C;
X=ricc(F,G,H,'disc');
norm(F'*X*F-(F'*X*G1/(G2+G1'*X*G1))*(G1'*X*F)+H-X)
H1=[eye(3,3) G;zeros(3,3) F'];
H2=[F zeros(3,3);-H eye(3,3)];
[T,d]=schur(H2,H1,'disc');T=T(:,1:d);X1=T(4:6,:)/T(1:3,:);
norm(X1-X,1)
Fi=inv(F);
Hami=[Fi Fi*G;H*Fi F'+H*Fi*G];
[T,d]=schur(Hami,'d');T=T(:,1:d);
Fit=inv(F');
Ham=[F+G*Fit*H -G*Fit;-Fit*H Fit];
[T,d]=schur(Ham,'d');T=T(:,1:d);X2=T(4:6,:)/T(1:3,:);
norm(X2-X,1)
```

See Also

riccati , ric_desc , schur

Authors

P. Petkov

Used Functions

See SCIDIR/routines/control/riccpack>

Name

riccati — Riccati equation

```
X=riccati(A,B,C,dom,[typ])
[X1,X2]=riccati(A,B,C,dom,[typ])
```

Parameters

A,B,C
: real matrices nxn, `B` and `C` symetric.

dom
: `'c'` or `'d'` for the time domain (continuous or discrete)

typ
: string : `'eigen'` for block diagonalization or `schur'` for Schur method.

X1,X2,X
: square real matrices (X2 invertible), X symmetric

Description

`X=riccati(A,B,C,dom,[typ])` solves the Riccati equation:

```
A'*X+X*A-X*B*X+C=0
```

in continuous time case, or:

```
A'*X*A-(A'*X*B1/(B2+B1'*X*B1))*(B1'*X*A)+C-X
```

with `B=B1/B2*B1'` in the discrete time case. If called with two output arguments, `riccati` returns `X1,X2` such that `X=X1/X2`.

See Also

ricc , ric_desc

Name

routh_t — Routh's table

```
r=routh_t(h [,k]).
```

Parameters

h
: square rational matrix

Description

`r=routh_t(h,k)` computes Routh's table of denominator of the system described by transfer matrix SISO `h` with the feedback by the gain `k`.

If `k=poly(0,'k')` we will have a polynomial matrix with dummy variable `k`, formal expression of the Routh table.

Name

rowinout — inner-outer factorization

```
[Inn,X,Gbar]=rowinout(G)
```

Parameters

G
: linear system (`syslin` list) `[A,B,C,D]`

Inn
: inner factor (`syslin` list)

Gbar
: outer factor (`syslin` list)

X
: row-compressor of `G` (`syslin` list)

Description

Inner-outer factorization (and row compression) of (`l`x`p`) `G =[A,B,C,D]` with `l>=p`.

`G` is assumed to be tall (`l>=p`) without zero on the imaginary axis and with a `D` matrix which is full column rank.

`G` must also be stable for having `Gbar` stable.

`G` admits the following inner-outer factorization:

```
        G = [ Inn ] | Gbar |
                    |  0   |
```

where `Inn` is square and inner (all pass and stable) and `Gbar` square and outer i.e: Gbar is square bi-proper and bi-stable (Gbar inverse is also proper and stable);

Note that:

```
        [ Gbar ]
  X*G = [  -   ]
        [  0   ]
```

is a row compression of `G` where `X` = `Inn` inverse is all-pass i.e:

```
 T
X (-s) X(s) = Identity
```

(for the continous time case).

See Also

syslin , colinout

Name

rowregul — removing poles and zeros at infinity

```
[Stmp,Ws]=rowregul(Sl,alfa,beta)
```

Parameters

Sl,Stmp
: `syslin` lists

alfa,beta
: real numbers (new pole and zero positions)

Description

computes a postfilter `Ws` such that `Stmp=Ws*Sl` is proper and with full rank `D` matrix.

Poles at infinity of `Sl` are moved to `alfa`;

Zeros at infinity of `Sl` are moved to `beta`;

`Sl` is a assumed to be a right invertible linear system (`syslin` list) in state-space representation with possibly a polynomial `D` matrix.

This function is the dual of colregul (see function code).

Examples

```
s=%s;
w=[1/s,0;s/(s^3+2),2/s];
Sl=tf2ss(w);
[Stmp,Ws]=rowregul(Sl,-1,-2);
Stmp('D')     // D matrix of Stmp
clean(ss2tf(Stmp))
```

See Also

invsyslin , colregul

Authors

F. D. , R. N. ;

Name

rtitr — discrete time response (transfer matrix)

```
[y]=rtitr(Num,Den,u [,up,yp])
```

Parameters

Num,Den
: polynomial matrices (resp. dimensions : `nxm` and `nxn`)

u
: real matrix (dimension `mx(t+1)`

up,yp
: real matrices (`up` dimension `mx(maxi(degree(Den)))` (default values=`0`) , `yp` dimension `nx (maxi(degree(Den)))`)

y
: real matrix

Description

`y=rtitr(Num,Den,u [,up,yp])` returns the time response of the discrete time linear system with transfer matrix `Den^-1 Num` for the input `u`, i.e `y` and `u` are such that `Den y = Num u` at t=0,1,...

If `d1=maxi(degree(Den))`, and `d2=maxi(degree(Num))` the polynomial matrices `Den(z)` and `Num(z)` may be written respectively as:

```
D(z)= D_0  + D_1  z + ... + D_d1   z^d1
N(z)= N_0  + N_1  z + ... + N_d2   z^d2
```

and `Den y = Num u` is interpreted as the recursion:

```
D(0)y(t)+D(1)y(t+1)+...+ D(d1)y(t+d1)= N(0) u(t) +....+ N(d2) u(t+d2)
```

It is assumed that `D(d1)` is non singular.

The columns of u are the inputs of the system at t=0,1,...,T:

```
u=[u(0) , u(1),...,u(T)]
```

The outputs at `t=0,1,...,T+d1-d2` are the columns of the matrix `y`:

```
 y=[y(0), y(1),  .... y(T+d1-d2)]
```

up and yp define the initial conditions for t < 0 i.e

```
 up=[u(-d1), ..., u(-1)  ]
 yp=[y(-d1), ...  y(-1)  ]
```

Depending on the relative values of d1 and d2, some of the leftmost components of up, yp are ignored. The default values of up and yp are zero: up = 0*ones(m,d1), yp=0*ones(n,d1)

Examples

```
z=poly(0,'z');
Num=1+z;Den=1+z;u=[1,2,3,4,5];
rtitr(Num,Den,u)-u
//Other examples
//siso
//causal
n1=1;d1=poly([1 1],'z','coeff');       // y(j)=-y(j-1)+u(j-1)
r1=[0 1 0 1 0 1 0 1 0 1 0];
r=rtitr(n1,d1,ones(1,10));norm(r1-r,1)
//hot restart
r=rtitr(n1,d1,ones(1,9),1,0);norm(r1(2:11)-r)
//non causal
n2=poly([1 1 1],'z','coeff');d2=d1;    // y(j)=-y(j-1)+u(j-1)+u(j)+u(j+1)
r2=[2 1 2 1 2 1 2 1 2];
r=rtitr(n2,d2,ones(1,10));norm(r-r2,1)
//hot restart
r=rtitr(n2,d2,ones(1,9),1,2);norm(r2(2:9)-r,1)
//
//MIMO example
//causal
d1=d1*diag([1 0.5]);n1=[1 3 1;2 4 1];r1=[5;14]*r1;
r=rtitr(n1,d1,ones(3,10));norm(r1-r,1)
//
r=rtitr(n1,d1,ones(3,9),[1;1;1],[0;0]);
norm(r1(:,2:11)-r,1)
//polynomial n1  (same ex.)
n1(1,1)=poly(1,'z','c');r=rtitr(n1,d1,ones(3,10));norm(r1-r,1)
//
r=rtitr(n1,d1,ones(3,9),[1;1;1],[0;0]);
norm(r1(:,2:11)-r,1)
//non causal
d2=d1;n2=n2*n1;r2=[5;14]*r2;
r=rtitr(n2,d2,ones(3,10));norm(r2-r)
//
r=rtitr(n2,d2,ones(3,9),[1;1;1],[10;28]);
norm(r2(:,2:9)-r,1)
//
//  State-space or transfer
```

```
a = [0.21 , 0.63 , 0.56 , 0.23 , 0.31
     0.76 , 0.85 , 0.66 , 0.23 , 0.93
     0 , 0.69 , 0.73 , 0.22 , 0.21
     0.33 , 0.88 , 0.2 , 0.88 , 0.31
     0.67 , 0.07 , 0.54 , 0.65 , 0.36];
b = [0.29 , 0.5 , 0.92
     0.57 , 0.44 , 0.04
     0.48 , 0.27 , 0.48
     0.33 , 0.63 , 0.26
     0.59 , 0.41 , 0.41];
c = [0.28 , 0.78 , 0.11 , 0.15 , 0.84
     0.13 , 0.21 , 0.69 , 0.7 , 0.41];
d = [0.41 , 0.11 , 0.56
     0.88 , 0.2 , 0.59];
s=syslin('d',a,b,c,d);
h=ss2tf(s);num=h('num');den=h('den');den=den(1,1)*eye(2,2);
u=1;u(3,10)=0;r3=flts(u,s);
r=rtitr(num,den,u);norm(r3-r,1)
```

See Also

ltitr , exp , flts

Name

sensi — sensitivity functions

```
[Se,Re,Te]=sensi(G,K)
[Si,Ri,Ti]=sensi(G,K,flag)
```

Parameters

G
: standard plant (`syslin` list)

K
: compensator (`syslin` list)

flag
: character string `'o'` (default value) or `'i'`

Se
: output sensitivity function `(I+G*K)^-1`

Re
: `K*Se`

Te
: `G*K*Se` (output complementary sensitivity function)

Description

`sensi` computes sensitivity functions. If `G` and `K` are given in state-space form, the systems returned are generically minimal. Calculation is made by `lft`, e.g., `Se` can be given by the commands `P = augment(G,'S')`, `Se=lft(P,K)`. If `flag` = `'i'`, `[Si,Ri,Ti]=sensi(G,K,'i')` returns the input sensitivity functions.

```
[Se;Re;Te]= [inv(eye()+G*K);K*inv(eye()+G*K);G*K*inv(eye()+G*K)];
[Si;Ri;Ti]= [inv(eye()+K*G);G*inv(eye()+K*G);K*G*inv(eye()+K*G)];
```

Examples

```
G=ssrand(1,1,3);K=ssrand(1,1,3);
[Se,Re,Te]=sensi(G,K);
Se1=inv(eye()+G*K);  //Other way to compute
ss2tf(Se)    //Se seen in transfer form
ss2tf(Se1)
ss2tf(Te)
ss2tf(G*K*Se1)
[Si,Ri,Ti]=sensi(G,K,'i');
w1=[ss2tf(Si);ss2tf(Ri);ss2tf(Ti)]
w2=[ss2tf(inv(eye()+K*G));ss2tf(G*inv(eye()+K*G));ss2tf(K*G*inv(eye()+K*G))];
clean(w1-w2)
```

See Also

augment , lft , h_cl

Name

sgrid — s-plane grid lines.

```
sgrid()
sgrid('new')
sgrid(zeta,wn [,color])
```

Description

Used in conjonction with evans, plots lines of constant damping ratio (`zeta`) and natural frequency (`wn`).

sgrid()
: add a grid over an existing continuous s-plane root with default values for `zeta` and `wn`.

sgrid('new')
: clears the graphic screen and then plots a default s-plane grid

sgrid(zeta,wn [,color])
: same as `sgrid()` but uses the provided damping ratio and natural frequency.

Examples

```
H=syslin('c',352*poly(-5,'s')/poly([0,0,2000,200,25,1],'s','c'));
evans(H,100)
sgrid()
sgrid(0.6,2,7)
```

See Also

evans

Name

show_margins — display gain and phase margin and associated crossover frequencies

```
show_margins(h)
show_margins(h,'bode')
show_margins(h,'nyquist')
```

Parameters

h
: a SISO linear system (see :syslin).

Description

Given a SISO linear system in continuous or discrete time, `show_margins` display gain and phase margin and associated crossover frequencies on a bode (the defaut) or nyquist representation of the frequency response of the system.

Examples

```
//continuous case
h=syslin('c',0.02909+0.11827*%s+0.12823*%s^2+0.35659*%s^3+0.256*%s^4+0.1*%s^5,.
                0.0409+0.1827*%s+1.28225*%s^2+3.1909*%s^3+2.56*%s^4+%s^5);
show_margins(h)
show_margins(h,'nyquist')

//discrete case
h = syslin(0.1,0.01547+0.01599*%z ,%z^2-1.81*%z+0.9048)
show_margins(h)
show_margins(h,'nyquist')
```

See Also

p_margin, g_margin, bode, nyquist

Authors

Serge Steer, INRIA

Name

sident — discrete-time state-space realization and Kalman gain

```
[(A,C)(,B(,D))(,K,Q,Ry,S)(,rcnd)] = sident(meth,job,s,n,l,R(,tol,t,Ai,
Ci,printw))
```

Parameters

meth
: integer option to determine the method to use:

  =
  : 1 : MOESP method with past inputs and outputs;

  =
  : 2 : N4SID method;

  =
  : 3 : combined method: A and C via MOESP, B and D via N4SID.

job
: integer option to determine the calculation to be performed:

  =
  : 1 : compute all system matrices, A, B, C, D;

  =
  : 2 : compute the matrices A and C only;

  =
  : 3 : compute the matrix B only;

  =
  : 4 : compute the matrices B and D only.

s
: the number of block rows in the processed input and output block Hankel matrices. s > 0.

n
: integer, the order of the system

l
: integer, the number of the system outputs

R
: the 2*(m+l)*s-by-2*(m+l)*s part of R contains the processed upper triangular factor R from the QR factorization of the concatenated block-Hankel matrices, and further details needed for computing system matrices.

tol
: (optional) tolerance used for estimating the rank of matrices. If tol > 0, then the given value of tol is used as a lower bound for the reciprocal condition number; an m-by-n matrix whose estimated condition number is less than 1/tol is considered to be of full rank. Default: m*n*epsilon_machine where epsilon_machine is the relative machine precision.

t
: (optional) the total number of samples used for calculating the covariance matrices. Either t = 0, or t >= 2*(m+l)*s. This parameter is not needed if the covariance matrices and/or the Kalman predictor gain matrix are not desired. If t = 0, then K, Q, Ry, and S are not computed. Default: t = 0.

Ai
: real matrix

Ci
: real matrix

printw
: (optional) switch for printing the warning messages.

  =
  : 1: print warning messages;

  =
  : 0: do not print warning messages.

  Default: printw = 0.

A
: real matrix

C
: real matrix

B
: real matrix

D
: real matrix

K
: real matrix, kalman gain

Q
: (optional) the n-by-n positive semidefinite state covariance matrix used as state weighting matrix when computing the Kalman gain.

RY
: (optional) the l-by-l positive (semi)definite output covariance matrix used as output weighting matrix when computing the Kalman gain.

S
: (optional) the n-by-l state-output cross-covariance matrix used as cross-weighting matrix when computing the Kalman gain.

rcnd
: (optional) vector of length lr, containing estimates of the reciprocal condition numbers of the matrices involved in rank decisions, least squares, or Riccati equation solutions, where lr = 4, if Kalman gain matrix K is not required, and lr = 12, if Kalman gain matrix K is required.

Description

SIDENT function for computing a discrete-time state-space realization (A,B,C,D) and Kalman gain K using SLICOT routine IB01BD.

```
              [A,C,B,D] = sident(meth,1,s,n,l,R)
[A,C,B,D,K,Q,Ry,S,rcnd] = sident(meth,1,s,n,l,R,tol,t)
                  [A,C] = sident(meth,2,s,n,l,R)
                      B = sident(meth,3,s,n,l,R,tol,0,Ai,Ci)
```

```
        [B,K,Q,Ry,S,rcnd] = sident(meth,3,s,n,l,R,tol,t,Ai,Ci)
                    [B,D] = sident(meth,4,s,n,l,R,tol,0,Ai,Ci)
      [B,D,K,Q,Ry,S,rcnd] = sident(meth,4,s,n,l,R,tol,t,Ai,Ci)
```

SIDENT computes a state-space realization (A,B,C,D) and the Kalman predictor gain K of a discrete-time system, given the system order and the relevant part of the R factor of the concatenated block-Hankel matrices, using subspace identification techniques (MOESP, N4SID, or their combination).

The model structure is :

```
        x(k+1) = Ax(k) + Bu(k) + Ke(k),   k >= 1,
        y(k)   = Cx(k) + Du(k) + e(k),
```

where x(k) is the n-dimensional state vector (at time k),

u(k) is the m-dimensional input vector,

y(k) is the l-dimensional output vector,

e(k) is the l-dimensional disturbance vector,

and A, B, C, D, and K are real matrices of appropriate dimensions.

Comments

1. The n-by-n system state matrix A, and the p-by-n system output matrix C are computed for job <= 2.

2. The n-by-m system input matrix B is computed for job <> 2.

3. The l-by-m system matrix D is computed for job = 1 or 4.

4. The n-by-l Kalman predictor gain matrix K and the covariance matrices Q, Ry, and S are computed for t > 0.

Examples

```
//generate data from a given linear system
A = [ 0.5, 0.1,-0.1, 0.2;
      0.1, 0,  -0.1,-0.1;
     -0.4,-0.6,-0.7,-0.1;
      0.8, 0,  -0.6,-0.6];
B = [0.8;0.1;1;-1];
C = [1 2 -1 0];
SYS=syslin(0.1,A,B,C);
nsmp=100;
U=prbs_a(nsmp,nsmp/5);
Y=(flts(U,SYS)+0.3*rand(1,nsmp,'normal'));

S = 15;
N = 3;
METH=1;
```

```
[R,N1] = findR(S,Y',U',METH);
[A,C,B,D,K] = sident(METH,1,S,N,1,R);
SYS1=syslin(1,A,B,C,D);
SYS1.X0 = inistate(SYS1,Y',U');

Y1=flts(U,SYS1);
xbasc();plot2d((1:nsmp)',[Y',Y1'])



METH = 2;
[R,N1,SVAL] = findR(S,Y',U',METH);
tol = 0;
t = size(U',1)-2*S+1;

[A,C,B,D,K] = sident(METH,1,S,N,1,R,tol,t)
SYS1=syslin(1,A,B,C,D)
SYS1.X0 = inistate(SYS1,Y',U');

Y1=flts(U,SYS1);
xbasc();plot2d((1:nsmp)',[Y',Y1'])
```

See Also

findBD , sorder

Authors

V. Sima, Research Institute for Informatics, Bucharest, Oct. 1999. Revisions: May 2000, July 2000.

Name

sm2des — system matrix to descriptor

```
[Des]=sm2des(Sm);
```

Parameters

Sm
: polynomial matrix (pencil system matrix)

Des
: descriptor system (`list('des',A,B,C,D,E)`)

Description

Utility function: converts the system matrix:

```
Sm = [-sE + A   B;
      [    C     D]
```

to descriptor system `Des=list('des',A,B,C,D,E)`).

See Also

ss2des , sm2ss

Name

sm2ss — system matrix to state-space

```
[Sl]=sm2ss(Sm);
```

Parameters

Sm
: polynomial matrix (pencil system matrix)

Sl
: linear system (`syslin` list)

Description

Utility function: converts the system matrix:

```
Sm = [-sI + A   B;
     [    C     D]
```

to linear system in state-space representation (`syslin`) list.

See Also

ss2des

Name

sorder — computing the order of a discrete-time system

```
[Ro(,n,sval,rcnd)] = sorder(meth,alg,jobd,batch,conct,s,Y(,U,tol,
printw,ldwork,Ri))
```

Parameters

meth
: integer option to determine the method to use:

  =
  : 1 : MOESP method with past inputs and outputs;

  =
  : 2 : N4SID method.

alg
: integer option to determine the algorithm for computing the triangular factor of the concatenated block-Hankel matrices built from the input-output data:

  =
  : 1 : Cholesky algorithm on the correlation matrix;

  =
  : 2 : fast QR algorithm;

  =
  : 3 : standard QR algorithm.

jobd
: integer option to specify if the matrices B and D should later be computed using the MOESP approach:

  =
  : 1 : the matrices B and D should later be computed using the MOESP approach;

  =
  : 2 : the matrices B and D should not be computed using the MOESP approach.

    This parameter is not relevant for meth = 2.

batch
: integer option to specify whether or not sequential data processing is to be used, and, for sequential processing, whether or not the current data block is the first block, an intermediate block, or the last block, as follows:

  =
  : 1 : the first block in sequential data processing;

  =
  : 2 : an intermediate block in sequential data processing;

  =
  : 3 : the last block in sequential data processing;

  =
  : 4 : one block only (non-sequential data processing).

conct
: integer option to specify whether or not the successive data blocks in sequential data processing belong to a single experiment, as follows:

  =
  : 1 : the current data block is a continuation of the previous data block and/or it will be continued by the next data block;

  =
  : 2 : there is no connection between the current data block and the previous and/or the next ones.

    This parameter is not used if batch = 4.

s
: the number of block rows in the input and output block Hankel matrices to be processed. s > 0

Y
: the t-by-l output-data sequence matrix. Column j of Y contains the t values of the j-th output component for consecutive time increments.

U
: (optional) the t-by-m input-data sequence matrix. Column j of U contains the t values of the j-th input component for consecutive time increments. Default: U = [].

tol
: (optional) vector of length 2 containing tolerances: tol(1) - tolerance used for estimating the rank of matrices. If tol(1) > 0, then the given value of tol(1) is used as a lower bound for the reciprocal condition number; an m-by-n matrix whose estimated condition number is less than 1/tol(1) is considered to be of full rank. If tol(1) <= 0, then a default value m*n*epsilon_machine is used, where epsilon_machine is the relative machine precision.

  tol(2) - tolerance used for determining an estimate of the system order. If tol(2) >= 0, the estimate is indicated by the index of the last singular value greater than or equal to tol(2). (Singular values less than tol(2) are considered as zero.) When tol(2) = 0, an internally computed default value, tol(2) = s*epsilon_machine*sval(1), is used, where sval(1) is the maximal singular value, and epsilon_machine the relative machine precision. When tol(2) < 0, the estimate is indicated by the index of the singular value that has the largest logarithmic gap to its successor.

  Default: tol(1:2) = [0,-1].

printw
: :(optional) switch for printing the warning messages.

  =
  : 1: print warning messages;

  =
  : 0: do not print warning messages.

  Default: printw = 0.

ldwork
: (optional) the workspace size. Default : computed by the formulas

```
nr = 2*( m + l )*s
LDWORK = ( t - 2*s + 3 + 64 )*nr
if ( CSIZE > MAX( nr*nr + t*( m + l ) + 16, 2*nr ) ) then
   LDWORK = MIN( LDWORK, CSIZE - nr*nr - t*( m + l ) - 16 )
```

```
else
   LDWORK = MIN( LDWORK, MAX( 2*nr, CSIZE/2 ) )
end if
```

LDWORK = MAX(minimum workspace size needed, LDWORK) where CSIZE is the cache size in double precision words.

If LDWORK is specified less than the minimum workspace size needed, that minimum value is used instead.

Ri

(optional) if batch = 2 or 3, the 2*(m+l)*s-by-2*(m+l)*s (upper triangular, if alg <> 2) part of R must contain the (upper triangular) matrix R computed at the previous call of this mexfile in sequential data processing. If conct = 1, R has an additional column, also set at the previous call.

If alg = 2, R has m+l+1 additional columns, set at the previous call.

This parameter is not used for batch = 1 or batch = 4.

Ro

if batch = 3 or 4, the 2*(m+l)*s-by-2*(m+l)*s part of R contains the processed upper triangular factor R from the QR factorization of the concatenated block-Hankel matrices, and further details needed for computing system matrices. If batch = 1 or 2, then R contains intermediate results needed at the next call of this mexfile. If batch = 1 or 2 and conct = 1, R has an additional column, also set before return. If batch = 1 or 2 and alg = 2, R has m+l+1 additional columns, set before return.

n

the order of the system.

sval

(optional) the singular values used for estimating the order of the system.

rcnd

(optional) if meth = 2, vector of length 2 containing the reciprocal condition numbers of the matrices involved in rank decisions or least squares solutions.

Description

sorder - function for computing the order of a discrete-time system using SLICOT routine IB01AD.

For one block (data sequences Y, U): [R,n,sval,rcnd] = sorder(meth,alg,jobd,4,conct,s,Y,U);

For f blocks (data sequences Yj, Uj, j = 1 : f):

```
R = sorder(meth,alg,jobd,1,conct,s,Y1,U1);
for j = 2 : f - 1
   R = sorder(meth,alg,jobd,2,conct,s,Yj,Uj,tol,printw,ldwork,R)
end
[R,n,sval,rcnd] = sorder(meth,alg,jobd,3,conct,s,Yf,Uf,tol);
```

sorder preprocesses the input-output data for estimating the matrices of a linear time-invariant dynamical system, using Cholesky or (fast) QR factorization and subspace identification techniques (MOESP and N4SID), and then estimates the order of a discrete-time realization.

The model structure is :

```
x(k+1) = Ax(k) + Bu(k) + w(k),   k >= 1,
y(k)   = Cx(k) + Du(k) + e(k),
```

where x(k) is the n-dimensional state vector (at time k),

u(k) is the m-dimensional input vector,

y(k) is the l-dimensional output vector,

w(k) is the n-dimensional state disturbance vector,

e(k) is the l-dimensional output disturbance vector,

and A, B, C, and D are real matrices of appropriate dimensions.

Comments

1. The Cholesy or fast QR algorithms can be much faster (for large data blocks) than QR algorithm, but they cannot be used if the correlation matrix, H'*H, is not positive definite. In such a case, the code automatically switches to the QR algorithm, if sufficient workspace is provided and batch = 4.

2. If ldwork is specified, but it is less than the minimum workspace size needed, that minimum value is used instead.

See Also

findBD , sident

Authors

V. Sima, Research Institute for Informatics, Bucharest, Oct. 1999.; ; Revisions:; V. Sima, May 2000, July 2000.

Name

specfact — spectral factor

```
[W0,L]=specfact(A,B,C,D)
```

Description

Given a spectral density matrix `phi(s)`:

```
                -1                      -1
    R + C*(s*I-A) * B  +  B'*(-s*I-A') * C'  with R=D+D' > 0
```

`specfact` computes `W0` and `L` such that `W(s)=W0+L*(s*I-A)^-1*B` is a spectral factor of of `PHI(s)`, i.e.

`phi(s)=W'(-s)*W(s)`

Examples

```
A=diag([-1,-2]);B=[1;1];C=[1,1];D=1;s=poly(0,'s');
W1=syslin('c',A,B,C,D);
phi=gtild(W1,'c')+W1;
phis=clean(ss2tf(phi))
clean(phis-horner(phis,-s)');   //check this is 0...
[A,B,C,D]=abcd(W1);
[W0,L]=specfact(A,B,C,D);
W=syslin('c',A,B,L,W0)
Ws=ss2tf(W);
horner(Ws,-s)*Ws
```

See Also

gtild , sfact , fspecg

Authors

F. D.

Name

ss2des — (polynomial) state-space to descriptor form

```
S=ss2des(Sl)
S=ss2des(Sl,flag)
```

Parameters

Sl
: `syslin` list: proper or improper linear system.

flag
: character string `"withD"`

S
: list

Description

Given the linear system in state-space representation `Sl` (`syslin` list), with a `D` matrix which is either polynomial or constant, but not zero `ss2des` returns a descriptor system as `list('des',A,B,C,0,E)` such that:

```
Sl=C*(s*E-A)^(-1)*B
```

If the flag `"withD"` is given, `S=list('des',A,B,C,D,E)` with a `D` matrix of maximal rank.

Examples

```
s=poly(0,'s');
G=[1/(s+1),s;1+s^2,3*s^3];Sl=tf2ss(G);
S=ss2des(Sl)
S1=ss2des(Sl,"withD")
Des=des2ss(S);Des(5)=clean(Des(5))
Des1=des2ss(S1)
```

See Also

pol2des , tf2des , des2ss

Authors

F. D.; ;

Name

ss2ss — state-space to state-space conversion, feedback, injection

```
[Sl1,right,left]=ss2ss(Sl,T, [F, [G , [flag]]])
```

Parameters

Sl
: linear system (`syslin` list) in state-space form

T
: square (non-singular) matrix

Sl1, right, left
: linear systems (syslin lists) in state-space form

F
: real matrix (state feedback gain)

G
: real matrix (output injection gain)

Description

Returns the linear system `Sl1=[A1,B1,C1,D1]` where `A1=inv(T)*A*T, B1=inv(T)*B, C1=C*T, D1=D`.

Optional parameters `F` and `G` are state feedback and output injection respectively.

For example, `Sl1=ss2ss(Sl,T,F)` returns `Sl1` with:

and `right` is a non singular linear system such that `Sl1=Sl*right`.

`Sl1*inv(right)` is a factorization of `Sl`.

`Sl1=ss2ss(Sl,T,0*F,G)` returns `Sl1` with:

and `left` is a non singular linear system such that `Sl1=left*Sl` (`right=Id` if `F=0`).

When both `F` and `G` are given, `Sl1=left*Sl*right`.

- When `flag` is used and `flag=1` an output injection as follows is used and then a feedback is performed, `F` must be of size `(m+p,n)`

  (x is in `R^n` , y in `R^p`, u in `R^m`).

  `right` and `left` have the following property:

  ```
  Sl1 =  left*sysdiag(sys,eye(p,p))*right
  ```

- When `flag` is used and `flag=2` a feedback (`F` must be of size `(m,n)`) is performed and then the above output injection is applied. `right` and `left` have the following property:

```
 Sl1 = left*sysdiag(sys*right,eye(p,p)))
```

Examples

```
Sl=ssrand(2,2,5); trzeros(Sl)       // zeros are invariant:
Sl1=ss2ss(Sl,rand(5,5),rand(2,5),rand(5,2));
trzeros(Sl1), trzeros(rand(2,2)*Sl1*rand(2,2))
// output injection [ A + GC, (B+GD,-G)]
//                  [   C   , (D   , 0)]
p=1,m=2,n=2; sys=ssrand(p,m,n);

// feedback (m,n)  first and then output injection.

F1=rand(m,n);
G=rand(n,p);
[sys1,right,left]=ss2ss(sys,rand(n,n),F1,G,2);

// Sl1 equiv left*sysdiag(sys*right,eye(p,p)))

res=clean(ss2tf(sys1) - ss2tf(left*sysdiag(sys*right,eye(p,p))))

// output injection then feedback (m+p,n)
F2=rand(p,n); F=[F1;F2];
[sys2,right,left]=ss2ss(sys,rand(n,n),F,G,1);

// Sl1 equiv left*sysdiag(sys,eye(p,p))*right

res=clean(ss2tf(sys2)-ss2tf(left*sysdiag(sys,eye(p,p))*right))

// when F2= 0; sys1 and sys2 are the same
F2=0*rand(p,n);F=[F1;F2];
[sys2,right,left]=ss2ss(sys,rand(n,n),F,G,1);

res=clean(ss2tf(sys2)-ss2tf(sys1))
```

See Also

projsl , feedback

Name

ss2tf — conversion from state-space to transfer function

```
[h]=ss2tf(sl)
[Ds,NUM,chi]=ss2tf(sl)

[h]=ss2tf(sl,"b")
[Ds,NUM,chi]=ss2tf(sl,"b")


[h]=ss2tf(sl,rmax)
[Ds,NUM,chi]=ss2tf(sl,rmax)
```

Parameters

sl
: linear system (`syslin` list)

h
: transfer matrix

Description

Called with three outputs `[Ds,NUM,chi]=ss2tf(sl)` returns the numerator polynomial matrix `NUM`, the characteristic polynomial `chi` and the polynomial part `Ds` separately i.e.:

```
h=NUM/chi + Ds
```

Method:

One uses the characteristic polynomial and `det(A+Eij)=det(A)+C(i,j)` where `C` is the adjugate matrix of `A`.

With `rmax` or `"b"` argument uses a block diagonalization of sl.A matrix and applies "Leverrier" algorithm on blocks. If given, `rmax` controls the conditionning (see bdiag).

Examples

```
s=poly(0,'s');
h=[1,1/s;1/(s^2+1),s/(s^2-2)]
sl=tf2ss(h);
h=clean(ss2tf(sl))
[Ds,NUM,chi]=ss2tf(sl)
```

See Also

tf2ss , syslin , nlev , glever

Name

st_ility — stabilizability test

```
[ns, [nc, [,U [,Slo] ]]]=st_ility(Sl [,tol])
```

Parameters

Sl
: `syslin` list (linear system)

ns
: integer (dimension of stabilizable subspace)

nc
: integer (dimension of controllable subspace `nc <= ns`)

U
: basis such that its `ns` (resp. `nc`) first components span the stabilizable (resp. controllable) subspace

Slo
: a linear system (`syslin` list)

tol
: threshold for controllability detection (see contr)

Description

`Slo=( U'*A*U, U'*B, C*U, D, U'*x0 )` (`syslin` list) displays the stabilizable form of `Sl`. Stabilizability means `ns=nx` (dim. of `A` matrix).

```
          [*,*,*]             [*]
U'*A*U = [0,*,*]     U'*B = [0]
          [0,0,*]             [0]
```

where `(A11,B1)` (dim(A11)= `nc`) is controllable and `A22` (dim(A22)=`ns-nc`) is stable. "Stable" means real part of eigenvalues negative for a continuous linear system, and magnitude of eigenvalues lower than one for a discrete-time system (as defined by `syslin`).

Examples

```
A=diag([0.9,-2,3]);B=[0;0;1];Sl=syslin('c',A,B,[]);
[ns,nc,U]=st_ility(Sl);
U'*A*U
U'*B
[ns,nc,U]=st_ility(syslin('d',A,B,[]));
U'*A*U
U'*B
```

See Also

dt_ility , contr , stabil , ssrand

Authors

S. Steer INRIA 1988

Name

stabil — stabilization

```
F=stabil(A,B,alfa)
K=stabil(Sys,alfa,beta)
```

Parameters

A
: square real matrix (`nx x nx`)

B
: real matrix (`nx x nu`)

alfa, beta
: real or complex vector (in conjugate pairs) or real number.

F
: real matrix (`nx x nu`)

Sys
: linear system (`syslin` list) (`m` inputs, `p` outputs).

K
: linear system (`p` inputs, `m` outputs)

Description

`F=stabil(A,B,alfa)` returns a gain matrix `F` such that `A+B*F` is stable if pair `(A,B)` is stabilizable. Assignable poles are set to `alfa(1),alfa(2),...`. If `(A,B)` is not stabilizable a warning is given and assignable poles are set to `alfa(1),alfa(2),...`. If `alfa` is a number all eigenvalues are set to this `alfa` (default value is `alfa=-1`).

`K=stabil(Sys,alfa,beta)` returns `K`, a compensator for `Sys` such that `(A,B)`-controllable eigenvalues are set to `alfa` and `(C,A)`-observable eigenvalues are set to `beta`.

All assignable closed loop poles (which are given by the eigenvalues of `Aclosed=h_cl(Sys,K)` are set to `alfa(i)`'s and `beta(j)`'s.

Examples

```
// Gain:
Sys=ssrand(0,2,5,list('st',2,3,3));
A=Sys('A');B=Sys('B');F=stabil(A,B);
spec(A) //2 controllable modes 2 unstable uncontrollable modes
//and one stable uncontrollable mode
spec(A+B*F) //the two controllable modes are set to -1.
// Compensator:
Sys=ssrand(3,2,5,list('st',2,3,3)); //3 outputs, 2 inputs, 5 states
//2 controllables modes, 3 controllable or stabilizable modes.
K=stabil(Sys,-2,-3);  //Compensator for Sys.
spec(Sys('A'))
spec(h_cl(Sys,K))   //K Stabilizes what can be stabilized.
```

See Also

st_ility , contr , ppol

Name

svplot — singular-value sigma-plot

```
[SVM]=svplot(sl,[w])
```

Parameters

sl
: `syslin` list (continuous, discrete or sampled system)

w
: real vector (optional parameter)

Description

computes for the system `sl=(A,B,C,D)` the singular values of its transfer function matrix:

```
              G(jw) = C(jw*I-A)B^-1+D
or
              G(exp(jw)) = C(exp(jw)*I-A)B^-1+D
or
              G(exp(jwT)) = C(exp(jw*T)*I-A)B^-1+D
```

evaluated over the frequency range specified by `w`. (T is the sampling period, `T=sl('dt')` for sampled systems).

`sl` is a `syslin` list representing the system `[A,B,C,D]` in state-space form. `sl` can be continous or discrete time or sampled system.

The `i`-th column of the output matrix `SVM` contains the singular values of `G` for the `i`-th frequency value `w(i)`.

```
SVM = svplot(sl)
```

is equivalent to

```
SVM = svplot(sl,logspace(-3,3))  (continuous)
```

```
SVM = svplot(sl,logspace(-3,%pi)) (discrete)
```

Examples

```
x=logspace(-3,3);
y=svplot(ssrand(2,2,4),x);
xbasc();plot2d1("oln",x',20*log(y')/log(10));
xgrid(12)
xtitle("Singular values plot","(Rd/sec)", "Db");
```

Authors

F.D; ;

Name

sysfact — system factorization

```
[S,Series]=sysfact(Sys,Gain,flag)
```

Parameters

Sys
: `syslin` list containing the matrices `[A,B,C,D]`.

Gain
: real matrix

flag
: string `'post'` or `'pre'`

S
: `syslin` list

Series
: `syslin` list

Description

If `flag` equals `'post'`, `sysfact` returns in `S` the linear system with ABCD matrices `(A +B*Gain, B , Gain, I)`, and `Series`, a minimal realization of the series system `Sys*S`. If `flag` equals `'pre'`, `sysfact` returns the linear system `(A+Gain*C, Gain , C, I)` and `Series`, a minimal realization of the series system `S*Sys`.

Examples

```
//Kalman filter
Sys=ssrand(3,2,4);Sys('D')=rand(3,2);
S=sysfact(Sys,lqr(Sys),'post');
ww=minss(Sys*S);
ss2tf(gtild(ww)*ww),Sys('D')'*Sys('D')
//Kernel
Sys=ssrand(2,3,4);
[X,d,F,U,k,Z]=abinv(Sys);
ss2tf(Sys*Z)
ss2tf(Sys*sysfact(Sys,F,'post')*U)
```

See Also

lqr , lqe

Authors

F.D.

Name

syssize — size of state-space system

```
[r,nx]=syssize(Sl)
```

Parameters

Sl
: linear system (`syslin` list) in state-space

r
: 1 x 2 real vector

nx
: integer

Description

returns in `r` the vector [number of outputs, number of inputs] of the linear system `Sl`. `nx` is the number of states of `Sl`.

See Also

size

Name

tf2des — transfer function to descriptor

```
S=tf2des(G)
S=tf2des(G,flag)
```

Parameters

G
: linear system (`syslin` list) with possibly polynomial `D` matrix

flag
: character string `"withD"`

S
: list

Description

Transfer function to descriptor form: `S=list('d',A,B,C,D,E)`

```
E*xdot = A*x+B*u
y = C*x + D*u
```

Note that `D=0` if the optional parameter flag=`"withD"` is not given. Otherwise a maximal rank `D` matrix is returned in the fifth entry of the list `S`

Examples

```
s=poly(0,'s');
G=[1/(s-1),s;1,2/s^3];
S1=tf2des(G);des2tf(S1)
S2=tf2des(G,"withD");des2tf(S2)
```

See Also

pol2des , tf2ss , ss2des , des2tf

Name

tf2ss — transfer to state-space

```
sl=tf2ss(h [,tol])
```

Parameters

h
: rational matrix

tol
: may be the constant rtol or the 2 vector `[rtol atol]`

  rtol
  : :tolerance used when evaluating observability.

  atol
  : :absolute tolerance used when evaluating observability.

sl
: linear system (`syslin` list `sl=[A,B,C,D(s)]`)

Description

transfer to state-space conversion:

`h=C*(s*eye()-A)^-1*B+D(s)`

Examples

```
s=poly(0,'s');
H=[2/s,(s+1)/(s^2-5)];
Sys=tf2ss(H)
clean(ss2tf(Sys))
```

See Also

ss2tf , tf2des , des2tf

Name

time_id — SISO least square identification

```
[H [,err]]=time_id(n,u,y)
```

Parameters

n
: order of transfer

u
: one of the following

  u1
  : a vector of inputs to the system

  "impuls"
  : if y is an impulse response

  "step"
  : if y is a step response.

y
: vector of response.

H
: rational function with degree n denominator and degree n-1 numerator if y(1)==0 or rational function with degree n denominator and numerator if y(1)<>0.

err
: `||y - impuls(H,npt)||^2`, where `impuls(H,npt)` are the `npt` first coefficients of impulse response of `H`

Description

Identification of discrete time response. If `y` is strictly proper (`y(1)=0`) then `time_id` computes the least square solution of the linear equation: `Den*y-Num*u=0` with the constraint `coeff(Den,n):=1`. if `y(1)~=0` then the algorithm first computes the proper part solution and then add y(1) to the solution

Examples

```
z=poly(0,'z');
h=(1-2*z)/(z^2-0.5*z+5)
rep=[0;ldiv(h('num'),h('den'),20)]; //impulse response
H=time_id(2,'impuls',rep)
//  Same example with flts and u
u=zeros(1,20);u(1)=1;
rep=flts(u,tf2ss(h));        //impulse response
H=time_id(2,u,rep)
//  step response
u=ones(1,20);
rep=flts(u,tf2ss(h));     //step response.
H=time_id(2,'step',rep)
H=time_id(3,u,rep)    //with u as input and too high order required
```

See Also

imrep2ss , arl2 , armax , frep2tf

Authors

Serge Steer INRIA

Name

trzeros — transmission zeros and normal rank

```
[tr]=trzeros(Sl)
[nt,dt,rk]=trzeros(Sl)
```

Parameters

Sl
: linear system (`syslin` list)

nt
: complex vectors

dt
: real vector

rk
: integer (normal rank of Sl)

Description

Called with one output argument, `trzeros(Sl)` returns the transmission zeros of the linear system `Sl`.

`Sl` may have a polynomial (but square) `D` matrix.

Called with 2 output arguments, `trzeros` returns the transmission zeros of the linear system `Sl` as `tr=nt./dt`;

(Note that some components of `dt` may be zeros)

Called with 3 output arguments, `rk` is the normal rank of `Sl`

Transfer matrices are converted to state-space.

If `Sl` is a (square) polynomial matrix `trzeros` returns the roots of its determinant.

For usual state-space system `trzeros` uses the state-space algorithm of Emami-Naeni and Van Dooren.

If `D` is invertible the transmission zeros are the eigenvalues of the "`A` matrix" of the inverse system : `A - B*inv(D)*C`;

If `C*B` is invertible the transmission zeros are the eigenvalues of `N*A*M` where `M*N` is a full rank factorization of `eye(A)-B*inv(C*B)*C`;

For systems with a polynomial `D` matrix zeros are calculated as the roots of the determinant of the system matrix.

Caution: the computed zeros are not always reliable, in particular in case of repeated zeros.

Examples

```
W1=ssrand(2,2,5);trzeros(W1)    //call trzeros
roots(det(systmat(W1)))         //roots of det(system matrix)
```

```
s=poly(0,'s');W=[1/(s+1);1/(s-2)];W2=(s-3)*W*W';[nt,dt,rk]=trzeros(W2);
St=systmat(tf2ss(W2));[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck(St);
St1=Q*St*Z;rowf=(Qd(1)+Qd(2)+1):(Qd(1)+Qd(2)+Qd(3));
colf=(Zd(1)+Zd(2)+1):(Zd(1)+Zd(2)+Zd(3));
roots(St1(rowf,colf)), nt./dt     //By Kronecker form
```

See Also

gspec , kroneck

Name

ui_observer — unknown input observer

```
[UIobs,J,N]=ui_observer(Sys,reject,C1,D1)
[UIobs,J,N]=ui_observer(Sys,reject,C1,D1,flag,alfa,beta)
```

Parameters

Sys
: `syslin` list containing the matrices `(A,B,C2,D2)`.

reject
: integer vector, indices of inputs of `Sys` which are unknown.

C1
: real matrix

D1
: real matrix. `C1` and `D1` have the same number of rows.

flag
: string `'ge'` or `'st'` (default) or `'pp'`.

alfa
: real or complex vector (loc. of closed loop poles)

beta
: real or complex vector (loc. of closed loop poles)

Description

Unknown input observer.

`Sys: (w,u) --> y` is a `(A,B,C2,D2)` syslin linear system with two inputs w and u, w being the unknown input. The matrices B and D2 of Sys are (implicitly) partitioned as: `B=[B1,B2]` and `D2=[D21,D22]` with `B1=B(:,reject)` and `D21=D2(:,reject)` where reject = indices of unknown inputs. The matrices `C1` and `D1` define `z = C1 x + D1 (w,u)`, the to-be-estimated output.

The matrix `D1` is (implicitly) partitioned as `D1=[D11,D12]` with `D11=D(:,reject)`

The data (Sys, reject,C1, D1) define a 2-input 2-output system:

```
xdot =  A x + B1  w + B2  u
   z = C1 x + D11 w + D12 u
   y = C2 x + D21 w + D22 u
```

An observer `(u,y) --> zhat` is looked for the output z.

`flag='ge'` no stability constraints `flag='st'` stable observer (default) `flag='pp'` observer with pole placement alfa,beta = desired location of closed loop poles (default -1, -2) J=y-output to x-state injection. N=y-output to z-estimated output injection.

UIobs = linear system (u,y) --> zhat such that: The transfer function: (w,u) --> z equals the composed transfer function: [0,I; UIobs Sys] (w,u) -----> (u,y) -----> zhat i.e. transfer function of system {A,B,C1,D1} equals transfer function `UIobs*[0,I; Sys]`

Stability (resp. pole placement) requires detectability (resp. observability) of (A,C2).

Examples

```
A=diag([3,-3,7,4,-4,8]);
B=[eye(3,3);zeros(3,3)];
C=[0,0,1,2,3,4;0,0,0,0,0,1];
D=[1,2,3;0,0,0];
rand('seed',0);w=ss2ss(syslin('c',A,B,C,D),rand(6,6));
[A,B,C,D]=abcd(w);
B=[B,matrix(1:18,6,3)];D=[D,matrix(-(1:6),2,3)];
reject=1:3;
Sys=syslin('c',A,B,C,D);
N1=[-2,-3];C1=-N1*C;D1=-N1*D;
nw=length(reject);nu=size(Sys('B'),2)-nw;
ny=size(Sys('C'),1);nz=size(C1,1);
[UIobs,J,N]=ui_observer(Sys,reject,C1,D1);

W=[zeros(nu,nw),eye(nu,nu);Sys];UIobsW=UIobs*W;
//(w,u) --> z=UIobs*[0,I;Sys](w,u)
clean(ss2tf(UIobsW));
wu_to_z=syslin('c',A,B,C1,D1);clean(ss2tf(wu_to_z));
clean(ss2tf(wu_to_z)-ss2tf(UIobsW),1.d-7)
/////2nd example//////
nx=2;ny=3;nwu=2;Sys=ssrand(ny,nwu,nx);
C1=rand(1,nx);D1=[0,1];
UIobs=ui_observer(Sys,1,C1,D1);
```

See Also

cainv , ddp , abinv

Authors

F.D.

Name

unobs — unobservable subspace

```
[n,[U]]=unobs(A,C,[tol])
```

Parameters

A, C
: real matrices

tol
: tolerance used when evaluating ranks (QR factorizations).

n
: dimension of unobservable subspace.

U
: orthogonal change of basis which puts `(A,B)` in canonical form.

Description

`[n,[U]]=unobs(A,C,[tol])` gives the unobservable form of an `(A,C)` pair. The `n` first columns of `U` make a basis for the unobservable subspace.

The `(2,1)` block (made of last `nx-n` rows and `n` first columns) of `U'*A*U` is zero and and the `n` first columns of `C*U` are zero.

Examples

```
A=diag([1,2,3]);C=[1,0,0];
unobs(A,C)
```

See Also

contr , contrss , canon , cont_mat , spantwo , dt_ility

Name

zeropen — zero pencil

```
[Z,U]=zeropen(Sl)
```

Parameters

Sl
: a linear system (`syslin` list in state-space form `[A,B,C,D]`)

Z
: matrix pencil `Z=s*E-A`

U
: square orthogonal matrix

Description

`Z = sE - F` is the zero pencil of the linear system `Sl` with matrices `[A,B,C,D]`. Utility function.

With `U` row compression of `[B;D]` i.e, `U*[B;D]=[0;*]`; one has:

```
U*[-sI+A |B;   [ Z |0;
      C  |D] = [ * |*]
```

The zeros of `Z` are the zeros of `Sl`.

See Also

systmat , kroneck

Name

zgrid — zgrid plot

```
zgrid()
```

Description

plots z-plane grid lines: lines of constant damping factor (zeta) and natural frequency (Wn) are drawn in within the unit Z-plane circle.

Iso-frequency curves are shown in frequency*step on the interval [0,0.5]. Upper limit corresponds to Shannon frequency `( 1/dt > 2*f )`.

See Also

frep2tf , freson

Compatibility Functions

Name

asciimat — string matrix ascii conversions

```
a=asciimat(txt)
txt=asciimat(a)
```

Parameters

txt
: character string or matrix of strings.

a
: vector or matrix of integer ASCII codes

Description

This function convert Scilab string to ASCII code or a matrix of ASCII code to Scilab string. Output is a matrix having same number of lines than input, what is not the case with `ascii`.

See Also

ascii

Authors

V.C.

Name

firstnonsingleton — Finds first dimension which is not 1

```
[dim]=firstnonsingleton(A[,opt])
```

Parameters

dim
: first dimension of A which is not 1

A
: a variable of any Scilab data type

opt
: a character string giving the type of output we want

  "num"
  : returned value is a numerical value

  "str"
  : returned value is a character string if possible ("r" instead of 1 and "c" instead of 2)

Description

This function is used by `mfile2sci` to emulate Matlab behavior under Scilab, particularly for functions which treat the values along the first non-singleton dimension of A in Matlab while in Scilab they treat all values of A.

Examples

```
A = [1 2 3;4 5 6];
// Scilab max
M = max(A)
// Matlab max emulation
M = max(A,firstnonsingleton(A))
```

Authors

V.C.

Nom

makecell — Creates a cell array.

```
s = makecell(dims,a1,a2,...an)
```

Parameters

dims
: a vector with positive integer elements, the cell array dimension

a1,a2,...,an
: Sequence of Scilab variables, `n` must be equal to `prod(dims)`

s
: resulting cell array

Description

`s = makecell(dims,a1,a2,...an)` creates a cell array of dimensions given by `dims` with the given input arguments. The `ai` are stored along the last dimension first.

Examples

```
makecell([2,3],1,2,3,'x','y','z')
```

See Also

cell

Authors

Farid Belhacen, INRIA

Name

mstr2sci — character string matrix to character matrix conversion

```
a=mstr2sci(txt)
```

Parameters

txt
: character string or string matrix

a
: character vector or matrix

Description

This function converts a Scilab character string to a vector of characters. Result is the Scilab equivalent for a Matlab string.

Caution: `mstr2sci` has not to be used for hand coded functions.

See Also

Matlab-Scilab_character_strings

Authors

V.C.

Name

mtlb_0 — Matlab non-conjugate transposition emulation function

Description

Matlab and Scilab non-conjugate transposition behave differently in some particular cases:

- With character strings operands: The `.'` operator is used to transpose whole character strings in Scilab while Matlab realizes the transposition of each character.

The function `mtlb_0(A)` is used by `mfile2sci` to replace `A.'` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_0` calls:

- If `A` is not a character string matrix `mtlb_0(A)` may be replaced by `A.'`

Caution: `mtlb_0` has not to be used for hand coded functions.

See Also

Matlab-Scilab_character_strings

Authors

V.C.

Name

mtlb_a — Matlab addition emulation function

Description

Matlab and Scilab addition behave differently in some particular cases:

- With character string operands: The + operator is used into Scilab to catenate character strings, while Matlab realizes the sum of the operands ASCII codes.
- With empty matrix: In Scilab, if one of the operands is an empty matrix the result of the addition is the other operand. In Matlab if one of the operand is an empty matrix the result of the addition should be an error or an empty matrix.

The function `mtlb_a(A,B)` is used by `mfile2sci` to replace `A+B` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_a` calls:

- If `A` and `B` are character strings, `mtlb_a(A,B)` may be replaced by `asciimat(A)+asciimat(B)`
- If both `A` and `B` are not empty matrices `mtlb_a(A,B)` may be replaced by `A+B`, else `mtlb_a(A,B)` may be replaced by `[]`.
- If mtlb_mode==%T, then `mtlb_a(A,B)` may be replaced by `A+B` in any case where `A` and `B` are not character string matrices.

Caution: `mtlb_a` has not to be used for hand coded functions.

See Also

mtlb_mode

Authors

V.C.

Name

mtlb_all — Matlab all emulation function

Description

Matlab `all` and Scilab `and` behave differently in some particular cases:

- When used with one input (`all(A)`), Matlab `all` treats the values along the first non-singleton dimension of A as vectors while Scilab `and` treats all A values.
- When used with two inputs (`all(A,dim)`), Matlab tolerates `dim` to be greater than the number of dimensions of A while Scilab returns an error message in this case.

The function `R = mtlb_all(A[,dim])` is used by `mfile2sci` to replace `all(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_all` calls:

- If `A` is a scalar or a vector `R = mtlb_all(A)` may be replaced by `R = and(A)`
- If `A` is a matrix `R = mtlb_all(A)` may be replaced by `R = and(A,1)`
- If `A` is an hypermatrix `R = mtlb_all(A)` may be replaced by `R = and(A,firstnonsingleton(A))` or by `R = and(A,user_defined_value)` if the first non-singleton dimensions of A is known.
- If `dim` is less equal to the number of dimensions of A `R = mtlb_all(A,dim)` may be replaced by `R = and(A,dim)`
- If `dim` is greater than then number of dimensions of A `R = mtlb_all(A,dim)` may be replaced by `R = A<>0` if A is not an empty matrix or by `R = A` if A is an empty matrix.

Caution: `mtlb_all` has not to be used for hand coded functions.

See Also

firstnonsingleton

Authors

V.C.

Name

mtlb_any — Matlab any emulation function

Description

Matlab `any` and Scilab `or` behave differently in some particular cases:

- When used with one input (`any(A)`), Matlab `any` treats the values along the first non-singleton dimension of A as vectors while Scilab `or` treats all A values.
- When used with two inputs (`any(A,dim)`), Matlab tolerates `dim` to be greater than the number of dimensions of A while Scilab returns an error message in this case.

The function `R = mtlb_any(A[,dim])` is used by `mfile2sci` to replace `any(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_any` calls:

- If `A` is a scalar or a vector `R = mtlb_any(A)` may be replaced by `R = or(A)`
- If `A` is a matrix `R = mtlb_any(A)` may be replaced by `R = or(A,1)`
- If `A` is an hypermatrix `R = mtlb_any(A)` may be replaced by `R = or(A,firstnonsingleton(A))` or by `R = or(A,user_defined_value)` if the first non-singleton dimensions of A is known.
- If `dim` is less equal to the number of dimensions of A `R = mtlb_any(A,dim)` may be replaced by `R = or(A,dim)`
- If `dim` is greater than then number of dimensions of A `R = mtlb_any(A,dim)` may be replaced by `R = A<>0` if A is not an empty matrix or by `R = A` if A is an empty matrix.

Caution: `mtlb_any` has not to be used for hand coded functions.

See Also

firstnonsingleton

Authors

V.C.

Name

mtlb_axis — Matlab axis emulation function

```
mtlb_axis(X)
mtlb_axis(st)
mtlb_axis(axeshandle, ...)
[mode,visibility,direction]=mtlb_axis('state')
```

Parameters

X
: a vector of reals ([xmin xmax ymin ymax] or [xmin xmax ymin ymax zmin zmax])

st
: a string ('auto', 'manual', 'tight', 'ij', 'xy', 'equal', 'square', 'vis3d', 'off', 'on')

axeshandle
: handle of the current axes entity

Description

Matlab `axis` have not a Scilab equivalent function.

The function `mtlb_axis(...)` is used by `mfile2sci` to replace `axis(...)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time (axis `fill`, axis `image` and axis `normal` are not implemented). If you want to have a more efficient code it is possible to replace `mtlb_axis` call by `get(axeshandle, prop)` call (prop is a axes property, see axis_properties)

Caution: `mtlb_axis` has not to be used for hand coded functions.

Authors

F.B.

Name

mtlb_beta — Matlab beta emulation function

Description

Matlab and Scilab `beta` behave differently in some particular cases:

- With inputs having different sizes: Matlab `beta` input parameters must have the same size unless one of them is a scalar. In Scilab `beta` input parameters must have the same size even if one of them is a scalar.

The function `mtlb_beta(A,B)` is used by `mfile2sci` to replace `beta(A,B)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_beta` calls:

- If `A` is a scalar but not `B` `Y = mtlb_beta(A,B)` may be replaced by `C=B;C(:)=A;Y = mtlb_beta(C,B);`
- If `B` is a scalar but not `A` `Y = mtlb_beta(A,B)` may be replaced by `C=A;C(:)=B;Y = mtlb_beta(A,C);`

Caution: `mtlb_beta` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_box — Matlab box emulation function

Description

There is no Scilab equivalent function for Matlab `box` but it can be easyly replaced.

The function `mtlb_box([axes_handle[,val]])` is used by `mfile2sci` to replace `box([axes_handle[,va]]l)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_box` calls:

- When called without input parameters, `mtlb_box()` may be replaced by `a=gca();if a.box=="on" then a.box="off";else a.box="on";end;`
- If `axes_handle` is a character string, `mtlb_box(axes_handle)` may be replaced by `a=gca();a.box=convstr(axes_handle,"l");`
- If `axes_handle` is a graphic handle `mtlb_box(axes_handle)` may be replaced by `if axes_handle.box=="on" then axes_handle.box="off";else axes_handle.box="on";end;`
- If `axes_handle` is a graphic handle and `val` is a character string `mtlb_box(axes_handle,val)` may be replaced by `axes_handle.box=convstr(val,"l");`

Caution: `mtlb_box` has not to be used for hand coded functions.

See Also

axes_properties

Authors

V.C.

Name

mtlb_close — Matlab close emulation function

Description

Scilab equivalent for Matlab `close` is different according to the current figure type (uicontrol or graphic one).

- When current figure is a uicontrol one: Scilab equivalent for Matlab `close` is `close`
- When current figure is a graphic one: Scilab equivalent for Matlab `close` is `xdel` or `delete`
- In Scilab we do not get an error status.

The function `mtlb_close([h])` is used by `mfile2sci` to replace `close([h])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_close` calls:

- If `h` is a uicontrol figure `mtlb_close(h)` may be replaced by `close(h)`
- If `h` is a graphic figure `mtlb_close(h)` may be replaced by `delete(h)`

Caution: `mtlb_close` has not to be used for hand coded functions.

See Also

xdel , delete , winsid , close

Authors

V.C.

Name

mtlb_colordef — Matlab colordef emulation function

Description

There is no Scilab equivalent function for Matlab `colordef` but there are equivalent instructions.

The function `h = mtlb_colordef(color_option)` or `h = mtlb_colordef(fig,color_option)` is used by `mfile2sci` to replace `colordef(color_option)` or `colordef(fig,color_option)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_colordef` calls:

- When called with one input parameter, if `color_option` is equal to "black" or "none" `mtlb_colordef(color_option)` may be replaced by `fig = gcf();fig.background = -1;`
- When called with one input parameter, if `color_option` is equal to "white" `mtlb_colordef(color_option)` may be replaced by `fig = gcf();fig.background = -2;`
- When called with two input parameters, if `fig` is a graphic handle and `color_option` is equal to "black" or "none" `mtlb_colordef(color_option)` may be replaced by `fig.background = -1;`
- When called with two input parameters, if `fig` is a graphic handle and `color_option` is equal to "white" `mtlb_colordef(color_option)` may be replaced by `fig.background = -2;`
- When called with two input parameters, if `fig` is equal to "new" and `color_option` is equal to "black" or "none" `mtlb_colordef(color_option)` may be replaced by `fig = scf(max(winsid())+1);fig.background = -1;`
- When called with two input parameters, if `fig` is equal to "new" and `color_option` is equal to "white" `mtlb_colordef(color_option)` may be replaced by `fig = scf(max(winsid())+1);fig.background = -2;`
- When called with one output parameter `h`, just add `h = fig;` after equivalent instructions.

Caution: `mtlb_colordef` has not to be used for hand coded functions.

See Also

figure_properties

Authors

V.C.

Name

mtlb_conv — Matlab conv emulation function

Description

Matlab `conv` and Scilab `convol` behave differently in some particular cases:

- With column vector inputs: if at least input is a column vector Matlab `conv` returns a column vector but Scilab `convol` always returns a row vector.

The function `mtlb_conv(u,v)` is used by `mfile2sci` to replace `conv(u,v)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_conv` calls:

- If `u` and `v` are row vectors, `mtlb_conv(u,v)` may be replaced by `convol(u,v)`
- If `u` or `v` is a column vector, `mtlb_conv(u,v)` may be replaced by `convol(u,v).'`
- If `u` and `v` are column vectors, `mtlb_conv(u,v)` may be replaced by `convol(u,v).'`

Scilab `convol` sometimes returns values that may be rounded using `clean` to have a closer result from Matlab one.

Caution: `mtlb_conv` has not to be used for hand coded functions.

See Also

clean

Authors

V.C.

Name

mtlb_cumprod — Matlab cumprod emulation function

Description

Matlab and Scilab `cumprod` behave differently in some particular cases:

- When used with one input (`cumprod(A)`), Matlab `cumprod` treats the values along the first non-singleton dimension of A as vectors while Scilab `cumprod` treats all A values.
- When used with two inputs (`cumprod(A,dim)`), Matlab tolerates `dim` to be greater than the number of dimensions of A while Scilab returns an error message in this case.

The function `R = mtlb_cumprod(A[,dim])` is used by `mfile2sci` to replace `cumprod(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_cumprod` calls:

- If `dim` is less equal to the number of dimensions of A `R = mtlb_cumprod(A,dim)` may be replaced by `R = cumprod(A,dim)`
- If `dim` is greater than then number of dimensions of A `R = mtlb_cumprod(A,dim)` may be replaced by `R = A`.

Caution: `mtlb_cumprod` has not to be used for hand coded functions.

See Also

firstnonsingleton

Authors

V.C.

Name

mtlb_cumsum — Matlab cumsum emulation function

Description

Matlab and Scilab `cumsum` behave differently in some particular cases:

- When used with one input (`cumsum(A)`), Matlab `cumsum` treats the values along the first non-singleton dimension of A as vectors while Scilab `cumsum` treats all A values.
- When used with two inputs (`cumsum(A,dim)`), Matlab tolerates `dim` to be greater than the number of dimensions of A while Scilab returns an error message in this case.

The function `R = mtlb_cumsum(A[,dim])` is used by `mfile2sci` to replace `cumsum(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_cumsum` calls:

- If `dim` is less equal to the number of dimensions of A `R = mtlb_cumsum(A,dim)` may be replaced by `R = cumsum(A,dim)`
- If `dim` is greater than then number of dimensions of A `R = mtlb_cumsum(A,dim)` may be replaced by `R = A`.

Caution: `mtlb_cumsum` has not to be used for hand coded functions.

See Also

firstnonsingleton

Authors

V.C.

Name

mtlb_dec2hex — Matlab dec2hex emulation function

Description

Matlab and Scilab `dec2hex` behave differently in some particular cases:

- With empty matrix: Matlab `dec2hex` returns "" but in Scilab you get [].
- With complex inputs: Matlab `dec2hex` automatically removes complex part of inputs but not Scilab one.
- Matlab `dec2hex` always returns a row vector but in Scilab `dec2hex` returns a value which have the same size as the input.
- Matlab `dec2hex` can have two inputs but not Scilab one.

The function `mtlb_dec2hex(D[,N])` is used by `mfile2sci` to replace `dec2hex(D[,N])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_dec2hex` calls:

- If `D` is not an empty matrix, `mtlb_dec2hex(D)` may be replaced by `matrix(dec2hex(real(D)),-1,1)` if `D` is complex and by `matrix(dec2hex(D),-1,1)` else.

Caution: `mtlb_dec2hex` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_delete — Matlab delete emulation function

Description

The function `mtlb_delete(A)` is used by `mfile2sci` to replace `delete(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_delete` calls:

- If `A` is a character string `mtlb_delete(A)` may be replaced by `mdelete(A)`
- If `A` is a graphic handle `mtlb_delete(A)` may be replaced by `delete(A)`

Caution: `mtlb_delete` has not to be used for hand coded functions.

See Also

mdelete

Authors

V.C.

Name

mtlb_diag — Matlab diag emulation function

Description

Matlab and Scilab `diag` behave differently in some particular cases:

- With character string matrices: Scilab `diag` function considers each character string as an object while Matlab considers each character individually.

The function `y = mtlb_diag(x[,dim])` is used by `mfile2sci` to replace `y = diag(x[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_diag` calls:

- If `x` is not a character string matrix `y = mtlb_diag(x[,dim])` may be replaced by `y = diag(x[,dim])`

Caution: `mtlb_diag` has not to be used for hand coded functions.

See Also

Matlab-Scilab_character_strings

Authors

V.C.

Name

mtlb_diff — Matlab diff emulation function

Description

Matlab and Scilab `diff` behave differently in some particular cases:

- With two input parameters: Scilab `diff` considers all values of first input as a vector what Matlab does not.
- With three input parameters: If dimension of first input along dimension given by third parameter reaches 1 before n interations have been made,Matlab switches to next non-singleton dimension what Scilab does not.

The function `mtlb_diff(A[,n[,dim]])` is used by `mfile2sci` to replace `diff(A[,n[,dim]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_diff` calls:

- With two inputs, if `A` is a scalar or a vector `mtlb_diff(A,n)` may be replaced by `diff(A,n)`
- With three inputs, if size of A along dimension given by `dim` can not reach 1 `mtlb_diff(A,n,dim)` may be replaced by `diff(A,n,dim)`

Caution: `mtlb_diff` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_dir — Matlab dir emulation function

Description

Matlab and Scilab `dir` behave differently:

- When result is stored in a variable: Matlab `dir` returns a structure but Scilab `dir` returns a 'dir' tlist, so data can not be extracted in the same way.

The function `mtlb_dir([path])` is used by `mfile2sci` to replace `dir([path])` when output is stored in a variable. There is no replacement possibility for it, else (when `mtlb_dir` is replaced by `dir`) data can not be extracted like in Matlab. For example, Scilab equivalent for Matlab `L=dir;file_name=L(1).name;` is `L=dir();file_name=L.name(1);`.

Caution: `mtlb_dir` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_double — Matlab double emulation function

Description

Matlab and Scilab `double` behave differently in some particular cases:

- With character string input: Scilab `double` does not work with this type of input while Matlab `double` returns a matrix of ASCII codes.
- With boolean input: Scilab `double` does not work with this type of input while Matlab `double` returns a matrix of double values.

The function `mtlb_double(A)` is used by `mfile2sci` to replace `double(A)` when it was not possible to know what were theinput while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_double` calls:

- If `A` is a character string, `mtlb_double(A)` may be replaced by `asciimat(A)`
- If `A` is a boolean matrix, `mtlb_double(A)` may be replaced by `bool2s(A)`
- If `A` is a double matrix, `mtlb_double(A)` may be replaced by `A`

Caution: `mtlb_double` has not to be used for hand coded functions.

See Also

asciimat , bool2s

Authors

V.C.

Name

mtlb_e — Matlab extraction emulation function

Description

Matlab and Scilab extraction behave differently in some particular cases:

- When extracting data from a matrix with a vector as index: Matlab returns a row vector and Scilab returns a column vector.
- When extracting data from a character string matrix: due to the fact that character string matrices in Matlab can be adressed as any other matrix (each character can be adressed), extraction in such a type of matrix does not differ from other. But in Scilab it can't be done so and `part` function has to be used.

The function `mtlb_e(B,k)` is used by `mfile2sci` to replace `A=B(k)` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_e` calls:

- If `B` is a vector `A=mtlb_e(B,k)` may be replaced by `A=B(k)`
- If `B` is a matrix `A=mtlb_e(B,k)` may be replaced by `A=B(k).'`
- If `B` is a character string matrix and k is a scalar or a vector `A=mtlb_e(B,k)` may be replaced by `A=part(B,k)`

Caution: `mtlb_e` has not to be used for hand coded functions.

See Also

Matlab-Scilab_character_strings , part

Authors

V.C.

Name

mtlb_echo — Matlab echo emulation function

Description

There is no equivalent for Matlab `echo` in Scilab but some cases can be replaced by calls to Scilab `mode`:

- `echo`: is equivalent to Scilab `mode(abs(mode()-1))` for scripts and non-compiled functions
- `echo on`: is equivalent to Scilab `mode(1)` for scripts and non-compiled functions
- `echo off`: is equivalent to Scilab `mode(0)`

The function `mtlb_echo(arg1[,arg2])` is used by `mfile2sci` to replace `echo arg1 [arg2]` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_echo` calls:

- If `arg1` is equal to "on" `mtlb_echo(arg1)` may be replaced by `mode(1)`
- If `arg1` is equal to "off" `mtlb_echo(arg1)` may be replaced by `mode(0)`

Caution: `mtlb_echo` has not to be used for hand coded functions.

See Also

mode

Authors

V.C.

Name

mtlb_eig — Matlab eig emulation function

Description

WARNING: This function is obsolete and will be removed in version 5.1.2, please use spec instead.

See Also

spec

Authors

V.C.

Name

mtlb_eval — Matlab eval emulation function

Description

Scilab equivalent for Matlab `eval` is different according to its inputs and outputs

The function `mtlb_eval(str1,str2)` is used by `mfile2sci` to replace `eval` because it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_eval` calls:

- When called with one input and no output, `mtlb_eval(str1)` may be replaced by `evstr(str1)` if `str1` is a valid Scilab expression or by `execstr(str1)` if `str1` is a valid Scilab instruction.
- When called with one input and one output, `val=mtlb_eval(str1)` may be replaced by `val=evstr(str1)` if `str1` is a valid Scilab instruction.
- When called with two inputs and no output, `mtlb_eval(str1,str2)` may be replaced by: `if execstr(str1,"errcatch")<>0 then execstr(str2);end` if `str1` and `str2` are valid Scilab instructions.
- When called with more than one output and one input, `[val1,val2,...]=mtlb_eval(str1)` may be replaced by `execstr("[val1,val2,...]"+str1)` if `str1` is a valid Scilab instruction.
- When called with two inputs and more than one output, `[val1,val2,...]=mtlb_eval(str1,str2)` may be replaced by:

```
if execstr("[val1,val2,...]"+str1,"errcatch")<>0 then
  execstr("[val1,val2,...]"+str2);
end
```

  if `str1` and `str2` are valid Scilab instructions.

Caution: `mtlb_eval` has not to be used for hand coded functions.

See Also

evstr , execstr

Authors

V.C.

Name

mtlb_exist — Matlab exist emulation function

Description

There is no Scilab equivalent for Matlab `exist` except when input is a variable. Scilab `mtlb_exist` is a partial emulation of of this function.

The function `r = mtlb_exist(nam[,tp])` is used by `mfile2sci` to replace `exist(nam[,tp])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_exist` calls:

- When called with one input and if `nam` is a variable name, `mtlb_exist(nam)` may be replaced by `exists(nam)`

Caution: `mtlb_exist` has not to be used for hand coded functions.

See Also

exists

Authors

V.C.

Name

mtlb_eye — Matlab eye emulation function

Description

Matlab and Scilab `eye` behave differently in some particular cases:

- With a scalar input: Matlab `eye` returns a n x n matrix while Scilab returns a 1.

The function `mtlb_eye(A)` is used by `mfile2sci` to replace `eye(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_eye` calls:

- If `A` is a scalar `mtlb_eye(A)` may be replaced by `eye(A,A)`
- If `A` is not a scalar `mtlb_eye(A)` may be replaced by `eye(A)`

Caution: `mtlb_eye` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_false — Matlab false emulation function

Description

There is no Scilab equivalent for Matlab `false`. However, Scilab `zeros` returns a result interpreted in an equivalent way for Scilab.

Matlab `false` and Scilab `zeros` behave differently in some particular cases:

- With a scalar input: Matlab `false` returns a n x n matrix of zeros while Scilab `zeros` returns a 0.

The function `mtlb_false(A)` is used by `mfile2sci` to replace `false(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_false` calls:

- If `A` is a scalar `mtlb_false(A)` may be replaced by `zeros(A,A)`
- If `A` is not a scalar `mtlb_false(A)` may be replaced by `zeros(A)`

Caution: `mtlb_false` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_fft — Matlab fft emulation function

Description

Matlab and Scilab `fft` behave differently in some particular cases:

- With one input parameter: If input is a scalar or vector Scilab equivalent for Matlab `fft` is `fft(...,-1)` else if input is a matrix Scilab equivalent for Matlab `fft` is `fft(...,-1,2,1)`

The function `mtlb_fft(X[,n,[job]])` is used by `mfile2sci` to replace `fft(X[,n,[job]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_fft` calls:

- If `X` is a scalar or a vector `mtlb_fft(X,-1)` may be replaced by `fft(X,-1)`
- If `X` is a matrix `mtlb_fft(X,-1)` may be replaced by `fft(X,-1,2,1)`

Caution: `mtlb_fft` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_fftshift — Matlab fftshift emulation function

Description

Matlab and Scilab `fftshift` behave differently in some particular cases:

- With character string input: due to the fact that character strings are not considered in the same way in Matlab and Scilab, results can be different for this kind of input.
- With two inputs: Matlab `fftshift` can work even if `dim` parameter is greater than number of dimensions of first input.

The function `mtlb_fftshift(A[,dim])` is used by `mfile2sci` to replace `fftshift(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_fftshift` calls:

- If `A` is not a character string matrix, `mtlb_fftshift(A)` may be replaced by `fftshift(A)`
- If `A` is not a character string matrix and `dim` is not greater than `size(size(a),"*")`, `mtlb_fftshift(A,dim)` may be replaced by `fftshift(A,dim)`

Caution: `mtlb_fftshift` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_find — Matlab find emulation function

Description

Matlab and Scilab `find` behave differently in some particular cases:

- With column vectors and matrices as input: Matlab `find` returns column vectors while Scilab returns row vectors.
- When called with three outputs: Matlab `find` can have three outputs but Scilab not.

The function `[i[,j[,v]]] = mtlb_find(B)` is used by `mfile2sci` to replace `[i[,j[,v]]] = find(B)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_find` calls:

- When called with one output, if `B` is a row vector `i = mtlb_find(B)` may be replaced by `i = find(B)`
- When called with one output, if `B` is not a row vector `i = mtlb_find(B)` may be replaced by `i = matrix(find(B),-1,1)`
- When called with two outputs, if `B` is a row vector `[i,j] = mtlb_find(B)` may be replaced by `[i,j] = find(B)`
- When called with two outputs, if `B` is not a row vector `[i,j] = mtlb_find(B)` may be replaced by `[i,j] = find(B);i = matrix(i,-1,1);j = matrix(j,-1,1);`

Caution: `mtlb_find` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_findstr — Matlab findstr emulation function

Description

There is no Scilab equivalent for Matlab `findstr`.

The function `mtlb_findstr(A,B)` is used by `mfile2sci` to replace `findstr(A,B)` when it was not possible to know what were the operands/inputs[CUSTOM] while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_findstr` calls:

- If `A` is always longer than `B` (i.e. `findstr` can be replaced by `strfind` in Matlab, `mtlb_findstr(A,B)` may be replaced by `strindex(A,B)`

Caution: `mtlb_findstr` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_fliplr — Matlab fliplr emulation function

Description

Matlab and Scilab `fliplr` behave differently in some particular cases:

- With character string matrices: due to te fact that Scilab and Matlab do not consider character string matrices in the same way, result can be different for input of this type.

The function `mtlb_fliplr(A)` is used by `mfile2sci` to replace `fliplr(A)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_fliplr` calls:

- If `A` is not a character string matrix, `mtlb_fliplr(A)` may be replaced by `A(:,$:-1:1)`

Caution: `mtlb_fliplr` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_fopen — Matlab fopen emulation function

Description

Matlab `fopen` and Scilab `mopen` behave differently in some particular cases:

- Scilab function returns no usable error message
- Scilab file identified does not exist in case of error but Matlab one is set to `-1`.
- Matlab function can work with inputs which do not exist in Scilab such as permission options...

The function `mtlb_fopen(filename,permission)` is used by `mfile2sci` to replace `mopen(filename,permission)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_fopen` calls:

- If error message is not used and no error can occurs, `mtlb_fopen(filename,permission)` may be replaced by `mopen(filename,permission,0)`

Caution: `mtlb_fopen` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_format — Matlab format emulation function

Description

Matlab and Scilab `format` behave differently in some particular cases:

- Some Matlab formats do not exist in Scilab
- Calling sequence for `format` is different in Scilab and Matlab

The function `mtlb_format(type,prec)` is used by `mfile2sci` to replace `format([type prec])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_format` calls:

- If `type=""` and `prec=""` `mtlb_format("","")` may be replaced by `format("v",6)`
- If `type="+"` and `prec=""` `mtlb_format("+","")` may be replaced by `format(6)`
- If `type="long"` and `prec=""` `mtlb_format("long","")` may be replaced by `format(16)`
- If `type="long"` and `prec="e"` `mtlb_format("long","e")` may be replaced by `format("e"16)`
- If `type="long"` and `prec="g"` `mtlb_format("long","g")` may be replaced by `format("e"16)`
- If `type="short"` and `prec=""` `mtlb_format("short","")` may be replaced by `format(6)`
- If `type="short"` and `prec="e"` `mtlb_format("short","e")` may be replaced by `format("e"6)`
- If `type="short"` and `prec="g"` `mtlb_format("short","g")` may be replaced by `format("e"6)`

Caution: `mtlb_format` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_fprintf — Matlab fprintf emulation function

Description

There is no Scilab exact equivalent for Matlab `fprintf`. Scilab `mfprintf` and Matlab `fprintf` behave differently in many cases, but they are equivalents in some cases.

The function `mtlb_fprintf(varargin)` is used by `mfile2sci` to replace `fprintf`. This function will determine the correct semantic at run time. It is sometimes possible to replace calls to `mtlb_fprintf` by calls to mfprintf.

Caution: `mtlb_fprintf` has not to be used for hand coded functions.

See Also

mfprintf

Authors

V.C.

Name

mtlb_fread — Matlab fread emulation function

Description

There is no Scilab equivalent for Matlab `fread`. Scilab `mget` and Matlab `fread` behave differently in many cases, but they are equivalents in some cases.

The function `mtlb_fread(varargin)` is used by `mfile2sci` to replace `fread`. This function will determine the correct semantic at run time. It is sometimes possible to replace calls to `mtlb_fread` by calls to mget.

Caution: `mtlb_fread` has not to be used for hand coded functions.

See Also

mget

Authors

V.C.

Name

mtlb_fscanf — Matlab fscanf emulation function

Description

There is no Scilab exact equivalent for Matlab `fscanf`. Scilab `mfscanf` and Matlab `fscanf` behave differently in many cases, but they are equivalents in some cases.

The function `mtlb_fscanf(varargin)` is used by `mfile2sci` to replace `fscanf`. This function will determine the correct semantic at run time. It is sometimes possible to replace calls to `mtlb_fscanf` by calls to mfscanf.

Caution: `mtlb_fscanf` has not to be used for hand coded functions.

See Also

mfscanf

Authors

V.C.

Name

mtlb_full — Matlab full emulation function

Description

Matlab and Scilab `full` behave differently in some particular cases:

- With character strings input: Matlab `full` can be used with character string input while Scilab function cannot.
- With boolean input: Matlab `full` can be used with boolean input while Scilab function cannot.

The function `mtlb_full(A)` is used by `mfile2sci` to replace `full(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_full` calls:

- If `A` is a double matrix `mtlb_full(A)` may be replaced by `full(A)`
- If `A` is a boolean matrix `mtlb_full(A)` may be replaced by `full(bool2s(A))`

Caution: `mtlb_full` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_fwrite — Matlab fwrite emulation function

Description

There is no Scilab equivalent for Matlab `fwrite`. Scilab `mput` and Matlab `fwrite` behave differently in many cases, but they are equivalents in some cases.

The function `mtlb_fwrite(varargin)` is used by `mfile2sci` to replace `fwrite`. This function will determine the correct semantic at run time. It is sometimes possible to replace calls to `mtlb_fwrite` by calls to mput.

Caution: `mtlb_fwrite` has not to be used for hand coded functions.

See Also

mput

Authors

V.C.

Name

mtlb_grid — Matlab grid emulation function

Description

There is no Scilab equivalent function for Matlab `grid` but there are equivalent instructions.

The function `mtlb_grid(flag_or_handle[,flag])` is used by `mfile2sci` to replace `grid(flag_or_handle[,flag])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_grid` calls:

- With one input, if `flag` is equal to "on" `mtlb_grid(flag)` may be replaced by `set(gca(),"grid",[1 1])`
- With one input, if `flag` is equal to "off" `mtlb_grid(flag)` may be replaced by `set(gca(),"grid",[-1 -1])`
- With two inputs, if `flag` is equal to "on" `mtlb_grid(axes_handle,flag)` may be replaced by `axes_handle.grid=[1 1]`
- With two inputs, if `arg2` is equal to "off" `mtlb_grid(axes_handle,flag)` may be replaced by `axes_handle.grid=[-1 -1]`

Caution: `mtlb_grid` has not to be used for hand coded functions.

See Also

axes_properties

Authors

V.C.

Name

mtlb_hold — Matlab hold emulation function

Description

There is no Scilab equivalent function for Matlab `hold` but there are equivalent instructions.

The function `mtlb_hold(flag)` is used by `mfile2sci` to replace `hold flag` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_hold` calls:

- If `flag` is equal to "on" `mtlb_hold(flag)` may be replaced by `set(gca(),"auto_clear","off")`
- If `flag` is equal to "off" `mtlb_hold(flag)` may be replaced by `set(gca(),"auto_clear","on")`

Caution: `mtlb_hold` has not to be used for hand coded functions.

See Also

axes_properties

Authors

V.C.

Name

mtlb_i — Matlab insertion emulation function

Description

Matlab and Scilab insertion behave differently in some particular cases:

- When inserting a matrix in a variable: Matlab automaticaly adjusts output variable to fit with matrix to insert but not Scilab. For example, with A=1, A([1,2,3,4])=[1,2;3,4]) returns an error in Scilab while in Matlab we get A=[1,2,3,4]. If values miss comparing to indexes, Matlab fills output value with 0.
- When inserting data into a character string matrix: due to the fact that character string matrices in Matlab can be adressed as any other matrix (each character can be adressed), insertion in such a type of matrix does not differ from other. But in Scilab it can't be done so...`mtlb_is` is an alternative.

The function `A=mtlb_i(A,k,B)` is used by `mfile2sci` to replace `A(k)=B` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_i` calls:

- If `A` is not a vector `A=mtlb_i(A,k,B)` may be replaced by `A(k)=B`
- If `A` and `B` are both row or column vectors `A=mtlb_i(A,k,B)` may be replaced by `A(k)=B`

Caution: `mtlb_i` has not to be used for hand coded functions.

See Also

Matlab-Scilab_character_strings , mtlb_is

Authors

V.C.

Name

mtlb_ifft — Matlab ifft emulation function

Description

Matlab `ifft` and Scilab `fft` behave differently in some particular cases:

- With one input parameter: If input is a scalar or vector Scilab equivalent for Matlab `ifft(A)` is `fft(A,1)` else if input is a matrix Scilab equivalent for Matlab `fft` is `fft(A,1,2,1)`

The function `mtlb_ifft(X[,n,[job]])` is used by `mfile2sci` to replace `ifft(X[,n,[job]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_ifft` calls:

- If `X` is a scalar or a vector `mtlb_ifft(X,1)` may be replaced by `fft(X,1)`
- If `X` is a matrix `mtlb_ifft(X,1)` may be replaced by `fft(X,1,2,1)`

Caution: `mtlb_ifft` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_imp — Matlab colon emulation function

Description

Matlab and Scilab colon behave differently in some particular cases:

- With empty matrices: The `:` operator must be used with scalars in Scilab and gives an error message when used with empty matrices while Matlab returns [] in these cases.

The function `mtlb_imp(A,B[,C])` is used by `mfile2sci` to replace `A:B[:C]` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_imp` calls:

- If `A`, `B` and `C` are not empty matrices `mtlb_imp(A,B[,C])` may be replaced by `A:B[:C]`

Caution: `mtlb_imp` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_int16 — Matlab int16 emulation function

Description

Matlab and Scilab `int16` behave differently in some particular cases:

- With complex input: Matlab `int16` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `int16` returns 32767 and Scilab returns -32768.
- With `%nan`: Matlab `int16` returns 0 and Scilab returns -32768.
- With `-%nan`: Matlab `int16` returns 0 and Scilab returns -32768.

The function `mtlb_int16(A)` is used by `mfile2sci` to replace `int16(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_int16` calls:

- If `A` does not contain `%inf`, `%nan` or `-%nan` values `mtlb_int16(A)` may be replaced by `int16(A)`

Caution: `mtlb_int16` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_int32 — Matlab int32 emulation function

Description

Matlab and Scilab `int32` behave differently in some particular cases:

- With complex input: Matlab `int32` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `int32` returns 2147483647 and Scilab returns -2147483648.
- With `%nan`: Matlab `int32` returns 0 and Scilab returns -2147483648.
- With `-%nan`: Matlab `int32` returns 0 and Scilab returns -2147483648.

The function `mtlb_int32(A)` is used by `mfile2sci` to replace `int32(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_int32` calls:

- If `A` does not contain `%inf`, `%nan` or `-%nan` values `mtlb_int32(A)` may be replaced by `int32(A)`

Caution: `mtlb_int32` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_int8 — Matlab int8 emulation function

Description

Matlab and Scilab `int8` behave differently in some particular cases:

- With complex input: Matlab `int8` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `int8` returns 127 and Scilab returns 0.
- With `-%inf`: Matlab `int8` returns -128 and Scilab returns 0.

The function `mtlb_int8(A)` is used by `mfile2sci` to replace `int8(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_int8` calls:

- If `A` does not contain `%inf` or `-%inf` values `mtlb_int8(A)` may be replaced by `int8(A)`

Caution: `mtlb_int8` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_is — Matlab string insertion emulation function

Description

Matlab and Scilab insertion behave differently for strings due to the fact that they do not consider character strings in the same way.

The function `str = mtlb_is(sto,sfrom,i,j)` is used by `mfile2sci` to replace insertion operations for character strings. This function will determine the correct semantic at run time. There is no replacement possibility for it.

Caution: `mtlb_is` has not to be used for hand coded functions.

See Also

Matlab-Scilab_character_strings , mtlb_i

Authors

V.C.

Name

mtlb_isa — Matlab isa emulation function

Description

There is no Scilab equivalent function for Matlab `isa` but some equivalent expressions can be used when the object "class" exists in Scilab.

The function `mtlb_isa(OBJ,class)` is used by `mfile2sci` to replace `isa(OBJ,class)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_isa` calls:

- If `class` is equal to "logical", `mtlb_isa(OBJ,class)` may be replaced by `type(OBJ)==4`
- If `class` is equal to "char", `mtlb_isa(OBJ,class)` may be replaced by `type(OBJ)==10`
- If `class` is equal to "numeric", `mtlb_isa(OBJ,class)` may be replaced by `or(type(OBJ)==[1,5,8])`
- If `class` is equal to "intX" (X being equal to 8, 16, or 32), `mtlb_isa(OBJ,class)` may be replaced by `typeof(OBJ)=="intX"`
- If `class` is equal to "uintX" (X being equal to 8, 16, or 32), `mtlb_isa(OBJ,class)` may be replaced by `typeof(OBJ)=="uintX"`
- If `class` is equal to "single", `mtlb_isa(OBJ,class)` may be replaced by `type(OBJ)==1`
- If `class` is equal to "double", `mtlb_isa(OBJ,class)` may be replaced by `type(OBJ)==1`
- If `class` is equal to "cell", `mtlb_isa(OBJ,class)` may be replaced by `typeof(OBJ)=="ce"`
- If `class` is equal to "struct", `mtlb_isa(OBJ,class)` may be replaced by `typeof(OBJ)=="st"`
- If `class` is equal to "function_handle", `mtlb_isa(OBJ,class)` may be replaced by `type(OBJ)==13`
- If `class` is equal to "sparse", `mtlb_isa(OBJ,class)` may be replaced by `type(OBJ)==5`
- If `class` is equal to "lti", `mtlb_isa(OBJ,class)` may be replaced by `typeof(OBJ)=="state-space"`

Caution: `mtlb_isa` has not to be used for hand coded functions.

See Also

type , typeof

Authors

V.C.

Name

mtlb_isfield — Matlab isfield emulation function

Description

There is no Scilab equivalent function for Matlab `isfield(st,f)` and equivalent expressions behave differently in some particular cases:

- If `st` is not a structure: Scilab equivalent returns an error message but Matlab returns `0`.

The function `mtlb_isfield(st,f)` is used by `mfile2sci` to replace `isfield(st,f)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_isfield` calls:

- If `st` is a structure `tf = mtlb_isfield(st,f)` may be replaced by `allf=getfield(1,st);tf=or(allf(3:$)==f);`
- If `st` is not a structure `tf = mtlb_isfield(st,f)` may be replaced by `tf=%F;`

Caution: `mtlb_isfield` has not to be used for hand coded functions.

See Also

getfield

Authors

V.C.

Name

mtlb_isletter — Matlab isletter emulation function

Description

There is no Scilab equivalent function for Matlab `isletter` and equivalent instructions are quite ugly, so `mfile2sci` uses `mtlb_isletter(A)` to replace `isletter(A)`. If you want to have a more efficient code it is possible to replace `mtlb_isletter` calls:

- If `A` is not a character string `tf = mtlb_isletter(A)` may be replaced by `tf = zeros(A)`
- If `A` is a character string `tf = mtlb_isletter(A)` may be replaced by `tf = (asciimat(A)>=65&asciimat(A)<=90)|(asciimat(A)>=97&asciimat(A)<=122)`

Caution: `mtlb_isletter` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_isspace — Matlab isspace emulation function

Description

There is no Scilab function equivalent for Matlab `isspace` but its behavior can be reproduced.

The function `mtlb_isspace(A)` is used by `mfile2sci` to replace `isspace(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_isspace` calls:

- If `A` is a character string matrix `mtlb_isspace(A)` may be replaced by `asciimat(A)==32`
- If `A` is not a character string matrix `mtlb_isspace(A)` may be replaced by `zeros(A)`

Caution: `mtlb_isspace` has not to be used for hand coded functions.

See Also

asciimat

Authors

V.C.

Name

mtlb_l — Matlab left division emulation function

Description

Matlab and Scilab left division behave differently in some particular cases:

- With character string operands: The \ operator can not be used into Scilab with character strings, while in Matlab it can. And in this case, result is transposed in a very strange way...

The function `mtlb_l(A,B)` is used by `mfile2sci` to replace `A\B` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_l` calls:

- If both `A` and `B` are not character strings `mtlb_l(A,B)` may be replaced by `A\B`.

Caution: `mtlb_l` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_legendre — Matlab legendre emulation function

```
P = mtlb_legendre(n,X)
P = mtlb_legendre(n,X[,normflag])
```

Parameters

X
: a scalar, vector, matrix or hypermatrix with elements in [-1,1]

n
: a positive scalar integer

normflag
: a string ('sch' or 'norm')

Description

Matlab and Scilab `legendre` behave differently in some particular cases:

- Scilab `legendre(m,n,X)` evaluates the `legendre` function of degree n and order n for the `X` elements. Matlab `legendre(n,X)` evaluates the Legendre functions of degree `n` and order `m`=0,1,...,n. (emulated by mtlb_legendre) for the `X` elements.
- The option normflag= `'sch'` doesn't exist for Scilab legendre (emulated)
- If `X` is a hypermatrix then Scilab `legendre(n,X)` doesn't work (emulated)

The function `mtlb_legendre(n,X[,normflag])` is used by `mfile2sci` to replace `legendre(n,X[,normflag])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_legendre` call:

- If `X` is a scalar, a vector or a matrix `mtlb_legendre(n,X[,'norm'])` may be replaced by `legendre(n,0:n,X[,'norm'])`

Caution: `mtlb_legendre` has not to be used for hand coded functions.

Authors

F.B.

Name

mtlb_linspace — Matlab linspace emulation function

Description

Matlab and Scilab `linspace` behave differently in some particular cases:

- With character string inputs: Matlab `linspace(A,B[,n])` returns a character string vector if `A` and/or `B` are character strings, but Scilab function does not work with such inputs.

The function `mtlb_linspace(A,B[,n])` is used by `mfile2sci` to replace `linspace(A,B[,n])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_linspace` calls:

- If `A` and `B` are not character strings `mtlb_linspace(A,B[,n])` may be replaced by `linspace(A,B[,n])`
- If `A` or `B` are character strings `mtlb_linspace(A,B[,n])` may be replaced by `ascii(linspace(ascii(A),ascii(B)[,n]))`

Caution: `mtlb_linspace` has not to be used for hand coded functions.

See Also

ascii

Authors

V.C.

Name

mtlb_logic — Matlab logical operators emulation function

Description

Matlab and Scilab logical operator behave differently in some particular cases:

- With complex operands: The <, <=, > and >= operators can not be used into Scilab with complex operands, while in Matlab they can. And in this case, only real part of complex operands is compared.
- With empty matrices: If both operands of <, <=, > and >= operators are empty matrices, Scilab returns an error message, while Matlab returns an empty matrix. For operators == and ~=, if at least one operand is an empty matrix, Matlab returns [] while Scilab returns a boolean value %T or %F.

The function `mtlb_logic(A,symbol,B)` (with "symbol" a character string containing the operator symbol) is used by `mfile2sci` to replace `A symbol B` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_logic` calls:

- If both `A` and `B` are not complex values nor empty matrices `mtlb_logic(A,symbol,B)` may be replaced by `A symbol B`.

Caution: `mtlb_logic` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_logical — Matlab logical emulation function

Description

There is no Scilab equivalent function for Matlab `logical` but its behavior can be easyly reproduced.

The function `mtlb_logical(A)` is used by `mfile2sci` to replace `logical(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_logical` calls:

- If `A` is a boolean matrix `mtlb_logical(A)` may be replaced by `A`
- If `A` is not an empty matrix `mtlb_logical(A)` may be replaced by `A<>[]`
- If `A` is an empty matrix `mtlb_logical(A)` may be replaced by `[]`

Caution: `mtlb_logical` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_lower — Matlab lower emulation function

Description

Matlab `lower(A)` and Scilab `convstr(A,"l")` behave differently in some particular cases:

- If `A` is not a chacarter string matrix: Matlab `lower` can be used with a not-character-string `A` but not Scilab `convstr`.

The function `mtlb_lower(A)` is used by `mfile2sci` to replace `lower(A)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_lower` calls:

- If `A` is a character string matrix `mtlb_lower(A)` may be replaced by `convstr(A,"l")`
- If `A` is not a character string matrix `mtlb_lower(A)` may be replaced by `A`

Caution: `mtlb_lower` has not to be used for hand coded functions.

See Also

convstr

Authors

V.C.

Name

mtlb_max — Matlab max emulation function

Description

Matlab and Scilab `max` behave differently in some particular cases:

- With complex values: Matlab `max` can be used with complex values but not Scilab function.
- When called with one input: Matlab `max` threats values along the first non-singleton dimension but Scilab threats all input values.
- When called with two inputs: if one is an empty matrix, Scilab returns an error message but Matlab returns [].
- When called with three inputs: if `dim` parameter is greater than number of dimensions of first input, Scilab returns an error message and Matlab returns the first input.

The function `[r[,k]] = mtlb_max(A[,B[,dim]])` is used by `mfile2sci` to replace `[r[,k]] = max(A[,B[,dim]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_max` calls:

- When called with one input, if `A` is a vector or a scalar `[r[,k]] = mtlb_max(A)` may be replaced by `max(A)`
- When called with one input, if `A` is a matrix `[r[,k]] = mtlb_max(A)` may be replaced by `max(A,"r")`
- When called with two inputs, if `A` and `B` are real matrices and not empty matrices `[r[,k]] = mtlb_max(A,B)` may be replaced by `max(A,B)`
- When called with three inputs, if `dim` is lesser than the number of dimensions of `A` `[r[,k]] = mtlb_max(A,[],dim)` may be replaced by `max(A,dim)`

Caution: `mtlb_max` has not to be used for hand coded functions.

See Also

firstnonsingleton

Authors

V.C.

Name

mtlb_min — Matlab min emulation function

Description

Matlab and Scilab `min` behave differently in some particular cases:

- With complex values: Matlab `min` can be used with complex values but not Scilab function.
- When called with one input: Matlab `min` threats values along the first non-singleton dimension but Scilab threats all input values.
- When called with two inputs: if one is an empty matrix, Scilab returns an error message but Matlab returns [].
- When called with three inputs: if `dim` parameter is greater than number of dimensions of first input, Scilab returns an error message and Matlab returns the first input.

The function `[r[,k]] = mtlb_min(A[,B[,dim]])` is used by `mfile2sci` to replace `[r[,k]] = min(A[,B[,dim]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_min` calls:

- When called with one input, if `A` is a vector or a scalar `[r[,k]] = mtlb_min(A)` may be replaced by `min(A)`
- When called with one input, if `A` is a matrix `[r[,k]] = mtlb_min(A)` may be replaced by `min(A,"r")`
- When called with two inputs, if `A` and `B` are real matrices and not empty matrices `[r[,k]] = mtlb_min(A,B)` may be replaced by `min(A,B)`
- When called with three inputs, if `dim` is lesser than the number of dimensions of `A` `[r[,k]] = mtlb_min(A,[],dim)` may be replaced by `min(A,dim)`

Caution: `mtlb_min` has not to be used for hand coded functions.

See Also

firstnonsingleton

Authors

V.C.

Name

mtlb_more — Matlab more emulation function

Description

Matlab `more` and Scilab `lines` behave differently in some particular cases:

- With character strings as input: Matlab `more` can take "on" and "off" as input but not Scilab `lines` but there are equivalents (0 and 60).

The function `mtlb_more(in)` is used by `mfile2sci` to replace `more(in)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_more` calls:

- If `in` is equal to "on" `mtlb_more(in)` may be replaced by `lines(60)`
- If `in` is equal to "off" `mtlb_more(in)` may be replaced by `lines(0)`
- If `in` is a double value `mtlb_more(in)` may be replaced by `lines(in)`

Caution: `mtlb_more` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_num2str — Matlab num2str emulation function

Description

Matlab `num2str` and Scilab equivalents (`string`, `msprintf`) behave differently in some particular cases:

- With two input parameters, the second giving precision: There is no Scilab equivalent function, but `msprintf` can be used to emulate.
- With two input parameters, the second giving format: Scilab equivalent for Matlab `num2string(a,format)` is `msprintf(format,a)`.

The function `mtlb_num2str(x,f)` is used by `mfile2sci` to replace `num2str(x,f)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_num2str` calls:

- If `f` is a character string `mtlb_num2str(x,f)` may be replaced by `msprintf(f,x)`

Caution: `mtlb_num2str` has not to be used for hand coded functions.

See Also

string , msprintf

Authors

V.C.

Name

mtlb_ones — Matlab ones emulation function

Description

Matlab and Scilab `ones` behave differently in some particular cases:

- With a scalar input: Matlab `ones` returns a n x n matrix while Scilab returns a 1.

The function `mtlb_ones(A)` is used by `mfile2sci` to replace `ones(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_ones` calls:

- If `A` is a scalar `mtlb_ones(A)` may be replaced by `ones(A,A)`
- If `A` is not a scalar `mtlb_ones(A)` may be replaced by `ones(A)`

Caution: `mtlb_ones` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_plot — Matlab plot emulation function

Description

Matlab `plot` and Scilab `plot2d` behave differently.

The function `mtlb_plot(varargin)` is used by `mfile2sci` to replace `plot(varargin)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_plot` calls when there is no output value, however in this case, you have to set colors manualy:

- With one input, if `Y` is real, `mtlb_plot(Y)` may be replaced by `plot2d(Y)`
- With one input, if `Y` is complex, `mtlb_plot(Y)` may be replaced by `plot2d(real(Y),imag(Y))`
- With two inputs `X` and `Y`, if `Y` is not a character string, `mtlb_plot(X,Y)` may be replaced by `plot2d(X,Y)`

Caution: `mtlb_plot` has not to be used for hand coded functions.

See Also

plot2d

Authors

V.C.

Name

mtlb_prod — Matlab prod emulation function

Description

Matlab and Scilab `prod` behave differently in some particular cases:

- When called with one input: Matlab `prod` threts the values along the first non-singleton dimension of input while Scilab `prod` threats all values of input.
- When called with two inputs: Matlab `prod` can be used with second input giving a non-existing dimension of first input while Scilab `prod` returns an error message.

The function `mtlb_prod(A[,dim])` is used by `mfile2sci` to replace `prod(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_prod` calls:

- When called with one input, if `A` is an empty matrix, a scalar or a vector, `mtlb_prod(A)` may be replaced by `prod(A)`
- When called with one input, if `A` is a not-empty matrix, `mtlb_prod(A)` may be replaced by `prod(A,1)`
- When called with one input, if `A` is a multidimensional array, `mtlb_prod(A)` may be replaced by `prod(A,firstnonsingleton(A))`
- When called with two inputs, if `dim` is lesser than the number of dimensions of `A` `mtlb_prod(A,dim)` may be replaced by `prod(A,dim)`

Caution: `mtlb_prod` has not to be used for hand coded functions.

See Also

firstnonsingleton

Authors

V.C.

Name

mtlb_rand — Matlab rand emulation function

Description

Matlab and Scilab `rand` behave differently in some particular cases:

- With a scalar input: Matlab `rand` returns a n x n matrix while Scilab returns a scalar.

The function `mtlb_rand(A)` is used by `mfile2sci` to replace `rand(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_rand` calls:

- If `A` is a scalar `mtlb_rand(A)` may be replaced by `rand(A,A)`
- If `A` is not a scalar `mtlb_rand(A)` may be replaced by `rand(A)`

Caution: `mtlb_rand` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_randn — Matlab randn emulation function

Description

Matlab `rand` and Scilab `rand(...,"normal")` behave differently in some particular cases:

- With a scalar input: Matlab `randn` returns a n x n matrix while Scilab `rand(...,"normal")` returns a scalar.

The function `mtlb_randn(A)` is used by `mfile2sci` to replace `randn(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_randn` calls:

- If `A` is a scalar `mtlb_randn(A)` may be replaced by `rand(A,A,"normal")`
- If `A` is not a scalar `mtlb_randn(A)` may be replaced by `rand(A,"normal")`

Caution: `mtlb_randn` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_rcond — Matlab rcond emulation function

Description

Matlab and Scilab `rcond` behave differently in some particular cases:

- With empty matrix: Matlab `rcond` returns `Inf` and Scilab `rcond` returns `[]`

The function `mtlb_rcond(A)` is used by `mfile2sci` to replace `rcond(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_rcond` calls:

- If `A` is not an empty matrix, `mtlb_rcond(A)` may be replaced by `rcond(A)`

Caution: `mtlb_rcond` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_realmax — Matlab realmax emulation function

Description

Scilab equivalent for Matlab `realmax` is `number_properties` but not all cases are implemented:

- Scilab equivalent for Matlab `realmax` or `realmax('double')` is `number_properties("huge")`.
- There is no Scilab equivalent for Matlab `realmax('single')`

The function `mtlb_realmax(prec)` is used by `mfile2sci` to replace `realmax(prec)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_realmax` calls:

- If `prec` is equal to `"double"` `mtlb_realmax(prec)` may be replaced by `number_properties("huge")`

Caution: `mtlb_realmax` has not to be used for hand coded functions.

See Also

number_properties

Authors

V.C.

Name

mtlb_realmin — Matlab realmin emulation function

Description

Scilab equivalent for Matlab `realmin` is `number_properties` but not all cases are implemented:

- Scilab equivalent for Matlab `realmin` or `realmin('double')` is `number_properties("tiny")`.
- There is no Scilab equivalent for Matlab `realmin('single')`

The function `mtlb_realmin(prec)` is used by `mfile2sci` to replace `realmin(prec)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_realmin` calls:

- If `prec` is equal to `"double"` `mtlb_realmin(prec)` may be replaced by `number_properties("tiny")`

Caution: `mtlb_realmin` has not to be used for hand coded functions.

See Also

number_properties

Authors

V.C.

Name

mtlb_repmat — Matlab repmat emulation function

Description

There is no Scilab equivalent function for Matlab `repmat` but there are equivalent instructions.

The function `mtlb_repmat(M,m[,n])` is used by `mfile2sci` to replace `repmat(M,m[,n])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_repmat` calls:

- If `A` is of Double type and `m` is a scalar, `mtlb_repmat(M,m)` may be replaced by `ones(m,m).*.M` and `mtlb_repmat(M,m,n)` may be replaced by `ones(m,n).*.M`
- If `A` is of Boolean type and `m` is a scalar, `mtlb_repmat(M,m)` may be replaced by `ones(m,m).*.bool2s(M)` and `mtlb_repmat(M,m,n)` may be replaced by `ones(m,n).*.bool2s(M)`
- If `A` is of String type and `m` is a scalar, `mtlb_repmat(M,m)` may be replaced by `asciimat(ones(m,m).*.asciimat(M))` and `mtlb_repmat(M,m,n)` may be replaced by `asciimat(ones(m,n).*.asciimat(M))`
- If `A` is of Double type and `m` is a vector, `mtlb_repmat(M,m)` may be replaced by `ones(m(1),m(2),...).*.M`
- If `A` is of Boolean type and `m` is a vector, `mtlb_repmat(M,m)` may be replaced by `ones(m(1),m(2),...).*.bool2s(M)`
- If `A` is of String type and `m` is a vector, `mtlb_repmat(M,m)` may be replaced by `asciimat(ones(m(1),m(2),...).*.asciimat(M))`

Caution: `mtlb_repmat` has not to be used for hand coded functions.

See Also

ones , kron

Authors

V.C.

Name

mtlb_s — Matlab substraction emulation function

Description

Matlab and Scilab substraction behave differently in some particular cases:

- With character string operands: The `-` operator can not be used into Scilab with character strings, while Matlab realizes the substraction of the operands ASCII codes.
- With empty matrix: In Scilab, if one of the operands is an empty matrix the result of the substraction is the other operand. In Matlab if one of the operand is an empty matrix the result of the substraction should be an error or an empty matrix.

The function `mtlb_s(A,B)` is used by `mfile2sci` to replace `A-B` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_s` calls:

- If `A` and `B` are character strings, `mtlb_s(A,B)` may be replaced by `asciimat(A)-asciimat(B)`
- If both `A` and `B` are not empty matrices `mtlb_s(A,B)` may be replaced by `A-B`, else `mtlb_s(A,B)` may be replaced by `[]`.
- If mtlb_mode()==%T, then `mtlb_s(A,B)` may be replaced by `A-B` in any case where `A` and `B` are not character string matrices.

Caution: `mtlb_s` has not to be used for hand coded functions.

See Also

mtlb_mode

Authors

V.C.

Name

mtlb_setstr — Matlab setstr emulation function

Description

Matlab `setstr` and Scilab `ascii` behave differently in some particular cases:

- With character string input: Matlab `setstr` returns a character string while Scilab `ascii` returns ASCII codes.
- With double matrix input: Matlab `setstr` returns a character matrix having the same size as input while Scilab `ascii` returns a single character string

The function `mtlb_setstr(A)` is used by `mfile2sci` to replace `setstr(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_setstr` calls:

- If `A` is a character string or a character string matrix `mtlb_setstr(A)` may be replaced by `A`
- If `A` is a double row vector `mtlb_setstr(A)` may be replaced by `ascii(A)`

Caution: `mtlb_setstr` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_size — Matlab size emulation function

Description

Matlab and Scilab `size` behave differently in some particular cases:

- With two inputs: Matlab `size` can be used with second parameter giving a not-existing dimension of first parameter (returns 1 in this case) but not Scilab one.
- With more than one output: if number of output is lesser than number of dimensions, last output is the product of all remaining dimensions in Matlab but not in Scilab. If number of output is greater than number of dimensions, outputs corresponding to a not-existing dimension are set to 1 in Matlab but Scilab gives an error in this case.

The function `[d1,[d2,...]] = mtlb_size(X[,dim])` is used by `mfile2sci` to replace `[d1,[d2,...]] = size(X[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_size` calls:

- With two inputs: if `dim` gives an existing dimension of `X` `mtlb_size(X,dim)` may be replaced by `size(X,dim)`
- With more than one outputs: if the number of outputs is equal to the number of dimensions of `X` `[d1,[d2,...]] = mtlb_size(X)` may be replaced by `[d1,[d2,...]] = size(X)`

Caution: `mtlb_size` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_sort — Matlab sort emulation function

```
P = mtlb_sort(X)
P = mtlb_sort(X,dim[,mode])
```

Parameters

X
: a scalar, vector, matrix of reals, booleans or a string

dim
: a positive scalar integer

mode
: a string ("ascend" or "descend")

Description

Matlab `sort` and Scilab `gsort` behave differently in some particular cases:

- For a vector `X` the Matlab `sort(X,'g','i')` function call is equivalent to the Scilab `gsort(X)` function call.
- The value `1` (resp. `2`) of the Matlab dim is equivalent to the Scilab `"r"` flag (resp. `"c"`).
- The Matlab "`ascend`" (resp. `"descend"`) mode is equivalent to the Scilab `"i"` (resp. `"d"`) flag.

The function `mtlb_sort(X[,dim[,mode]])` is used by `mfile2sci` to replace `sort(X[,dim[,mode]])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_sort` call by gsort call.

Caution: `mtlb_sort` has not to be used for hand coded functions.

Authors

F.B.

Name

mtlb_strcmp — Matlab strcmp emulation function

Description

There is no Scilab function equivalent for Matlab `strcmp`, there is equivalent instructions.

The function `mtlb_strcmp(A,B)` is used by `mfile2sci` to replace `strcmp(A,B)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_strcmp` calls:

- If `A` and `B` are character strings `mtlb_strcmp(A,B)` may be replaced by `A==B`
- If `A` and/or `B` is not a character string `mtlb_strcmp(A,B)` may be replaced by `0`

Caution: `mtlb_strcmp` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_strcmpi — Matlab strcmpi emulation function

Description

There is no Scilab function equivalent for Matlab `strcmpi`, there is equivalent instructions.

The function `mtlb_strcmpi(A,B)` is used by `mfile2sci` to replace `strcmpi(A,B)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_strcmpi` calls:

- If `A` and `B` are character strings `mtlb_strcmpi(A,B)` may be replaced by `convstr(A)==convstr(B)`
- If `A` and/or `B` is not a character string `mtlb_strcmpi(A,B)` may be replaced by `0`

Caution: `mtlb_strcmpi` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_strfind — Matlab strfind emulation function

Description

Matlab `strfind` and Scilab `strindex` behave differently in some particular cases:

- With inputs which not character strings: Matlab `strfind` can be used with not character strings inputs but not Scilab `strindex`.

The function `mtlb_strfind(A,B)` is used by `mfile2sci` to replace `strfind(A,B)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_strfind` calls:

- If `A` and `B` are character strings `mtlb_strfind(A,B)` may be replaced by `strindex(A,B)`

Caution: `mtlb_strfind` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_strrep — Matlab strrep emulation function

Description

Matlab `strrep` and Scilab `strsubst` behave differently in some particular cases:

- With inputs which not character strings: Matlab `strrep` can be used with not character strings inputs but not Scilab `strsubst`.

The function `mtlb_strrep(A,B,C)` is used by `mfile2sci` to replace `strrep(A,B,C)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_strrep` calls:

- If `A`, `B` and `C` are character strings `mtlb_strrep(A,B,C)` may be replaced by `strsubst(A,B,C)`

Caution: `mtlb_strrep` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_sum — Matlab sum emulation function

Description

Matlab and Scilab `sum` behave differently in some particular cases:

- When called with one input: Matlab `sum` threts the values along the first non-singleton dimension of input while Scilab `sum` threats all values of input.
- When called with two inputs: Matlab `sum` can be used with second input giving a non-existing dimension of first input while Scilab `sum` returns an error message.

The function `mtlb_sum(A[,dim])` is used by `mfile2sci` to replace `sum(A[,dim])` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_sum` calls:

- When called with one input, if `A` is an empty matrix, a scalar or a vector, `mtlb_sum(A)` may be replaced by `sum(A)`
- When called with one input, if `A` is a not-empty matrix, `mtlb_sum(A)` may be replaced by `sum(A,1)`
- When called with one input, if `A` is a multidimensional array, `mtlb_sum(A)` may be replaced by `sum(A,firstnonsingleton(A))`
- When called with two inputs, if `dim` is lesser than the number of dimensions of `A` `mtlb_sum(A,dim)` may be replaced by `sum(A,dim)`

Caution: `mtlb_sum` has not to be used for hand coded functions.

See Also

firstnonsingleton

Authors

V.C.

Name

mtlb_t — Matlab transposition emulation function

Description

Matlab and Scilab transposition behave differently in some particular cases:

- With character strings operands: The ' operator is used to transpose whole character strings in Scilab while Matlab realizes the transposition of each character.

The function `mtlb_t(A)` is used by `mfile2sci` to replace `A'` when it was not possible to know what were the operands while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_t` calls:

- If `A` is not a character string matrix `mtlb_t(A)` may be replaced by `A'`

Caution: `mtlb_t` has not to be used for hand coded functions.

See Also

Matlab-Scilab_character_strings

Authors

V.C.

Name

mtlb_toeplitz — Matlab toeplitz emulation function

Description

Matlab and Scilab `toeplitz` behave differently in some particular cases:

- With one input parameter: if this parameter is complex or is a matrix, output value of Matlab and Scilab `toeplitz` can be different.
- With two input parameters: if they are vectors and their first elements are not equal, Scilab returns an error but Matlab gives priority to the column element. If they are matrices, output value of Matlab and Scilab `toeplitz` are different.

The function `mtlb_toeplitz(c[,r])` is used by `mfile2sci` to replace `toeplitz(c[,r])` when it was not possible to know what were the operands/inputs[CUSTOM] while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_toeplitz` calls:

- When called with one input, if `c` is a real scalar or vector `mtlb_toeplitz(c)` may be replaced by `toeplitz(c)`
- When called with two inputs, if `c` and `r` are scalars or vectors and their first elements are equal `mtlb_toeplitz(c,r)` may be replaced by `toeplitz(c,r)`

Caution: `mtlb_toeplitz` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_tril — Matlab tril emulation function

Description

Matlab and Scilab `tril` behave differently in some particular cases:

- With complex input: Matlab `tril` can be used with complex data but not Scilab one.
- With character strings inputs: due to the fact the Matlab and Scilab do not consider character strings in the same way, Scilab and Matlab `tril` do not give the same results for this type of input.
- With boolean inputs: Matlab `tril` can be used with boobean data but not Scilab one.

The function `mtlb_tril(x,k)` is used by `mfile2sci` to replace `tril(x,k)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_tril` calls:

- If `X` contains real double values `mtlb_tril(x,k)` may be replaced by `tril(x,k)`
- If `X` contains boolean values `mtlb_tril(x,k)` may be replaced by `tril(bool2s(x),k)`

Caution: `mtlb_tril` has not to be used for hand coded functions.

See Also

Matlab-Scilab_character_strings

Authors

V.C.

Name

mtlb_triu — Matlab triu emulation function

Description

Matlab and Scilab `triu` behave differently in some particular cases:

- With complex input: Matlab `triu` can be used with complex data but not Scilab one.
- With character strings inputs: due to the fact the Matlab and Scilab do not consider character strings in the same way, Scilab and Matlab `triu` do not give the same results for this type of input.
- With boolean inputs: Matlab `triu` can be used with boobean data but not Scilab one.

The function `mtlb_triu(x,k)` is used by `mfile2sci` to replace `triu(x,k)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_triu` calls:

- If `X` contains real double values `mtlb_triu(x,k)` may be replaced by `triu(x,k)`
- If `X` contains boolean values `mtlb_triu(x,k)` may be replaced by `triu(bool2s(x),k)`

Caution: `mtlb_triu` has not to be used for hand coded functions.

See Also

Matlab-Scilab_character_strings

Authors

V.C.

Name

mtlb_true — Matlab true emulation function

Description

There is no Scilab equivalent for Matlab `true`. However, Scilab `ones` returns a result interpreted in an equivalent way for Scilab.

Matlab `true` and Scilab `ones` behave differently in some particular cases:

- With a scalar input: Matlab `true` returns a n x n matrix of ones while Scilab `ones` returns a 1.

The function `mtlb_true(A)` is used by `mfile2sci` to replace `true(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_true` calls:

- If `A` is a scalar `mtlb_true(A)` may be replaced by `ones(A,A)`
- If `A` is not a scalar `mtlb_true(A)` may be replaced by `ones(A)`

Caution: `mtlb_true` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_uint16 — Matlab uint16 emulation function

Description

Matlab and Scilab `uint16` behave differently in some particular cases:

- With complex input: Matlab `uint16` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `uint16` returns 65535 and Scilab returns 0.

The function `mtlb_uint16(A)` is used by `mfile2sci` to replace `uint16(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_uint16` calls:

- If `A` does not contain `%inf` values `mtlb_uint16(A)` may be replaced by `uint16(A)`

Caution: `mtlb_uint16` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_uint32 — Matlab uint32 emulation function

Description

Matlab and Scilab `uint32` behave differently in some particular cases:

- With complex input: Matlab `uint32` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `uint32` returns 4294967295 and Scilab returns 0.

The function `mtlb_uint32(A)` is used by `mfile2sci` to replace `uint32(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_uint32` calls:

- If `A` does not contain `%inf` values `mtlb_uint32(A)` may be replaced by `uint32(A)`

Caution: `mtlb_uint32` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_uint8 — Matlab uint8 emulation function

Description

Matlab and Scilab `uint8` behave differently in some particular cases:

- With complex input: Matlab `uint8` can be used with complex values what Scilab function can not.
- With `%inf`: Matlab `uint8` returns 255 and Scilab returns 0.

The function `mtlb_uint8(A)` is used by `mfile2sci` to replace `uint8(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_uint8` calls:

- If `A` does not contain `%inf` values `mtlb_uint8(A)` may be replaced by `uint8(A)`

Caution: `mtlb_uint8` has not to be used for hand coded functions.

Authors

V.C.

Name

mtlb_upper — Matlab upper emulation function

Description

Matlab `upper(A)` and Scilab `convstr(A,"u")` behave differently in some particular cases:

- If `A` is not a chacarter string matrix: Matlab `upper` can be used with a not-character-string `A` but not Scilab `convstr`.

The function `mtlb_upper(A)` is used by `mfile2sci` to replace `upper(A)` when it was not possible to know what were the inputs while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_upper` calls:

- If `A` is a character string matrix `mtlb_upper(A)` may be replaced by `convstr(A,"u")`
- If `A` is not a character string matrix `mtlb_upper(A)` may be replaced by `A`

Caution: `mtlb_upper` has not to be used for hand coded functions.

See Also

convstr

Authors

V.C.

Name

mtlb_var — Matlab var emulation function

Parameters

x
: a real or a complex vector or matrix.

s
: a real scalar or real vector.

  - If x is a vector, s is the variance of x.
  - If x is a matrix, s is a row vector containing the variance of each column of x.

w
: type of normalization to use. Valid values are, depending on the number of columns m of x :

  - w = 0 : normalizes with m-1, provides the best unbiased estimator of the variance (this is the default).
  - w = 1: normalizes with m, this provides the second moment around the mean.

dim
: the dimension along which the variance is computed (default is 1, i.e. column by column). If dim is 2, the variance is computed row by row.

Description

This function computes the variance of the values of a vector or matrix x. It provides the same service as Octave and Matlab. It differs from Scilab's variance primitive:

- mtlb_var returns a real (i.e. with a zero imaginary part) variance, even if x is a complex vector or matrix. The Scilab variance primitive returns a complex value if the input vector x is complex and if no option additionnal is used.
- Whatever the type of the input data x (i.e. vector or matrix), mtlb_var computes the variance either on dimension 1 or on dimension 2 while, if no option is passed to the Scilab's variance primitive, the variance is computed on all dimension at once.

Examples

The following 3 examples illustrates the use of the mtlb_var function. In the first case, a column vector is passed to the function, which returns the value 750. In the second case, a matrix is passed to the function, which returns the row vector [0.16 0.09]. In the third case, a complex column vector is passed to the function, which returns a value close to 2.

```
x = [10; 20; 30; 40; 50; 60; 70; 80; 90];
computed = mtlb_var(x);

x = [0.9    0.7
    0.1    0.1
    0.5    0.4];
computed = mtlb_var(x);

N=1000;
```

```
x = grand(N,1,'nor',0,1) + %i*grand(N,1,'nor',0,1);
computed = mtlb_var(x);
```

See Also

variance

Authors

Michael Baudin

Name

mtlb_zeros — Matlab zeros emulation function

Description

Matlab and Scilab `zeros` behave differently in some particular cases:

- With a scalar input: Matlab `zeros` returns a n x n matrix while Scilab returns a 0.

The function `mtlb_zeros(A)` is used by `mfile2sci` to replace `zeros(A)` when it was not possible to know what was the input while porting Matlab code to Scilab. This function will determine the correct semantic at run time. If you want to have a more efficient code it is possible to replace `mtlb_zeros` calls:

- If `A` is a scalar `mtlb_zeros(A)` may be replaced by `zeros(A,A)`
- If `A` is not a scalar `mtlb_zeros(A)` may be replaced by `zeros(A)`

Caution: `mtlb_zeros` has not to be used for hand coded functions.

Authors

V.C.

Completion

Name

completion — returns words that start with the text you pass as parameter.

```
r = completion(beginning_of_a_word)
r = completion(beginning_of_a_word,dictionary)
[functions,commands,variables,macros,graphic_properties,files] = completion(beg
[functions,commands,variables,macros,graphic_properties] = completion(beginning
[functions,commands,variables,macros] = completion(beginning_of_a_word)
[functions,commands,variables] = completion(beginning_of_a_word)
[functions,commands] = completion(beginning_of_a_word)
```

Parameters

r
: a string matrix

beginning_of_a_word
: a string

dictionary
: a string ("functions","commands","variables","macros","graphic_properties","files")

functions,commands,variables,macros,graphic_properties,files
: a string matrix

Description

returns words that start with the text you pass as parameter.

Examples

```
r = completion('w')
r = completion('w','functions')
r = completion('w','commands')
r = completion('w','variables')
r = completion('w','macros')
r = completion('w','graphic_properties')
r = completion('w','files')

[functions,commands,variables,macros,graphic_properties,files] = completion('w'
[functions,commands,variables,macros,graphic_properties] = completion('w')
[functions,commands,variables,macros] = completion('w')
[functions,commands,variables] = completion('w')
[functions,commands] = completion('w')
```

See Also

getscilabkeywords

Console

Name

console — Keyboard Shortcuts in the Console Window

Description

UP or Ctrl+P	recall previous line.
DOWN or Ctrl+N	recall next line.
F1	call help.
F2	clear console.
F12	open console box only on Windows.
Ctrl+space or TAB	completion : scilab displays a list of all names that start with somes characters.
Ctrl + A or HOME	move to beginning of current line.
Ctrl + B or LEFT	moves the cursor one character to the left.
Ctrl + C	interrupts Scilab if nothing selected in the console, else text selected is sent to clipboard.
Ctrl + D or DELETE	deletes the current character.
Ctrl + E or END	moves the cursor to the end of command line.
Ctrl + F or RIGHT	moves the cursor one character to the right.
Ctrl + H or BACKSPACE	deletes the previous character.
Ctrl + K	kills command line from cursor to the end.
Ctrl + S	select all.
Ctrl + U	delete the whole command line.
Ctrl + V	do a paste from clipboard.
Ctrl + W	delete the last word of the command line.
Ctrl + X	Interrupt Scilab
Ctrl + LEFT	move left one word.
Ctrl + RIGHT	move right one word.
Shift + HOME	select from cursor to beginning of statement.
Shift + END	select from cursor to end of statement.
Double-click	select current word.

Data Structures

Name

cell — Create a cell array of empty matrices.

```
c=cell()
c=cell(m1)
c=cell(m1, m2)
c=cell(m1, m2, ..., mn)
c=cell(x)
```

Parameters

x
: Vector containing the dimensions of the cell to create.

m1, m2,..
: Dimensions of the cell to create.

Description

Returns the create cell of empty matrices.

cell()
: returns a `(0,0)` cell array of empty matrices.

cell(m1)
: returns a `(m1,m1)` cell array of empty matrices.

cell(m1,m2)
: returns a `(m1,m2)` cell array of empty matrices.

cell(m1,m2,..,mn)
: creates a `(m1,m2,..,mn)` cell array of empty matrices.

cell(x)
: returns a cell array of empty matrices with: the first dimension of the cell array is `x(1)`, the second dimension is `x(2), ...`

Remarks

`cell(x)` is not the same size that `x`.

`cell()` is equivalent to `cell(0)`.

If A is a cell array, you can access the contents of an element of A by using `A(m1, m2, ..., mn).entries`, the expression `A(1,1) = zeros(2,2)` is not valid, the right syntax is `A(1,1).entries = zeros(2,2)`.

If A is a cell array, you can get its dimensions by using `A.dims`.

Examples

```
a=cell(3)
b=cell(3,1)
c=cell([2,3,4])
```

```
// Assigning cell entries
b=cell(3,1);
// Assigning the first element of b using the 'entries' field
b(1).entries=1:3
// Assigning the second element of b using the 'entries' field
b(2).entries='Scilab'
// Assigning the third element of b using the 'entries' field
b(3).entries=poly(1:3,'s')

// Assigning sub-cells
X=cell(3,2);
X(:,1)=b

// Extracting a sub-cell: result is a cell
b(1)
b(1:2)

// Extracting a sub-cell value: result is an array
b(1).entries

// Dimensions of b
b.dims
```

See Also

eye, ones, zeros

Name

definedfields — return index of list's defined fields

```
k=definedfields(l)
```

Parameters

l
: a list , tlist or mlist variable.

k
: a vector of index.

Description

If `l` is a `list` `tlist` `mlist` `k=definedfields(l)` returns in `k` the indices of the defined list fields. This function is useful because indexing undefined fields produces an error.

Examples

```
l=list(1);l(3)=5
k=definedfields(l)

t=tlist('x');t(5)=4
definedfields(t)

m=mlist(['m','a','b']);m.b='sdfgfgd'
definedfields(m)
```

See Also

list , tlist , mlist , insertion , extraction

Name

getfield — list field extraction

```
[x,...]=getfield(i,l)
```

Parameters

x
: matrix of any possible types

l
: list, tlist or mlist variable

i
: field index, see extraction for more details.

Description

This function is an equivalent of `[x,...]=l(i)` syntax for field extraction with the only difference that it also applies to `mlist` objects.

Examples

```
l=list(1,'qwerw',%s)
[a,b]=getfield([3 2],l)

a=hypermat([2,2,2],rand(1:2^3));// hypermatrices are coded using mlists
a(1) // the a(1,1,1) entry
getfield(1,a) // the mlist first field
```

See Also

extraction

Name

hypermat — initialize an N dimensional matrices

```
M=hypermat(dims [,v])
```

Parameters

dims
: vector of hypermatrix dimensions

v
: vector of hypermatrix entries (default value `zeros(prod(dims),1)`)

Description

Initialize an hypermatrix whose dimensions are given in the vector dims and entries are given in optional argument `v`

M data structure contains the vector of matrix dimensions `M('dims')` and the vector of entries `M('entries')` such as the leftmost subcripts vary first `[M(1,1,..);..;M(n1,1,..);...;M(1,n2,..);..;M(n1,n2,..);...]`

Examples

```
M=hypermat([2 3 2 2],1:24)
```

Name

hypermatrices — Scilab object, N dimensional matrices in Scilab

Description

Hypermatrix type allows to manipulate multidimensional arrays

They can be defined by extension of 2D matrices as follows a=[1 2;3 4];a(:,:,2)=rand(2,2)

or directly using `hypermat` function

Entries can be real or complex numbers, polynomials, rationals, strings, booleans.

Hypermatrices are `mlists`: `mlist(['hm','dims','entries'],sz,v)` where `sz` is the row vector of dimensions and `v` the column vector of entries (first dimension are stored first)

NOTES: The number of dimension of hypermatrices with right-most sizes equal to 1 are automatically reduced. An hypermatrix with only two dimensions is automatically changed to a regular matrix (type 1).

Examples

```
a(1,1,1,1:2)=[1 2]
a=[1 2;3 4];a(:,:,2)=rand(2,2)
a(1,1,:)

size(a)

a(:,:,1) //dimensionnality reduction
type(a(:,:,1))

[a a]
```

See Also

hypermat

Name

iscell — Check if a variable is a cell array

```
bool = iscell(x)
```

Parameters

x
: Scilab variable

bool
: A boolean

Description

`iscell(x)` returns true if x is a cell array and false otherwise.

Examples

```
iscell(1)

iscell(cell())

c = cell(1,2);
c(1).entries="Scilab";
c(2).entries=datenum();
iscell(c)
```

See Also

cell, isstruct

Author

V.C.

Name

iscellstr — Check if a variable is a cell array of strings

```
bool = iscellstr(x)
```

Parameters

x
: Scilab variable

bool
: A boolean

Description

`iscellstr(x)` returns true if x is a cell array of strings (or an empty cell array) and false otherwise.

Examples

```
iscellstr(1)

iscellstr(cell())

iscellstr(cell(3))

strcell = cell(3,1);
strcell(1).entries="Scilab";
strcell(2).entries="iscellstr";
strcell(3).entries="help";
iscellstr(strcell)
```

See Also

cell, iscell, isstruct

Author

V.C.

Name

isfield — Checks if the given fieldname exists in the structure

```
bool = isfield(s,fieldname)
```

Parameters

s
: A struct array

fieldname
: A matrix of strings

bool
: A matrix of boolean.

Description

This function returns true if the specified structure "s" includes the field "field", regardless of the corresponding value.

Examples

```
s = struct("field_1",123,"field_2",456,"field_4",789)

// Single Fieldname Syntax
isfield( s , "field_1"  )

// Multiple Fieldname Syntax
isfield( s , [ "field_1" "field_2" ; "field_3" "field_4" ] )
```

See Also

struct , getfield , definedfields

Name

isstruct — Check if a variable is a structure array

```
bool = isstruct(x)
```

Parameters

x
: Scilab variable

bool
: A boolean

Description

`isstruct(x)` returns true if x is a struct array and false otherwise.

Examples

```
isstruct(1)

isstruct(cell())

isstruct(struct("name","Scilab", "version", getversion()))

info.name="Scilab";
info.function="isstruct";
info.module="help";
isstruct(info)
```

See Also

struct, iscell

Author

V.C.

Name

list — Scilab object and list function definition

```
list(a1,....an)
```

Description

Creates a `list` with elements `ai`'s which are arbitrary Scilab objects (`matrix, list,...`). Type of `list` objects is 15. `list()` creates the empty `list` (0 element).

Operations on lists

extraction
: `[x,y,z...]=L(v)` where `v` is a vector of indices; `[x,y,z]=L(:)` extracts all the elements.

insertion at index i
: `L(i)=a` (note that it is not an error to use `L(i)=a` with *i > 1 + size(L)* but some list entries are then undefined and their extraction gives raise to an error).

append an element in queue
: `L($+1)=e`.

append an element in head
: `L(0)=e`. (note that after this operation `e` is at index 1, the initial elements being shifted on the right).

deletion
: `L(i)=null()` removes the i-th element of the list `L`.

concatenation of two lists
: `L3 = lstcat(L1,L2)`.

number of elements of a list
you can use either `nb_elm = size(L)` or `nb_elm = length(L)`.

iterations with a list
it is possible to use a list `L` with a for loop: `for e=L,...,end` is a loop with `length(L)` iterations, the loop variable `e` being equal to `L(i)` at the i th iteration.

Remarks

Scilab provides also other kinds of list, the tlist type (typed list) and the mlist type which are useful to define a new data type with operator overloading facilities (hypermatrices which are multi-dimensionnal arrays in scilab are in fact *mlist*).

Matlab *struct* are also available.

Examples

```
l = list(1,["a" "b"])
l(0) = "foo"
l($+1) = "hello"
l(2) = "toto"
l(3) = rand(1,2)
l(3) = null()
lbis = list("gewurtz", "caipirina" ,"debug")
```

```
lter = lstcat(l,lbis)
size(lter) - size(lbis) - size(l)  // must be zero
```

See Also

null , lstcat , tlist , insertion , extraction , size , length

Name

lsslist — Scilab linear state space function definition

```
lsslist()
lsslist(a1,....an)
```

Description

`lsslist(a1,....an)` is a shortcut to `tlist(['lss','A';'B';'C';'X0','dt'], a1,....an)`

Creates a `tlist` with `['lss','A';'B';'C';'X0','dt']` as first entry and `ai`'s as next entries if any. No type nor size checking is done on `ai`'s.

See Also

tlist , syslin

Name

lstcat — list concatenation

```
lc=lstcat(l1,..ln)
```

Parameters

li
: list or any other type of variable

lc
: a list

Description

`lc=lstcat(l1,..ln)` catenates components of `li` lists in a single list. If the `li` are other type of variables they are simply added to the resulting list.

Examples

```
lstcat(list(1,2,3),33,list('foo',%s))
lstcat(1,2,3)
```

See Also

list

Name

mlist — Scilab object, matrix oriented typed list definition.

```
mlist(typ,a1,....an )
```

Parameters

typ
: vector of character strings

ai
: any Scilab object (`matrix, list,string...`).

Description

`mlist` object are very similar to tlist objects. The only difference concerns the extraction and insertion syntax: if `M` is an mlist, for any index `i` which is not a field name, `M(i)` is no more the `i`th field of the list.

The semantic of the extraction and insertion syntax should be given by an overloading functions.

The overloading function for extraction syntax `b=a(i1,...,in)` has the following calling sequence: `b=%<type_of_a>_e_(i1,...,in,a)`

and the syntax `[x1,..,xm]=a(i1,...,in)` has the following calling sequence: `[x1,..,xm]= %<type_of_a>_e_(i1,...,in,a)`

The overloading function associated to the insertion syntax `a(i1,...,in)=b` has the following calling sequence: `a=%<type_of_b>_i_<type_of_a>(i1,...,in,b,a)`.

mlist fields must then be designed by their names. They can also be handled using the `getfield` and `setfield` functions.

Examples

```
M=mlist(['V','name','value'],['a','b';'c' 'd'],[1 2; 3 4]);
//define display
function %V_p(M),disp(M.name+':'+string(M.value)),endfunction

//define extraction operation
function r=%V_e(varargin)
   M=varargin($)
   r=mlist(['V','name','value'],M.name(varargin(1:$-1)),M.value(varargin(1:$-1)
endfunction
M(2,:) // the second row of  M
M.value

//define insertion operations
function M=%V_i_V(varargin)
   M=varargin($)
   N=varargin($-1)
   M.value(varargin(1:$-2))=N.value
   M.name(varargin(1:$-2))=N.name
```

```
endfunction
M(1,1)=M(2,2)

function M=%s_i_V(varargin) //insertion of a regular matrix into a V matrix
   M=varargin($)
   N=varargin($-1)
   M.value(varargin(1:$-2))=N
   M.name(varargin(1:$-2))=emptystr(N)
endfunction
M(1,1)=44



//tlist case
M=tlist(['V','name','value'],['a','b';'c' 'd'],[1 2; 3 4]);
M(2)
M(2)='a'+string([1 2;3 4])


M('name')
```

See Also

tlist , list , overloading , getfield , setfield

Name

rlist — Scilab rational fraction function definition

```
rlist()
rlist(a1,....an)
```

Description

`rlist(a1,....an)` is a shortcut to `tlist(['r','num';'den','dt'], a1,....an)`

Creates a `tlist` with `['r','num';'den','dt']` as first entry and `ai`'s as next entries if any. No type nor size checking is done on `ai`'s.

See Also

tlist , syslin

Name

setfield — list field insertion

```
setfield(i,x,l)
```

Parameters

x
: matrix of any possible types

l
: list, tlist or mlist variable

i
: field index, see insertion for more details.

Description

This function is an equivalent of `l(i)=x` syntax for field extraction with the only difference that it also applies to `mlist` objects.

Examples

```
l=list(1,'qwerw',%s)
l(1)='Changed'
l(0)='Added'
l(6)=['one more';'added']
//

a=hypermat([2,2,2],rand(1:2^3));// hypermatrices are coded using mlists
setfield(3,1:8,a);a // set the field value to 1:8
```

See Also

insertion

Name

struct — create a struct

```
st=struct(field1,value1,field2,value2...)
```

Parameters

field1, field2, ..
: strings represents the fields names

value1, value2, ..
: all data type (double, char, int, ...), represents the fields values

Description

This function returns a struct with the fields names `fields1`, `field2`, ..., and the fields values corresponding `value1`, `value2`, ...

Examples

```
// create a struct date
date=struct('day',25,'month' ,'DEC','year',2006)
//change the month
date.month='AUG';
// change the year
date.year=1973;
//change the day
date.day=19;
// add a new field
date.semaine=32
```

See Also

cell

Name

tlist — Scilab object and typed list definition.

```
tlist(typ,a1,....an )
```

Parameters

typ
: Character string or vector of character strings

ai
: Any Scilab object (`matrix, list,string...`).

Description

Creates a `typed-list` with elements `ai`'s. The `typ` argument specifies the list type. Such `typed-list` allow the user to define new operations working on these object through scilab functions. The only difference between `typed-list` and `list` is the value of the type (16 instead of 15).

`typ(1)` specifies the list type (character string used to define soft coded operations)

if specified `typ(i)` may give the `i+1`th element formal name

Standard Operations on `list` work similarly for `typed-list`:

extraction : `[x,y,z...]=l(v)` where `v` is a vector of indices; `[x,y,z]=l(:)` extracts all the elements.

insertion : `l(i)=a`

deletion : `l(i)=null()` removes the i-th element of the `tlist l`.

display

Moreover if `typ(2:n+1)` are specified, user may point elements by their names

We give below examples where tlist are used.

Linear systems are represented by specific `typed-list` e.g. a linear system `[A,B,C,D]` is represented by the tlist `Sys=tlist(['lss';'A';'B';'C';'D';'X0';'dt'],A,B,C,D,x0,'c')` and this specific list may be created by the function `syslin`.

Sys(2), Sys('A') or Sys.A is the state-matrix and Sys('dt') or Sys.dt is the time domain

A rational matrix `H` is represented by the `typed-list` `H=tlist(['r';'num';'den';'dt'],Num,Den,[])` where `Num` and `Den` are two polynomial matrices and a (e.g. continuous time) linear system with transfer matrix `H` maybe created by `syslin('c',H)`.

H(2), H('num') or H.num is the transfer matrix numerator

See Also

null , percent , syslin , list

Demo Tools

Nom

demo_begin — begin a demonstration

```
demo_begin()
```

Description

The function demo_begin is used to begin a demonstration. It sets the script and the values in mode of non display on the console, save the environment variables in a temporary file and save the width of the console. This function shall be used with the function demo_end.

Voir Aussi

demo_end , demo_run , demo_message

Auteurs

G.H

Name

demo_choose — create a dialog box for the choice of options

```
num = demo_choose(fil)
```

Description

The function demo_choose create a dialog box for the choice of options. It takes as argument the binary file 'fil'. This file is built by a .sce file written like below. It shall contain the variables 'choice', an array of text within bracket (the different options), and 'titl', the title of the dialog box. After that, the .sce file shall save both variables in the binary file in argument. Before the use of demo_choose, the .sce file shall be executed. The function demo_choose returns the number of line chosen in the options array.

Examples

```
exec('SCI/demos/control/pid/pid_ch_2.sce');
[n]=demo_choose('SCI/demos/control/pid/pid_ch_2.bin');
select n
  case 0
    break
  case 1
    mode(1)
  case 2
 mode(-1)
end
```

See Also

x_choose , demo_mdialog

Authors

G.H

Name

demo_compiler — test the presence of a compileur

```
status = demo_compiler()
```

Description

The function demo_compiler asks the computer if it owns a compileur C or not. It returns a boolean indicating wether the compiler exists or not.

Examples

```
select num,
 case 0
  return;
 case 2 then
  st = demo_compiler(); //The compiler will be used
  if (st==%t) then
   mode(0);
   wheel_build_and_load()
 end
 case 1 then // A precomputed value
 x=read(path+'/x.wrt',8,301);
 wheelg=wheelgs;
 show(x);
end
```

See Also

findmsvccompiler

Authors

G.H

Nom

demo_end — completes a demonstration

```
demo_end()
```

Description

The function demo_end() is used to complete a demonstration. It shall be used complementarily with the function demo_begin. It resets the state of the environment as it was before to use the function demo_begin : width of the console and the variables value.

Voir Aussi

demo_begin , demo_run , demo_message

Auteurs

G.H

Name

demo_file_choice — choose and executes an item within a list

```
demo_file_choice(path,ch)
```

Description

The function demo_file_choice choose and executes an item chosen in the 'demolist' variable, that shall be written above. The variable 'demolist' is a text matrix whose first column contains names of items displayed in an options window and whose second column contains the names of the files that will be executed. The title of the options window is 'Choose a demo'. The 'path' variable is the access to the files called in the second column. The 'ch' variable allows to avoid the special cases 'root' and 'anim' that are used in peculiar demonstrations of Scilab. Then you have to enter another word than 'root' or 'ch', 'no' for example. If you choose to cancel the options window, the console is put back to its previous width.

Examples

```
demolist=['n-pendulum','npend/npend_gateway.sce';
   'Wheel simulation','wheel2/wheel_gateway.sce';
   'Bike Simulation','bike/bike_gateway.sce';
   'ODE''S','ode/ode.dem';
   'DAE''S','dae/dae.dem']

demo_file_choice('SCI/demos/simulation/','no');
```

See Also

demo_function_choice

Authors

G.H

Name

demo_function_choice — choose and execute an item within a list

```
demo_function_choice()
```

Description

The function demo_function_choice choose and execute an item chosen in the variable 'demolist' that shall appear above. The variable 'demolist' is a text matrix whose first column contains item names displayed in an options window and whose second column contains the function that will be called. The title of the options window is 'Choose a demo'. If the options window is cancelled, the console is put back to its previous width.

Examples

```
demolist=[
 'Simulation of a binomial random variable','set figure_style new;xbasc();Binom
 'Simulation of a discrete random variable','set figure_style new;xbasc();RndDi
 'Simulation of a geometric random variable','set figure_style new;xbasc();Geom
 'Simulation of a Poisson random variable','set figure_style new;xbasc();Poisso
 'Simulation of an exponential random variable','set figure_style new;xbasc();E
 'Simulation of a Weibull random variable','set figure_style new;xbasc();Weibul
 'Simulation of an hyper geometric random variable','set figure_style new;xbasc
 'Simulation of an Erlang random variable','set figure_style new;xbasc();Erlang

demo_function_choice();
```

See Also

demo_file_choice

Authors

G.H

Name

demo_mdialog — create a dialog box

```
resp = demo_mdialog(fil)
```

Description

The function demo_mdialog create a dialog box. It takes as argument a binary file. This file is built by a .sce file written like below. It shall contain the variables 'titl', the title a the dialog box, 'namevar', the name of the fields to fill, and 'value', the values written by default. After this, these three variables shall be saved in the binary file. The use of demo_mdialog shall be preceded by the execution of the .sce associated. The function demo_mdialog returns 'resp', the values chosen by the user.

Examples

```
exec('SCI/demos/control/pid/pid_dial_4.sce');
[defv]=demo_mdialog('SCI/demos/control/pid/pid_dial_4.bin');

  if defv==[] then warning('Demo stops!');return;end
```

See Also

demo_choose , x_mdialog , x_dialog

Authors

G.H

Name

demo_message — display a message

```
demo_message(fil)
```

Description

The function demo_message displays the text message within the file 'fil' given in argument.

Examples

```
demo_message('SCI/demos/control/pid/pid_3.sce');
```

See Also

demo_run , messagebox , demo_begin , demo_end

Authors

G.H

Name

demo_run — script file execution

```
demo_run(fil)
```

Description

The function demo_run executes a script in the file 'fil' given in argument.

See Also

exec , demo_message , demo_begin , demo_end

Authors

G.H

Development tools

Name

tbx_build_gateway — Build a gateway (toolbox compilation process)

```
tbx_build_gateway(libname, names, files, [gateway_path [, libs [, ldflags
```

Parameters

libname
: a character string, the generic name of the library without path and extension.

names
: 2 column string matrix giving the table of pairs 'scilab-name', 'interface name'

files
: string matrix giving objects files needed for shared library creation

gateway_path
: Path to the sources of the gateway ; in a normal toolbox it should be the directory containing the builder_gateway_(lang).sce script (which should be the script calling this function). Default is current directory.

libs
: string matrix giving extra libraries needed for shared library creation

ldflags,cflags,fflags
: character strings to provide options for the loader, the C compiler and the Fortran compiler.

cc
: character string. The name of or path to the compiler.

makename
: character string. The path of the Makefile file without extension.

Examples

```
// Recommended usage
tbx_build_gateway('mytoolbox', ['c_sum','sci_csum';'c_sub','sci_csub'], ['sci_c
                  get_absolute_file_path('builder_gateway_c.sce'), ..
                  ['../../src/c/libcsum']);
```

See Also

ilib_build

Authors

SL

Name

tbx_build_gateway_loader — Generate a loader_gateway.sce script (toolbox compilation process)

```
tbx_build_gateway_loader(langs)
tbx_build_gateway_loader(langs, gateway_path)
```

Parameters

langs
: Languages of source files.

gateway_path
: Path to the sources of the gateway ; in a normal toolbox it should be the directory containing the builder_gateway.sce script (which should be the script calling this function). Default is current directory.

Examples

```
// Recommended usage
tbx_build_gateway_loader(['c', 'fortran'], get_absolute_file_path('builder_gate
```

Authors

SL

Name

tbx_build_help — Generate help files (toolbox compilation process)

```
tbx_build_help(title)
tbx_build_help(title, help_path)
```

Parameters

title
: Title of the chapter.

help_path
: Directory where the files will be generated ; in a normal toolbox it should be the directory containing the build_help.sce script (which should be the script calling this function). Default is current directory.

Examples

```
// Recommended usage
tbx_build_help("Toolbox Example", get_absolute_file_path('build_help.sce'))
```

Authors

SL

Name

tbx_build_help_loader — Generate a addchapter.sce script (toolbox compilation process)

```
tbx_build_help_loader(title)
tbx_build_help_loader(title, help_path)
```

Parameters

title
: Title of the chapter.

help_path
: Directory where the script will be generated ; in a normal toolbox it should be the directory containing the build_help.sce script (which should be the script calling this function). Default is current directory.

Examples

```
// Recommended usage
tbx_build_help_loader("Toolbox Example", get_absolute_file_path('build_help.sce
```

Authors

SL

Name

tbx_build_loader — Generate a loader.sce script (toolbox compilation process)

```
tbx_build_loader(toolbox_name)
tbx_build_loader(toolbox_name, toolbox_path)
```

Parameters

toolbox_name
: Toolbox short name ; that is, the prefix of the .start file of the toolbox (which shall be in the etc subdirectory).

toolbox_path
: Root directory of toolbox sources ; the script will be generated here (default: current directory).

Examples

```
// Recommended usage
tbx_build_loader("mytoolbox", get_absolute_file_path('builder.sce'))
```

Authors

SL

Name

tbx_build_macros — Compile macros (toolbox compilation process)

```
tbx_build_macros(toolbox_name)
tbx_build_macros(toolbox_name, macros_path)
```

Parameters

toolbox_name
: Toolbox short name ; that is, the prefix of the .start file of the toolbox (which shall be in the etc subdirectory).

macros_path
: Directory where the macros files can be found and where the compiled macros will be placed into ; in a normal toolbox it should be the directory containing the buildmacros.sce script (which should be the script calling this function). Default is current directory.

Examples

```
// Recommended usage
tbx_build_macros("toolbox_example", get_absolute_file_path('buildmacros.sce'))
```

Authors

SL

Name

tbx_build_src — Build sources (toolbox compilation process)

```
tbx_build_src(names, files, flag, [src_path [, libs [, ldflags [, cflags
```

Parameters

names
: a string matrix giving the entry names which are to be linked.

files
: string matrix giving objects files needed for shared library creation

flag
: a string flag ("c" or "f") for C or Fortran entry points.

src_path
: Path to the source files ; in a normal toolbox it should be the directory containing the builder_src_(lang).sce script (which should be the script calling this function). Default is current directory.

libs
: string matrix giving extra libraries needed for shared library creation

ldflags
: optional character string. It can be used to add specific linker options in the generated Makefile. Default value is "

cflags
: optional character string. It can be used to add specific C compiler options in the generated Makefile. Default value is "

fflags
: optional character string. It can be used to add specific Fortran compiler options in the generated Makefile. Default value is "

cc
: optional character string. It can be used to specify a C compiler. Default value is "

libname
: optional character string. The name of the generated shared library (default value is '', and in this case the name is derived from `names(1)`).

loadername
: character string. The pathname of the loader file (default value is `loader.sce`).

makename
: character string. The pathname of the Makefile file without extension (default value `Makelib`).

Examples

```
// Recommended usage
tbx_build_src(['csum','csub'], ['csum.o','csub.o'], 'c', ..
              get_absolute_file_path('builder_c.sce'));
```

See Also

ilib_for_link

Authors

SL

Name

tbx_builder_gateway — Run builder_gateway.sce script if it exists (toolbox compilation process)

```
tbx_builder_gateway(toolbox_path)
```

Parameters

toolbox_path
: Root directory of toolbox sources ; builder_gateway.sce script will be searched in the sci_gateway subdirectory of this directory.

Examples

```
// Recommended usage
tbx_builder_gateway(get_absolute_file_path('builder.sce'))
```

Authors

SL

Name

tbx_builder_gateway_lang — Run builder_gateway_(language).sce script if it exists (toolbox compilation process)

```
tbx_builder_gateway_lang(lang)
tbx_builder_gateway_lang(lang, gw_path)
```

Parameters

lang
: Language of sources files ; the builder_gateway_(lang).sce script will be searched in the subdirectory lang (e.g. fortran) of the gw_path directory.

gw_path
: Path to the sources of the gateway ; in a normal toolbox it should be the directory containing the builder_gateway.sce script (which should be the script calling this function). Default is current directory.

Examples

```
// Recommended usage
tbx_builder_gateway_lang("fortran", get_absolute_file_path('builder_gateway.sce
```

Authors

SL

Name

tbx_builder_help — Run builder_help.sce script if it exists (toolbox compilation process)

```
tbx_builder_help(toolbox_path)
```

Parameters

toolbox_path

Root directory of toolbox sources ; builder_help.sce script will be searched in the help subdirectory of this directory.

Examples

```
// Recommended usage
tbx_builder_help(get_absolute_file_path('builder.sce'))
```

Authors

SL

Name

tbx_builder_help_lang — Run build_help.sce script if it exists (toolbox compilation process)

```
tbx_builder_help_lang(lang)
tbx_builder_help_lang(lang, help_path)
```

Parameters

lang
: Language of help files to use ; the build_help.sce script will be searched in the subdirectory lang (e.g. en_US) of the help_path directory

help_path
: Path to help directory ; in a normal toolbox it should be the directory containing the builder_help.sce script (which should be the script calling this function). Default is current directory.

Examples

```
// Recommended usage
tbx_builder_help_lang("en_US", get_absolute_file_path('builder_help.sce'))
```

Authors

SL

Name

tbx_builder_macros — Run buildmacros.sce script if it exists (toolbox compilation process)

```
tbx_builder_macros(toolbox_path)
```

Parameters

toolbox_path
: Root directory of toolbox sources ; buildmacros.sce script will be searched in the macros subdirectory of this directory.

Examples

```
// Recommended usage
tbx_builder_macros(get_absolute_file_path('builder.sce'))
```

Authors

SL

Name

tbx_builder_src — Run builder_src.sce script if it exists (toolbox compilation process)

```
tbx_builder_src(toolbox_path)
```

Parameters

toolbox_path
: Root directory of toolbox sources ; builder_src.sce script will be searched in the src subdirectory of this directory.

Examples

```
// Recommended usage
tbx_builder_src(get_absolute_file_path('builder.sce'))
```

Authors

SL

Name

tbx_builder_src_lang — Run builder_(language).sce script if it exists (toolbox compilation process)

```
tbx_builder_src_lang(lang)
tbx_builder_src_lang(lang, src_path)
```

Parameters

lang
: Language of sources files ; the builder_(lang).sce script will be searched in the subdirectory lang (e.g. fortran) of the src_path directory.

src_path
: Path to the sources ; in a normal toolbox it should be the directory containing the builder_src.sce script (which should be the script calling this function). Default is current directory.

Examples

```
// Recommended usage
tbx_builder_src_lang("fortran", get_absolute_file_path('builder_src.sce'))
```

Authors

SL

Name

test_run — Launch tests

```
N = test_run()
N = test_run(module[,test_name[,options]])
```

Parameters

module
: a vector of string. It can be the name of a module or the absolute path of a toolbox.

test_name
: a vector of string

options
: a vector of string

- no_check_ref
- no_check_error_output
- create_ref
- list
- help
- mode_nw
- mode_nwni
- nonreg_test
- unit_test
- skip_tests

Examples

```
// Launch all tests
test_run();
test_run([]);
test_run([],[]);

// Test one or several module
test_run('core');
test_run(SCI+'/modules/core');
test_run('core',[]);
test_run(['core','string']);

// Launch one or several test in a specified module
test_run('core',['trycatch','opcode']);

// Options
```

```
test_run([],[],'no_check_ref');
test_run([],[],'create_ref');
test_run([],[],'list');
test_run([],[],'help');
test_run([],[],'nonreg_test');
test_run([],[],'unit_test');

test_run([],[],['no_check_ref','mode_nw']);

// Do not check the error output (std err)
test_run('boolean','bug_2799','no_check_error_output');
```

Authors

PM

Differential Equations

Name

dae — Differential algebraic equations solver

```
 y=dae(initial,t0,t,res)
[y [,hd]]=dae(initial,t0,t [,rtol, [atol]],res [,jac] [,hd])
[y,rd]=dae("root",initial,t0,t,res,ng,surface)
[y ,rd [,hd]]=dae("root",initial,t0,t [,rtol, [atol]],res [,jac], ng, surface [
```

Parameters

initial
: a column vector. It may be equal to `x0` or `[x0;xdot0]`. Where `x0` is the state value at initial time `t0` and `ydot0` is the initial state derivative value or an estimation of it (see below).

t0
: a real number, the initial time.

t
: real scalar or vector. Gives instants for which you want the solution. Note that you can get solution at each dae's step point by setting `%DAEOPTIONS(2)=1`.

rtol
: a real scalar or a column vector of same size as `x0`. `The` relative error tolerance of solution. If `rtol` is a vector the tolerances are specified for each component of `the state`.

atol
: a real scalar or a column vector of same size as `x0`. `The` absolute error tolerance of solution. If `atol` is a vector the tolerances are specified for each component of `the state`.

res
: an external. Computes the value of `g(t,y,ydot)`. It may be

  a Scilab function
  : In this case, Its calling sequence must be `[r,ires]=res(t,x,xdot)` and `res` must return the residue `r=g(t,x,xdot)` and error flag `ires`. `ires = 0` if `res` succeeds to compute `r`, `=-1` if residue is locally not defined for `(t,x,xdot)`, `=-2` if parameters are out of admissible range.

  a list
  : This form of external is used to pass parameters to the function. It must be as follows:

    ```
    list(res,p1,p2,...)
    ```

    where the calling sequence of the function `res` is now

    ```
    r=res(t,y,ydot,p1,p2,...)
    ```

    `res` still returns the residual value as a function of `(t,x,xdot,x1,x2,...)`, and p1,p2,... are function parameters.

  a character string
  : it must refer to the name of a C or fortran routine. Assuming that <r_name> is the given name.

    - `The Fortran calling sequence must be`

      ```
      <r_name>(t,x,xdot,res,ires,rpar,ipar)

      double precision t,x(*),xdot(*),res(*),rpar(*)
      ```

```
integer ires,ipar(*)
```

- The C calling sequence must be

  C2F(<r_name>)(double *t, double *x, double *xdot, double *res, integer *ires, double *rpar, integer *ipar)

where

- `t` is the current time value
- `x` the state array
- `xdot` the array of state derivatives
- res the array of residuals
- `ires` the execution indicator
- `rpar` is the array of floating point parameter values, needed but cannot be set by the `dae` function
- `ipar` is the array of floating integer parameter values, needed but cannot be set by the `dae` function

jac

an external. Computes the value of `dg/dx+cj*dg/dxdot` for a given value of parameter `cj`. It may be

a Scilab function

Its calling sequence must be `r=jac(t,x,xdot,cj)` and the `jac` function must return `r=dg(t,x,xdot)/dy+cj*dg(t,x,xdot)/dxdot` where `cj` is a real scalar

a list

This form of external is used to pass parameters to the function. It must be as follows:

```
list(jac,p1,p2,...)
```

where the calling sequence of the function `jac` is now

```
r=jac(t,x,xdot,p1,p2,...)
```

`jac` still returns `dg/dx+cj*dg/dxdot` as a function of `(t,x,xdot,cj,p1,p2,...)`.

a character string

it must refer to the name of a C or fortran routine. Assuming that <j_name> is the given name,

- The Fortran calling sequence must be

  ```
  <j_name>(t, x, xdot, r, cj, ires, rpar, ipar)

  double precision t, x(*), xdot(*), r(*), ci, rpar(*)

  integer ires, ipar(*)
  ```

- The C calling sequence must be

  C2F(<j_name>)(double *t, double *x, double *xdot, double *r, double *cj,

  integer *ires, double *rpar, integer *ipar)

where t, x, xdot, ires, rpar, ipar have similar definition as above, r is the results array

surface
: an external. Computes the value of the column vector `surface(t,x)` with `ng` components. Each component defines a surface.

  a Scilab function
  : Its calling sequence must be `r=surface(t,x)`, this function must return a vector with `ng` elements.

  a list
  : This form of external is used to pass parameters to the function. It must be as follows:

    ```
    list(surface,p1,p2,...)
    ```

    where the calling sequence of the function `surface` is now

    ```
    r=surface(t,x,p1,p2,...)
    ```

  character string
  : it must refer to the name of a C or fortran routine. Assuming that <s_name> is the given name,

    - The Fortran calling sequence must be

      ```
      <r_name>(nx, t, x, ng, r, rpar, ipar)
      double precision t, x(*), r(*), rpar(*)
      integer nx, ng,ipar(*)
      ```

    - The C calling sequence must be

      C2F(<r_name>)(double *t, double *x, double *xdot, double *r, double *cj,

      integer *ires, double *rpar, integer *ipar)

    where t, x, rpar, ipar have similar definition as above, `ng` is the number of surfaces, `nx` the dimension of the state and r is the results array.

rd
: a vector with two entries `[times num]` `times` is the value of the time at which the surface is crossed, `num` is the number of the crossed surface

hd
: a real vector, an an output it stores the `dae` context. It can be used as an input argument to resume integration (hot restart).

y
: real matrix . If `%DAEOPTIONS(2)=1`, each column is the vector `[t;x(t);xdot(t)]` where `t` is time index for which the solution had been computed. Else `y` is the vector `[x(t);xdot(t)]`.

Description

The `dae` function is a gateway built above the dassl and dasrt function designed for implicit differential equations integration.

```
g(t,x,xdot)=0
x(t0)=x0  and   xdot(t0)=xdot0
```

If `xdot0` is not given in the `initial` argument, the dae function tries to compute it solving g(t,x0,xdot0)=0,

if `xdot0` is given in the `initial` argumente it may be either a compatible derivative satisfying g(t,x0,xdot0)=0 or an approximate value . In the latter case `%DAEOPTIONS(7) must be set to 1.`

Detailed examples using Scilab and C coded externals are given in `modules/differential_equations/tests/unit_tests/dassldasrt.tst`

Examples

```
//Example with Scilab  code
 function [r,ires]=chemres(t,y,yd)
     r(1) = -0.04*y(1) + 1d4*y(2)*y(3) - yd(1);
     r(2) =  0.04*y(1) - 1d4*y(2)*y(3) - 3d7*y(2)*y(2) - yd(2);
     r(3) =       y(1) +     y(2)      + y(3)-1;
     ires =  0;
 endfunction
 function pd=chemjac(x,y,yd,cj)
     pd=[-0.04-cj , 1d4*y(3)               , 1d4*y(2);
          0.04    ,-1d4*y(3)-2*3d7*y(2)-cj ,-1d4*y(2);
          1       , 1                      , 1       ]
 endfunction

x0=[1; 0; 0];
xd0=[-0.04; 0.04; 0];
t=[1.d-5:0.02:.4, 0.41:.1:4, 40, 400, 4000, 40000, 4d5, 4d6, 4d7, 4d8, 4d9, 4d1

y=dae([x0,xd0],0,t,chemres);// returns requested observation time points

%DAEOPTIONS=list([],1,[],[],[],0,0); // ask  dae mesh points to be returned
y=dae([x0,xd0],0,4d10,chemres); // without jacobian
y=dae([x0,xd0],0,4d10,chemres,chemjac); // with jacobian

//example with C code (c compiler needed) ------------------------------------
//-1- create the C codes in TMPDIR - Vanderpol equation, implicit form
code=['#include <math.h>'
      'void res22(double *t,double *y,double *yd,double *res,int *ires,double *
      '{res[0] = yd[0] - y[1];'
      ' res[1] = yd[1] - (100.0*(1.0 - y[0]*y[0])*y[1] - y[0]);}'
      ' '
      'void jac22(double *t,double *y,double *yd,double *pd,double *cj,double *
      '{pd[0]=*cj - 0.0;'
      ' pd[1]=    - (-200.0*y[0]*y[1] - 1.0);'
      ' pd[2]=    - 1.0;'
      ' pd[3]=*cj - (100.0*(1.0 - y[0]*y[0]));}'
      ' '
      'void gr22(int *neq, double *t, double *y, int *ng, double *groot, double
      '{ groot[0] = y[0];}']
mputl(code,TMPDIR+'/t22.c')
//-2- compile and load them
ilib_for_link(['res22' 'jac22' 'gr22'],'t22.o',[],'c',TMPDIR+'/Makefile',TMPDIR+
exec(TMPDIR+'/t22loader.sce')
//-3- run
rtol=[1.d-6;1.d-6];atol=[1.d-6;1.d-4];
t0=0;y0=[2;0];y0d=[0;-2];t=[20:20:200];ng=1;
//simple simulation
```

```
t=0:0.003:300;
yy=dae([y0,y0d],t0,t,atol,rtol,'res22','jac22');
clf();plot(yy(1,:),yy(2,:))
//find first point where yy(1)=0
[yy,nn,hotd]=dae("root",[y0,y0d],t0,300,atol,rtol,'res22','jac22',ng,'gr22');
plot(yy(1,1),yy(2,1),'r+')
xstring(yy(1,1)+0.1,yy(2,1),string(nn(1)))

//hot restart for next point
t01=nn(1);[pp,qq]=size(yy);y01=yy(2:3,qq);y0d1=yy(3:4,qq);
[yy,nn,hotd]=dae("root",[y01,y0d1],t01,300,atol,rtol,'res22','jac22',ng,'gr22',
plot(yy(1,1),yy(2,1),'r+')
xstring(yy(1,1)+0.1,yy(2,1),string(nn(1)))
```

See Also

ode, daeoptions, dassl, impl, fort, link, external

Name

daeoptions — set options for dae solver

```
daeoptions()
```

Description

If it exists in the dae function calling context the variable `%DAEOPTIONS` the dae function use it to sets options.

This daeoptions function interactively displays a command which should be executed to set various options of the dae solver.

CAUTION: the `dae` function checks if this variable exists and in this case it uses it. For using default values you should clear this variable. Note that `daeoptions` does not create this variable. To create it you must execute the command line displayed by `daeoptions`.

The variable `%DAEOPTIONS` is a list with the following elements:

```
list(tstop,imode,band,maxstep,stepin,nonneg,isest)
```

The default value is:

```
list([],0,[],[],[],0,0)
```

The meaning of the elements is described below.

tstop
: a real scalar or an empty matrix, gives the maximum time for which `g` is allowed to be evaluated. An empty matrix means "no limits" imposed for time.

imode
: if it is 0 dae returns only the user specified time point values if it is 1 dae returns its intermediate computed values.

band
: a two components vector which give the definition `[ml,mu]` of band matrix computed by `jac` ;

  `r(i - j + ml + mu + 1,j)` = `dg(i)/dy(j)+cj*dg(i)/dydot(j)` . If `jac` returns a full matrix set `band=[]`

maxstep
: A scalar or an empty matrix, the maximum step size, empty matrix means "no limitation".

stepin
: A scalar or an empty matrix, the minimum step size, empty matrix means "not specified".

nonneg
: A scalar, must be set to 0 if `the solution is known to be non negative.` In the other case it must be set to 1.

isest
: A scalar, must be set to 0 is the given initial condition is compatible: `g(t0,x0,xdot0)=0`. 1 an set to 1 if `xdot0` is just an estimation.

See Also

dae

Name

dasrt — DAE solver with zero crossing

```
[r,nn,[,hd]]=dasrt(x0,t0,t [,atol,[rtol]],res [,jac],ng, surf [,info] [,hd])
```

Parameters

x0
: is either `y0` (`ydot0` is estimated by `dassl` with zero as first estimate) or the matrix `[y0 ydot0]`. `g(t,y0,ydot0)` must be equal to zero. If you only know an estimate of `ydot0` set `info(7)=1`

  y0
  : real column vector of initial conditions.

  ydot0
  : real column vector of the time derivative of `y` at `t0` (may be an estimate).

t0
: real number is the initial instant.

t
: real scalar or vector. Gives instants for which you want the solution. Note that you can get solution at each dassl's step point by setting `info(2)=1`.

nn
: a vector with two entries `[times num]` `times` is the value of the time at which the surface is crossed, `num` is the number of the crossed surface

atol,rtol
: real scalars or column vectors of same size as `y`. `atol,rtol` give respectively absolute and relative error tolerances of solution. If vectors the tolerances are specified for each component of `y`.

res
: external (function or list or string). Computes the value of `g(t,y,ydot)`.It may be :

  - A Scilab function.

    Its calling sequence must be `[r,ires]=res(t,y,ydot)` and `res` must return the residue `r=g(t,y,ydot)` and error flag `ires`. `ires = 0` if `res` succeeds to compute `r`, `=-1` if residue is locally not defined for `(t,y,ydot)`, `=-2` if parameters are out of admissible range.

  - A list.

    This form allows to pass parameters other than t,y,ydot to the function. It must be as follows:

    ```
    list(res,x1,x2,...)
    ```

    where the calling sequence of the function `res` is now

    ```
    r=res(t,y,ydot,x1,x2,...)
    ```

    `res` still returns `r=g(t,y,ydot)` as a function of `(t,y,ydot,x1,x2,...)`.

    Warning: this form must not be used if there is no extra argument to pass to `the function.`

  - A string.

    it must refer to the name of a C or fortran subroutine linked with Scilab.

In C The calling sequence must be:

```
void res(double *t, double y[], double yd[], double r[],
                                          int *ires, double rpar[], int ipa
```

In Fortran it must be:

```
subroutine res(t,y,yd,r,ires,rpar,ipar)
double precision t, y(*),yd(*),r(*),rpar(*)
integer ires,ipar(*)
```

The rpar and ipar arrays must be present but cannot be used.

jac

external (function or list or string). Computes the value of `dg/dy+cj*dg/dydot` for a given value of parameter `cj`

- A Scilab function.

  Its calling sequence must be `r=jac(t,y,ydot,cj)` and the `jac` function must return `r=dg(t,y,ydot)/dy+cj*dg(t,y,ydot)/dydot` where `cj` is a real scalar

- A list.

  it must be as follows

  ```
  list(jac,x1,x2,...)
  ```

  where the calling sequence of the function `jac` is now

  ```
  r=jac(t,y,ydot,cj,x1,x2,...)
  ```

  `jac` still returns `dg/dy+cj*dg/dydot` as a function of `(t,y,ydot,cj,x1,x2,...)`.

- A character string.

  it must refer to the name of a fortran subroutine linked with scilab

  In C The calling sequence must be:

  ```
  void jac(double *t, double y[], double yd[], double
              pd[], double *cj, double rpar[], int ipar[])
  ```

  In Fortran it must be:

  ```
  subroutine jac(t,y,yd,pd,cj,rpar,ipar)
  double precision t, y(*),yd(*),pd(*),cj,rpar(*)
  integer ipar(*)
  ```

surf

external (function or list or string). Computes the value of the column vector `surf(t,y)` with `ng` components. Each component defines a surface. It may be defined by:

- A Scilab function.

  Its calling sequence must be `surf(t,y)`

- A list.

it must be as follows

```
list(surf,x1,x2,...)
```

where the calling sequence of the function `surf` is now

```
r=surf(t,y,x1,x2,...)
```

- A character string.

  it must refer to the name of a fortran subroutine linked with scilab

  In C The calling sequence must be:

  ```
  void surf(int *ny, double *t, double y[], int *ng, double gout[])
  ```

  In Fortran it must be:

  ```
  subroutine surf(ny,t,y,ng,gout)
  double precision t, y(*),gout(*)
  integer ny,ng
  ```

info
: list which contains 7 elements, default value is list([],0,[],[],[],0,0)

  info(1)
  : real scalar which gives the maximum time for which `g` is allowed to be evaluated or an empty matrix `[]` if no limits imposed for time.

  info(2)
  : flag which indicates if `dassl` returns its intermediate computed values (`flag=1`) or only the user specified time point values (`flag=0`).

  info(3)
  : 2 components vector which give the definition `[ml,mu]` of band matrix computed by `jac`; `r(i - j + ml + mu + 1,j) = "dg(i)/dy(j)+cj*dg(i)/dydot(j)"`. If `jac` returns a full matrix set `info(3)=[]`.

  info(4)
  : real scalar which gives the maximum step size. Set `info(4)=[]` if no limitation.

  info(5)
  : real scalar which gives the initial step size. Set `info(4)=[]` if not specified.

  info(6)
  : set `info(6)=1` if the solution is known to be non negative, else set `info(6)=0`.

  info(7)
  : set `info(7)=1` if `ydot0` is just an estimation, `info(7)=0` if `g(t0,y0,ydot0)=0`.

hd
: real vector which allows to store the `dassl` context and to resume integration

r
: real matrix . Each column is the vector [t;x(t);xdot(t)] where t is time index for which the solution had been computed

Description

Solution of the implicit differential equation

```
g(t,y,ydot)=0
y(t0)=y0  and   ydot(t0)=ydot0
```

Returns the surface crossing instants and the number of the surface reached in nn.

Detailed examples can be found in SCIDIR/tests/dassldasrt.tst

Examples

```
//dy/dt = ((2*log(y)+8)/t -5)*y,  y(1) = 1,  1<=t<=6
//g1 = ((2*log(y)+8)/t - 5)*y
//g2 = log(y) - 2.2491
y0=1;t=2:6;t0=1;y0d=3;
atol=1.d-6;rtol=0;ng=2;

deff('[delta,ires]=res1(t,y,ydot)','ires=0;delta=ydot-((2*log(y)+8)/t-5)*y')
deff('[rts]=gr1(t,y)','rts=[((2*log(y)+8)/t-5)*y;log(y)-2.2491]')

[yy,nn]=dasrt([y0,y0d],t0,t,atol,rtol,res1,ng,gr1);
//(Should return nn=[2.4698972 2])
```

See Also

ode, dassl, impl, fort, link, external

Name

dassl — differential algebraic equation

```
[r [,hd]]=dassl(x0,t0,t [,atol,[rtol]],res [,jac] [,info] [,hd])
```

Parameters

x0
: is either `y0` (`ydot0` is estimated by `dassl` with zero as first estimate) or the matrix `[y0 ydot0]`. `g(t,y0,ydot0)` must be equal to zero. If you only know an estimate of `ydot0` set `info(7)=1`

  y0
  : real column vector of initial conditions.

  ydot0
  : real column vector of the time derivative of `y` at `t0` (may be an estimate).

t0
: real number is the initial instant.

t
: real scalar or vector. Gives instants for which you want the solution. Note that you can get solution at each dassl's step point by setting `info(2)=1`.

atol,rtol
: real scalars or column vectors of same size as `y`. `atol,rtol` give respectively absolute and relative error tolerances of solution. If vectors the tolerances are specified for each component of `y`.

res
: external (function or list or string). Computes the value of `g(t,y,ydot)`. It may be :

  - A Scilab function.

    Its calling sequence must be `[r,ires]=res(t,y,ydot)` and `res` must return the residue `r=g(t,y,ydot)` and error flag `ires`. `ires = 0` if `res` succeeds to compute `r`, `=-1` if residue is locally not defined for `(t,y,ydot)`, `=-2` if parameters are out of admissible range.

  - A list.

    This form allows to pass parameters other than t,y,ydot to the function. It must be as follows:

    ```
    list(res,x1,x2,...)
    ```

    where the calling sequence of the function `res` is now

    ```
    r=res(t,y,ydot,x1,x2,...)
    ```

    `res` still returns `r=g(t,y,ydot)` as a function of `(t,y,ydot,x1,x2,...)`.

  - A string.

    it must refer to the name of a C or fortran subroutine linked with Scilab.

    In C The calling sequence must be:

    ```
    void res(double *t, double y[], double yd[], double r[],
                                                int *ires, double rpar[], int ipa
    ```

In Fortran it must be:

```
subroutine res(t,y,yd,r,ires,rpar,ipar)
double precision t, y(*),yd(*),r(*),rpar(*)
integer ires,ipar(*)
```

The rpar and ipar arrays must be present but cannot be used.

jac

external (function or list or string). Computes the value of `dg/dy+cj*dg/dydot` for a given value of parameter `cj`

- A Scilab function.

  Its calling sequence must be `r=jac(t,y,ydot,cj)` and the `jac` function must return `r=dg(t,y,ydot)/dy+cj*dg(t,y,ydot)/dydot` where `cj` is a real scalar

- A list.

  it must be as follows

  ```
  list(jac,x1,x2,...)
  ```

  where the calling sequence of the function `jac` is now

  ```
  r=jac(t,y,ydot,cj,x1,x2,...)
  ```

  `jac` still returns `dg/dy+cj*dg/dydot` as a function of `(t,y,ydot,cj,x1,x2,...)`.

- A character string.

  it must refer to the name of a fortran subroutine linked with scilab

  In C The calling sequence must be:

  ```
  void jac(double *t, double y[], double yd[], double
              pd[], double *cj, double rpar[], int ipar[])
  ```

  In Fortran it must be:

  ```
  subroutine jac(t,y,yd,pd,cj,rpar,ipar)
  double precision t, y(*),yd(*),pd(*),cj,rpar(*)
  integer ipar(*)
  ```

info

optional list which contains 7 elements. Default value is list([],0,[],[],[],0,0);

info(1)

real scalar which gives the maximum time for which `g` is allowed to be evaluated or an empty matrix `[]` if no limits imposed for time.

info(2)

flag which indicates if `dassl` returns its intermediate computed values (`flag=1`) or only the user specified time point values (`flag=0`).

info(3)

: 2 components vector which give the definition `[ml,mu]` of band matrix computed by `jac`; `r(i - j + ml + mu + 1,j) = "dg(i)/dy(j)+cj*dg(i)/dydot(j)"`. If `jac` returns a full matrix set `info(3)=[]`.

info(4)

real scalar which gives the maximum step size. Set `info(4)=[]` if no limitation.

info(5)
real scalar which gives the initial step size. Set `info(4)=[]` if not specified.

info(6)
set `info(6)=1` if the solution is known to be non negative, else set .

info(7)
set `info(7)=1` if `ydot0` is just an estimation, `info(7)=0` if `g(t0,y0,ydot0)=0`.

hd
real vector which allows to store the `dassl` context and to resume integration

r
real matrix . Each column is the vector [t;x(t);xdot(t)] where t is time index for which the solution had been computed

Description

The dassl function integrate the algebro-differencial equation and returns the evolution of `y` a given time points

```
    g(t,y,ydot)=0
    y(t0)=y0  and   ydot(t0)=ydot0
```

Examples

```
  function [r,ires]=chemres(t,y,yd)
     r=[-0.04*y(1)+1d4*y(2)*y(3)-yd(1)
         0.04*y(1)-1d4*y(2)*y(3)-3d7*y(2)*y(2)-yd(2)
         y(1)+y(2)+y(3)-1];
     ires=0
  endfunction
  function pd=chemjac(x,y,yd,cj)
      pd=[-0.04-cj , 1d4*y(3)               , 1d4*y(2);
           0.04    ,-1d4*y(3)-2*3d7*y(2)-cj ,-1d4*y(2);
           1       , 1                      , 1       ]
  endfunction

y0=[1;0;0];
yd0=[-0.04;0.04;0];
t=[1.d-5:0.02:.4,0.41:.1:4,40,400,4000,40000,4d5,4d6,4d7,4d8,4d9,4d10];


y=dassl([y0,yd0],0,t,chemres);

info=list([],0,[],[],[],0,0);
info(2)=1;
y=dassl([y0,yd0],0,4d10,chemres,info);
y=dassl([y0,yd0],0,4d10,chemres,chemjac,info);

//Using extra argument for parameters
//-----------------------------------
  function [r,ires]=chemres(t,y,yd ,a,b,c)
     r=[-a*y(1)+b*y(2)*y(3)-yd(1)
         a*y(1)-b*y(2)*y(3)-c*y(2)*y(2)-yd(2)
         y(1)+y(2)+y(3)-1];
     ires=0
  endfunction
```

```
  function pd=chemjac(x,y,yd,cj, a,b,c)
      pd=[-a-cj , b*y(3)             , b*y(2);
           a    ,-b*y(3)-2*c*y(2)-cj ,-b*y(2);
           1    , 1                  , 1      ]
  endfunction
y=dassl([y0,yd0],0,t,list(chemres,0.04,1d4,3d7),list(chemjac,0.04,1d4,3d7));

//using C code
//------------
// - create the C code
rescode=['void chemres(double *t, double y[], double yd[], double r[], int *ires
         ' {'
         '   r[0] = -0.04*y[0]+1.0e4*y[1]*y[2]                -yd[0];'
         '   r[1] =  0.04*y[0]-1.0e4*y[1]*y[2]-3.0e7*y[1]*y[1]-yd[1];'
         '   r[2] =       y[0]+y[1]+y[2]-1;'
         '   *ires = 0;'
         ' }'];

jaccode=['void chemjac(double *t, double y[], double yd[], double pd[], double
         ' {'
         '   /* first column*/'
         '   pd[0] = -0.04-*cj;'
         '   pd[1] =  0.04;'
         '   pd[2] =  1.0;'
         '    /* second column*/'
         '   pd[3] =  1.0e4*y[2];'
         '   pd[4] = -1.0e4*y[2]-2*3.0e7*y[1]-*cj;'
         '   pd[5] =  1.0;'
         '    /* third column*/'
         '   pd[6] =  1.0e4*y[1];'
         '   pd[7] = -1.0e4*y[1];'
         '   pd[8] =  1.0;'
         ' }'];
mputl([rescode;jaccode],TMPDIR+'/mycode.c') //create the C file
// - compile it
ilib_for_link(['chemres','chemjac'],'mycode.o',[],'c',TMPDIR+'/Makefile',TMPDIR+
// - link it with Scilab
exec(TMPDIR+'/loader.sce') //incremental linking
// - call dassl
y=dassl([y0,yd0],0,t,'chemres','chemjac');
```

See Also

ode, dasrt, impl, fort, link, external

Name

feval — multiple evaluation

```
[z]=feval(x,y,f)
[z]=feval(x,f)
```

Parameters

x,y
: two vectors

f
: function or character string (for Fortran or C call)

Description

Multiple evaluation of a function for one or two arguments of vector type :

z=feval(x,f)
: returns the vector `z` defined by `z(i)=f(x(i))`

z=feval(x,y,f)
: returns the matrix `z` such as `z(i,j)=f(x(i),y(j))`

`f` is an external (function or routine) accepting on one or two arguments which are supposed to be real. The result returned by `f` can be real or complex. In case of a Fortran call, the function `'f'` must be defined in the subroutine `ffeval.f` (in directory `SCIDIR/routines/default`)

Examples

```
deff('[z]=f(x,y)','z=x^2+y^2');
feval(1:10,1:5,f)
deff('[z]=f(x,y)','z=x+%i*y');
feval(1:10,1:5,f)
feval(1:10,1:5,'parab')   //See ffeval.f file
feval(1:10,'parab')
// For dynamic link (see example ftest in ffeval.f)
// you can use the link command (the parameters depend on the machine):
// unix('make ftest.o');link('ftest.o','ftest); feval(1:10,1:5,'ftest')
```

See Also

evstr , horner , execstr , external , link

Name

impl — differential algebraic equation

```
y=impl([type],y0,ydot0,t0,t [,atol, [rtol]],res,adda [,jac])
```

Parameters

y0,ydot0
: real vectors or matrix (initial conditions).

t0
: real scalar (initial time).

t
: real vector (times at which the solution is computed).

res,adda
: externals (function or character string or list).

type
: string `'adams'` or `'stiff'`

atol,rtol
: real scalar or real vector of the same size as as `y`.

jac
: external (function or character string or list).

Description

Solution of the linear implicit differential equation

A(t,y) dy/dt=g(t,y), y(t0)=y0

`t0` is the initial instant, `y0` is the vector of initial conditions Vector `ydot0` of the time derivative of `y` at `t0` must also be given. r The input `res` is an external i.e. a function with specified syntax, or the name a Fortran subroutine or a C function (character string) with specified calling sequence or a list.

If `res` is a function, its syntax must be as follows:

```
r = res(t,y,ydot)
```

where `t` is a real scalar (time) and `y` and `ydot` are real vector (state and derivative of the state). This function must return `r=g(t,y)-A(t,y)*ydot`.

If `res` is a character string, it refers to the name of a Fortran subroutine or a C function. See `SCIDIR/routines/default/Ex-impl.f` for an example to do that.

`res` can also be a list: see the help of `ode`.

The input `adda` is also an external.

If `adda` is a function, its syntax must be as follows:

```
r = adda(t,y,p)
```

and it must return `r=A(t,y)+p` where `p` is a matrix to be added to `A(t,y)`.

If `adda` is a character string, it refers to the name of a Fortran subroutine or a C function. See `SCIDIR/routines/default/Ex-impl.f` for an example to do that.

`adda` can also be a list: see the help of `ode`.

The input `jac` is also an external.

If `jac` is a function, its syntax must be as follows:

```
j = jac(t,y,ydot)
```

and it must return the Jacobian of `r=g(t,y)-A(t,y)*ydot` with respect to `y`.

If `jac` is a character string, it refers to the name of a Fortran subroutine or a C function. See `SCIDIR/routines/default/Ex-impl.f` for an example to do that.

`jac` can also be a list: see the help of `ode`.

Examples

```
y=impl([1;0;0],[-0.04;0.04;0],0,0.4,'resid','aplusp');
// Using hot restart
//[x1,w,iw]=impl([1;0;0],[-0.04;0.04;0],0,0.2,'resid','aplusp');
// hot start from previous call
//[x1]=impl([1;0;0],[-0.04;0.04;0],0.2,0.4,'resid','aplusp',w,iw);
//maxi(abs(x1-x))
```

See Also

dassl , ode , external

Name

int2d — definite 2D integral by quadrature and cubature method

```
[I,err]=int2d(X,Y,f [,params])
```

Parameters

X
: a 3 by N array containing the abscissae of the vertices of the N triangles.

Y
: a 3 by N array containing the ordinates of the vertices of the N triangles.

f
: external (function or list or string) defining the integrand `f(u,v)`;

params
: real vector `[tol, iclose, maxtri, mevals, iflag]`. default value is `[1.d-10, 1, 50, 4000, 1]`.

  tol
  : :the desired bound on the error. If `iflag=0`, `tol` is interpreted as a bound on the relative error; if `iflag=`1, the bound is on the absolute error.

  iclose
  : :an integer parameter that determines the selection of LQM0 or LQM1 methods. If `iclose=1` then LQM1 is used. Any other value of `iclose` causes LQM0 to be used. LQM0 uses function values only at interior points of the triangle. LQM1 is usually more accurate than LQM0 but involves evaluating the integrand at more points including some on the boundary of the triangle. It will usually be better to use LQM1 unless the integrand has singularities on the boundary of the triangle.

  maxtri
  : :the maximum number of triangles in the final triangulation of the region

  mevals
  : the maximum number of function evaluations to be allowed. This number will be effective in limiting the computation only if it is less than 94*`maxtri` when LQM1 is specified or 56*`maxtri` when LQM0 is specified.

  iflag
  : :

I
: the integral value

err
: the estimated error

Description

`int2d` computes the two-dimensional integral of a function `f` over a region consisting of `n` triangles. A total error estimate is obtained and compared with a tolerance - `tol` - that is provided as input to the subroutine. The error tolerance is treated as either relative or absolute depending on the input value of `iflag`. A 'Local Quadrature Module' is applied to each input triangle and estimates of the total integral and the total error are computed. The local quadrature module is either subroutine LQM0 or subroutine LQM1 and the choice between them is determined by the value of the input variable `iclose`.

If the total error estimate exceeds the tolerance, the triangle with the largest absolute error is divided into two triangles by a median to its longest side. The local quadrature module is then applied to each of the subtriangles to obtain new estimates of the integral and the error. This process is repeated until either (1) the error tolerance is satisfied, (2) the number of triangles generated exceeds the input parameter `maxtri`, (3) the number of integrand evaluations exceeds the input parameter `mevals`, or (4) the function senses that roundoff error is beginning to contaminate the result.

Examples

```
X=[0,0;1,1;1,0];
Y=[0,0;0,1;1,1];
deff('z=f(x,y)','z=cos(x+y)')
[I,e]=int2d(X,Y,f)
// computes the integrand over the square [0 1]x[0 1]
```

See Also

intc , intl , int3d , intg , mesh2d

Authors

Fortran routine twodq Authors: Kahaner,D.K.,N.B.S., Rechard,O.W.,N.B.S.,; Barnhill,Robert,Univ. of UTAH

Name

int3d — definite 3D integral by quadrature and cubature method

```
[result,err]=int3d(X,Y,Z,f [,nf[,params]])
```

Parameters

X
: a 4 by NUMTET array containing the abscissae of the vertices of the NUMTET tetrahedrons.

Y
: a 4 by NUMTET array containing the ordinates of the vertices of the NUMTET tetrahedrons.

Z
: a 4 by NUMTET array containing the third coordinates of the vertices of the NUMTET tetrahedrons.

f
: external (function or list or string) defining the integrand `f(xyz,nf)`, where `xyz` is the vector of a point coordinates and nf the number functions

nf
: the number of function to integate (default is 1)

params
: real vector `[minpts, maxpts, epsabs, epsrel]`. default value is `[0, 1000, 0.0, 1.d-5]`.

  epsabs
  : Desired bound on the absolute error.

  epsrel
  : Desired bound on the relative error.

  minpts
  : Minimum number of function evaluations.

  maxpts
  : Maximum number of function evaluations. The number of function evaluations over each subregion is 43

result
: the integral value,or vector of the integral values.

err
: Estimates of absolute errors.

Description

The function calculates an approximation to a given vector of definite integrals

```
I  I  I (F ,F ,...,F )      dx(3)dx(2)dx(1),
         1  2      numfun
```

where the region of integration is a collection of NUMTET tetrahedrons and where

```
F = F (X(1),X(2),X(3)), J = 1,2,...,NUMFUN.
 J   J
```

A globally adaptive strategy is applied in order to compute approximations `result(k)` hopefully satisfying, for each component of I, the following claim for accuracy: `ABS(I(K)-RESULT(K))<=MAX(EPSABS,EPSREL*ABS(I(K)))`

`int3d` repeatedly subdivides the tetrahedrons with greatest estimated errors and estimates the integrals and the errors over the new subtetrahedrons until the error request is met or `MAXPTS` function evaluations have been used.

A 43 point integration rule with all evaluation points inside the tetrahedron is applied. The rule has polynomial degree 8.

If the values of the input parameters `EPSABS` or `EPSREL` are selected great enough, an integration rule is applied over each tetrahedron and the results are added up to give the approximations `RESULT(K)`. No further subdivision of the tetrahedrons will then be applied.

When `int3d` computes estimates to a vector of integrals, all components of the vector are given the same treatment. That is, `I(Fj)` and `I(Fk)` for

`j` not equal to `k`, are estimated with the same subdivision of the region of integration. For integrals with enough similarity, we may save time by applying `int3d` to all integrands in one call. For integrals that varies continuously as functions of some parameter, the estimates produced by `int3d` will also vary continuously when the same subdivision is applied to all components. This will generally not be the case when the different components are given separate treatment.

On the other hand this feature should be used with caution when the different components of the integrals require clearly different subdivisions.

References

Fortran routine dcutet.f

Examples

```
X=[0;1;0;0];
Y=[0;0;1;0];
Z=[0;0;0;1];
[RESULT,ERROR]=int3d(X,Y,Z,'int3dex')
// computes the integrand exp(x*x+y*y+z*z) over the
//tetrahedron (0.,0.,0.),(1.,0.,0.),(0.,1.,0.),(0.,0.,1.)


//integration over a cube  -1<=x<=1;-1<=y<=1;-1<=z<=1

//  bottom  -top-     right    -left-   front   -rear-
X=[ 0, 0,    0, 0,    0, 0,    0, 0,    0, 0,    0, 0;
   -1,-1,   -1,-1,    1, 1,   -1,-1,   -1,-1,   -1,-1;
    1,-1,    1,-1,    1, 1,   -1,-1,    1,-1,    1,-1;
    1, 1,    1, 1,    1, 1,   -1,-1,    1, 1,    1, 1];
Y=[ 0, 0,    0, 0,    0, 0,    0, 0,    0, 0,    0, 0;
   -1,-1,   -1,-1,   -1, 1,   -1, 1,   -1,-1,    1, 1;
```

```
   -1, 1,   -1, 1,    1, 1,    1, 1,   -1,-1,    1, 1;
    1, 1,    1, 1,   -1,-1,   -1,-1,   -1,-1,    1, 1];
Z=[ 0, 0,    0, 0,    0, 0,    0, 0,    0, 0,    0, 0;
   -1,-1,    1, 1,   -1, 1,   -1, 1,   -1,-1,   -1,-1;
   -1,-1,    1, 1,   -1,-1,   -1,-1,   -1, 1,   -1, 1;
   -1,-1,    1, 1,    1, 1,    1, 1,    1, 1,    1, 1];

function v=f(xyz,numfun),v=exp(xyz'*xyz),endfunction
[result,err]=int3d(X,Y,Z,f,1,[0,100000,1.d-5,1.d-7])

function v=f(xyz,numfun),v=1,endfunction
[result,err]=int3d(X,Y,Z,f,1,[0,100000,1.d-5,1.d-7])
```

See Also

intc , intl , int2d

Authors

Jarle Berntsen

The Computing Centre, University of Bergen, Thormohlens gt. 55, N-5008 Bergen, Norway Phone.. 47-5-544055 Email.. jarle@eik.ii.uib.no,

Ronald Cools

Dept. of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3030 Heverlee, Belgium Phone.. 32-16-201015 (3562) Email.. ronald@cs.kuleuven.ac.be,

Terje O. Espelid

Department of Informatics, University of Bergen, Thormohlens gt. 55, N-5008 Bergen, Norway Phone.. 47-5-544180 Email.. terje@eik.ii.uib.no

Name

intg — definite integral

```
[v,err]=intg(a,b,f [,ea [,er])
```

Parameters

a,b
: real numbers

f
: external (function or list or string)

ea, er
: real numbers

ea
: absolute error required on the result. Default value: 1.d-14

er
: relative error required on the result. Default value: 1.d-8

err
: estimated absolute error on the result.

Description

`intg(a,b,f)` evaluates the definite integral from `a` to `b` of `f(t)dt`. The function `f(t)` should be continuous.

The evaluation hopefully satisfies following claim for accuracy: `abs(I-v)<= max(ea,er*abs(I))` where I stands for the exact value of the integral.

`f` is an external :

If `f` is function its definition must be as follows `y = f(t)`

If `f` is a list the list must be as follows: `list(f,x1,x2,...)` where `f` is a function with calling sequence `f(t,x1,x2,...)`.

If `f` is a string it refers to a the name of a Fortran function or a C prodedure with a given calling sequence:

In the fortran case the calling sequence should be `double precision function f(x)` where `x` is also a double precision number.

In the C case the calling sequence should be `double f(double *x)`.

Examples

```
//Scilab function case
function y=f(x),y=x*sin(30*x)/sqrt(1-((x/(2*%pi))^2)),endfunction
exact=-2.5432596188;
I=intg(0,2*%pi,f)
abs(exact-I)
```

```
//Scilab function case with parameter
function y=f1(x,w),y=x*sin(w*x)/sqrt(1-((x/(2*%pi))^2)),endfunction
I=intg(0,2*%pi,list(f1,30))
abs(exact-I)


// Fortran code case (Fortran compiler required)
// write down the fortran code
F=['      double precision function ffun(x)'
   '      double precision x,pi'
   '      pi=3.14159265358979312d+0'
   '      ffun=x*sin(30.0d+0*x)/sqrt(1.0d+0-(x/(2.0d+0*pi))**2)'
   '      return'
   '      end'];
mputl(F,TMPDIR+'/ffun.f')
// compile the fortran code
l=ilib_for_link('ffun','ffun.o',[],'f',TMPDIR+'/Makefile');
// incremental linking
link(l,'ffun','f')
//integrate the function
I=intg(0,2*%pi,'ffun')
abs(exact-I)

// C code case (C compiler required)
// write down the C code
C=['#include <math.h>'
   'double cfun(double *x)'
   '{'
   '  double y,pi=3.14159265358979312;'
   '  y=*x/(2.0e0*pi);'
   '  return *x*sin(30.0e0**x)/sqrt(1.0e0-y*y);'
   '}'];
mputl(C,TMPDIR+'/cfun.c')
// compile the C code
l=ilib_for_link('cfun','cfun.o',[],'c',TMPDIR+'/Makefile');
// incremental linking
link(l,'cfun','c')
//integrate the function
I=intg(0,2*%pi,'cfun')
abs(exact-I)
```

See Also

intc , intl , inttrap , intsplin , ode

Used Functions

The associated routines can be found in routines/integ directory :

dqag0.f and dqags.f from quadpack

Name

ode — ordinary differential equation solver

```
y=ode(y0,t0,t,f)
[y,w,iw]=ode([type],y0,t0,t [,rtol [,atol]],f [,jac] [,w,iw])
[y,rd,w,iw]=ode("root",y0,t0,t [,rtol [,atol]],f [,jac],ng,g [,w,iw])
y=ode("discrete",y0,k0,kvect,f)
```

Parameters

y0
: real vector or matrix (initial conditions).

t0
: real scalar (initial time).

t
: real vector (times at which the solution is computed).

f
: external (function or character string or list).

type
: one of the following character string: `"adams"` `"stiff"` `"rk"` `"rkf"` `"fix"` `"discrete"` `"roots"`

rtol,atol
: real constants or real vectors of the same size as `y`.

jac
: external (function or character string or list).

w,iw
: real vectors.

ng
: integer.

g
: external (function or character string or list).

k0
: integer (initial time). kvect : integer vector.

Description

`ode` is the standard function for solving explicit ODE systems defined by: dy/dt=f(t,y) , y(t0)=y0. It is an interface to various solvers, in particular to ODEPACK. The type of problem solved and the method used depend on the value of the first optional argument `type` which can be one of the following strings:

<not given>:
: `lsoda` solver of package ODEPACK is called by default. It automatically selects between nonstiff predictor-corrector Adams method and stiff Backward Differentiation Formula (BDF) method. It uses nonstiff method initially and dynamically monitors data in order to decide which method to use.

"adams":
: This is for nonstiff problems. `lsode` solver of package ODEPACK is called and it uses the Adams method.

"stiff":
This is for stiff problems. `lsode` solver of package ODEPACK is called and it uses the BDF method.

"rk":
Adaptive Runge-Kutta of order 4 (RK4) method.

"rkf":
The Shampine and Watts program based on Fehlberg's Runge-Kutta pair of order 4 and 5 (RKF45) method is used. This is for non-stiff and mildly stiff problems when derivative evaluations are inexpensive. This method should generally not be used when the user is demanding high accuracy.

"fix":
Same solver as "rkf", but the user interface is very simple, i.e. only `rtol` and `atol` parameters can be passed to the solver. This is the simplest method to try.

"root":
ODE solver with rootfinding capabilities. The `lsodar` solver of package ODEPACK is used. It is a variant of the `lsoda` solver where it finds the roots of a given vector function. See help on ode_root for more details.

"discrete":
Discrete time simulation. See help on ode_discrete for more details.

In this help we only describe the use of `ode` for standard explicit ODE systems.

- The simplest call of `ode` is: `y=ode(y0,t0,t,f)` where `y0` is the vector of initial conditions, `t0` is the initial time, `t` is the vector of times at which the solution `y` is computed and `y` is matrix of solution vectors `y=[y(t(1)),y(t(2)),...]`.

  The input argument `f` defines the RHS of the first order differential equation: dy/dt=f(t,y). It is an external i.e. a function with specified syntax, or the name of a Fortran subroutine or a C function (character string) with specified calling sequence or a list:

  - If `f` is a Scilab function, its syntax must be `ydot = f(t,y)`, where `t` is a real scalar (time) and `y` a real vector (state) and `ydot`a real vector (dy/dt)
  - If `f` is a character string, it refers to the name of a Fortran subroutine or a C function, i.e. if `ode(y0,t0,t,"fex")` is the command, then the subroutine `fex` is called.

    The Fortran routine must have the following calling sequence: `fex(n,t,y,ydot)`, with n an integer, t a double precision scalar, y and ydot double precision vectors.

    The C function must have the following prototype: `fex(int *n,double *t,double *y,double *ydot)`

    `t` is the time, `y` the state and `ydot`the state derivative (dy/dt)

    This external can be build in a OS independant way using ilib_for_link and dynamically linked to Scilab by the link function.
  - The `f` argument can also be a list with the following structure: `lst=list(realf,u1,u2,...un)` where `realf` is a Scilab function with syntax: `ydot = f(t,y,u1,u2,...,un)`

    This syntax allows to use parameters as the arguments of `realf`.

  The function `f` can return a `p x q` matrix instead of a vector. With this matrix notation, we solve the `n=p+q` ODE's system `dY/dt=F(t,Y)` where `Y` is a `p x q` matrix. Then initial conditions, `Y0`, must also be a `p x q` matrix and the result of `ode` is the `p x q(T+1)` matrix `[Y(t_0),Y(t_1),...,Y(t_T)]`.

- Optional input parameters can be given for the error of the solution: `rtol` and `atol` are threshold for relative and absolute estimated errors. The estimated error on `y(i)` is: `rtol(i)*abs(y(i))+atol(i)`

  and integration is carried out as far as this error is small for all components of the state. If `rtol` and/or `atol` is a constant `rtol(i)` and/or `atol(i)` are set to this constant value. Default values for `rtol` and `atol` are respectively `rtol=1.d-5` and `atol=1.d-7` for most solvers and `rtol=1.d-3` and `atol=1.d-4` for `"rfk"` and `"fix"`.

- For stiff problems, it is better to give the Jacobian of the RHS function as the optional argument `jac`. It is an external i.e. a function with specified syntax, or the name of a Fortran subroutine or a C function (character string) with specified calling sequence or a list.

  If `jac` is a function the syntax should be `J=jac(t,y)`

  where `t` is a real scalar (time) and `y` a real vector (state). The result matrix `J` must evaluate to df/dx i.e. `J(k,i) = dfk/dxi` with `fk` = kth component of f.

  If `jac` is a character string it refers to the name of a Fortran subroutine or a C function, with the following calling sequence:

  Fortran case:

  ```
  subroutine fex(n,t,y,ml,mu,J,nrpd)
  integer n,ml,mu,nrpd
  double precision t,y(*),J(*)
  ```

  C case:

  ```
  void fex(int *n,double *t,double *y,int *ml,int *mu,double *J,int *nrpd
  ```

  `jac(n,t,y,ml,mu,J,nrpd)`. In most cases you have not to refer `ml`, `mu` and `nrpd`.

  If `jac` is a list the same conventions as for `f` apply.

- Optional arguments `w` and `iw` are vectors for storing information returned by the integration routine (see ode_optional_output for details). When these vectors are provided in RHS of `ode` the integration re-starts with the same parameters as in its previous stop.

- More options can be given to ODEPACK solvers by using `%ODEOPTIONS` variable. See odeoptions.

Examples

```
// ---------- Simple one dimension ODE (Scilab function external)
// dy/dt=y^2-y sin(t)+cos(t), y(0)=0
function ydot=f(t,y),ydot=y^2-y*sin(t)+cos(t),endfunction
y0=0;t0=0;t=0:0.1:%pi;
y=ode(y0,t0,t,f)
plot(t,y)

// ---------- Simple one dimension ODE (C coded external)
ccode=['#include <math.h>'
'void myode(int *n,double *t,double *y,double *ydot)'
```

```
'{'
'  ydot[0]=y[0]*y[0]-y[0]*sin(*t)+cos(*t);'
'}']
mputl(ccode,TMPDIR+'/myode.c') //create the C file
ilib_for_link('myode','myode.o',[],'c',TMPDIR+'/Makefile',TMPDIR+'/loader.s
exec(TMPDIR+'/loader.sce') //incremental linking
y0=0;t0=0;t=0:0.1:%pi;
y=ode(y0,t0,t,'myode');

// ---------- Simulation of dx/dt = A x(t) + B u(t) with u(t)=sin(omega*t),
// x0=[1;0]
// solution x(t) desired at t=0.1, 0.2, 0.5 ,1.
// A and u function are passed to RHS function in a list.
// B and omega are passed as global variables
function xdot=linear(t,x,A,u),xdot=A*x+B*u(t),endfunction
function ut=u(t),ut=sin(omega*t),endfunction
A=[1 1;0 2];B=[1;1];omega=5;
ode([1;0],0,[0.1,0.2,0.5,1],list(linear,A,u))

// ---------- Matrix notation Integration of the Riccati differential equat
// Xdot=A'*X + X*A - X'*B*X + C , X(0)=Identity
// Solution at t=[1,2]
function Xdot=ric(t,X),Xdot=A'*X+X*A-X'*B*X+C,endfunction
A=[1,1;0,2]; B=[1,0;0,1]; C=[1,0;0,1];
t0=0;t=0:0.1:%pi;
X=ode(eye(A),0,t,ric)
//
// ---------- Matrix notation, Computation of exp(A)
A=[1,1;0,2];
function xdot=f(t,x),xdot=A*x;,endfunction
ode(eye(A),0,1,f)
ode("adams",eye(A),0,1,f)

// ---------- Matrix notation, Computation of exp(A) with stiff matrix, Jac
A=[10,0;0,-1];
function xdot=f(t,x),xdot=A*x,endfunction
function J=Jacobian(t,y),J=A,endfunction
ode("stiff",[0;1],0,1,f,Jacobian)
```

See Also

ode_discrete , ode_root , dassl , impl , odedc , odeoptions , csim , ltitr , rtitr

Authors

Alan C. Hindmarsh
, mathematics and statistics division, l-316 livermore, ca 94550.19

Bibliography

Alan C. Hindmarsh, lsode and lsodi, two new initial value ordinary differential equation solvers, acm-signum newsletter, vol. 15, no. 4 (1980), pp. 10-11.

Used Functions

The associated routines can be found in routines/integ directory :

lsode.f lsoda.f lsodar.f

Name

ode_discrete — ordinary differential equation solver, discrete time simulation

```
y=ode("discrete",y0,k0,kvect,f)
```

Parameters

y0
: real vector or matrix (initial conditions).

t0
: real scalar (initial time).

f
: external i.e. function or character string or list.

k0
: integer (initial time).

kvect
: integer vector.

Description

With this syntax (first argument equal to `"discrete"`) `ode` computes recursively `y(k+1)=f(k,y(k))` from an initial state `y(k0)` and returns `y(k)` for `k` in `kvect`. `kvect(1)` must be greater than or equal to `k0`.

Other arguments and other options are the same as for `ode`, see the ode help.

Examples

```
y1=[1;2;3]; deff("yp=a_function(k,y)","yp=A*y+B*u(k)")
A=diag([0.2,0.5,0.9]); B=[1;1;1];u=1:10;n=5;
y=ode("discrete",y1,1,1:n,a_function);
y(:,2)-(A*y1+B*u(1))
// Now y evaluates  at [y3,y5,y7,y9]
y=ode("discrete",y1,1,3:2:9,a_function)
```

See Also

ode

Name

ode_optional_output — ode solvers optional outputs description

Description

This page describes the the most important values returned in the optional lhs ode function arguments `w` and `iw`. These are valid only for the `lsode` `lsoda` and `lsodar` ode solver. For more details, one can look at the solvers fortran code comments in `routines/integ/lsod*.f`.

w(11)
: the step size in t last used (successfully).

w(12)
: the step size to be attempted on the next step.

w(13)
: the current value of the independent variable which the solver has actually reached, i.e. the current internal mesh point in t. on output, tcur will always be at least as far as the argument t, but may be farther (if interpolation was done).

w(14)
: a tolerance scale factor, greater than 1.0, computed when a request for too much accuracy was detected (istate = -3 if detected at the start of the problem, istate = -2 otherwise). if itol is left unaltered but rtol and atol are uniformly scaled up by a factor of `tolsf=w(14)` for the next call, then the solver is deemed likely to succeed. (the user may also ignore `tolsf` and alter the tolerance parameters in any other way appropriate.)

w(15)
: the value of t at the time of the last method switch, if any. This value is not significant with `lsode` solver.

iw(10)
: the number of `g` evaluations for the problem so far. This value is only significant for `lsodar` solver.

iw(11)
: the number of steps taken for the problem so far.

iw(12)
: the number of f evaluations for the problem so far.

iw(13)
: the number of jacobian evaluations (and of matrix lu decompositions) for the problem so far.

iw(14)
: the method order last used (successfully).

iw(15)
: the order to be attempted on the next step.

iw(16)
: the index of the component of largest magnitude in the weighted local error vector (e(i)/ewt(i)), on an error return with istate = -4 or -5.

iw(17)
: the length of `w` actually required, assuming that the length of rwork is to be fixed for the rest of the problem, and that switching may occur. this is defined on normal returns and on an illegal input return for insufficient storage.

iw(18)
: the length of `iw` actually required, assuming that the length of `iw` is to be fixed for the rest of the problem, and that switching may occur. this is defined on normal returns and on an illegal input return for insufficient storage.

iw(19)
: the method indicator for the last successful step.. 1 means adams (nonstiff), 2 means bdf (stiff). This value is not significant with `lsode` solver.

iw(20)
: the current method indicator.. 1 means adams (nonstiff), 2 means bdf (stiff). this is the method to be attempted on the next step. thus it differs from `iw(19)` only if a method switch has just been made. This value is not significant with `lsode` solver.

Name

ode_root — ordinary differential equation solver with root finding

```
y,rd[,w,iw]=ode("root",y0,t0,t [,rtol [,atol]],f [,jac],ng,g [,w,iw])
```

Parameters

y0
: real vector or matrix (initial conditions).

t0
: real scalar (initial time).

t
: real vector (times at which the solution is computed).

f
: external i.e. function or character string or list.

rtol,atol
: real constants or real vectors of the same size as `y`.

jac
: external i.e. function or character string or list.

w,iw
: real vectors.

ng
: integer.

g
: external i.e. function or character string or list.

Description

With this syntax (first argument equal to `"root"`) `ode` computes the solution of the differential equation `dy/dt=f(t,y)` until the state `y(t)` crosses the surface `g(t,y)=0`.

`g` should give the equation of the surface. It is an external i.e. a function with specified syntax, or the name of a Fortran subroutine or a C function (character string) with specified calling sequence or a list.

If `g` is a function the syntax should be as follows:

```
z=g(t,y)
```

where `t` is a real scalar (time) and `y` a real vector (state). It returns a vector of size `ng` which corresponds to the `ng` constraints. If `g` is a character string it refers to the name of a Fortran subroutine or a C function, with the following calling sequence: `g(n,t,y,ng,gout)` where `ng` is the number of constraints and `gout` is the value of `g` (output of the program). If `g` is a list the same conventions as for `f` apply (see ode help).

Ouput `rd` is a `1 x k` vector. The first entry contains the stopping time. Other entries indicate which components of `g` have changed sign. `k` larger than 2 indicates that more than one surface (`(k-1)` surfaces) have been simultaneously traversed.

Other arguments and other options are the same as for ode, see the ode help.

Examples

```
// Integration of the differential equation
// dy/dt=y , y(0)=1, and finds the minimum time t such that y(t)=2
deff("[ydot]=f(t,y)","ydot=y")
deff("[z]=g(t,y)","z=y-2")
y0=1;ng=1;
[y,rd]=ode("roots",y0,0,2,f,ng,g)


deff("[z]=g(t,y)","z=y-[2;2;33]")
[y,rd]=ode("roots",1,0,2,f,3,g)
```

See Also

dasrt , ode

Name

odedc — discrete/continuous ode solver

```
yt=odedc(y0,nd,stdel,t0,t,f)
```

Parameters

y0
: real column vector (initial conditions), `y0=[y0c;y0d]` where `y0d` has `nd` components.

nd
: integer, dimension of `y0d`

stdel
: real vector with one or two entries, `stdel=[h, delta]` (with `delta=0` as default value).

t0
: real scalar (initial time).

t
: real (row) vector, instants where `yt` is calculated .

f
: external i.e. function or character string or list with calling sequence: `yp=f(t,yc,yd,flag)`.

Description

`y=odedc([y0c;y0d],nd,[h,delta],t0,t,f)` computes the solution of a mixed discrete/ continuous system. The discrete system state `yd_k` is embedded into a piecewise constant `yd(t)` time function as follows:

```
yd(t)=yd_k for t in
[t_k=delay+k*h,t_(k+1)=delay+(k+1)*h[ (with delay=h*delta).
```

The simulated equations are now:

```
dyc/dt=f(t,yc(t),yd(t),0),  for t in [t_k,t_(k+1)[
yc(t0)=y0c
```

and at instants `t_k` the discrete variable `yd` is updated by:

```
yd(t_k+)=f(yc(t_k-),yd(t_k-),1)
```

Note that, using the definition of `yd(t)` the last equation gives

```
yd_k = f (t_k,yc(t_k-),yd(t_(k-1)),1)  (yc is time-continuous: yc(t_k-)=yc(tk))
```

The calling parameters of `f` are fixed: `ycd=f(t,yc,yd,flag)`; this function must return either the derivative of the vector `yc` if `flag=0` or the update of `yd` if `flag=1`.

`ycd=dot(yc)` must be a vector with same dimension as `yc` if `flag=0` and `ycd=update(yd)` must be a vector with same dimension as `yd` if `flag=1`.

`t` is a vector of instants where the solution `y` is computed.

`y` is the vector `y=[y(t(1)),y(t(2)),...]`. This function can be called with the same optional parameters as the `ode` function (provided `nd` and `stdel` are given in the calling sequence as second and third parameters). In particular integration flags, tolerances can be set. Optional parameters can be set by the `odeoptions` function.

An example for calling an external routine is given in directory `SCIDIR/default/fydot2.f`

External routines can be dynamically linked (see `link`).

Examples

```
//Linear system with switching input
deff('xdu=phis(t,x,u,flag)','if flag==0 then xdu=A*x+B*u; else xdu=1-u;end');
x0=[1;1];A=[-1,2;-2,-1];B=[1;2];u=0;nu=1;stdel=[1,0];u0=0;t=0:0.05:10;
xu=odedc([x0;u0],nu,stdel,0,t,phis);x=xu(1:2,:);u=xu(3,:);
nx=2;
plot2d1('onn',t',x',[1:nx],'161');
plot2d2('onn',t',u',[nx+1:nx+nu],'000');
//Fortran external( see fydot2.f):
norm(xu-odedc([x0;u0],nu,stdel,0,t,'phis'),1)

//Sampled feedback
//
//        |     xcdot=fc(t,xc,u)
//  (system)   |
//        |     y=hc(t,xc)
//
//
//        |     xd+=fd(xd,y)
//  (feedback) |
//        |     u=hd(t,xd)
//
deff('xcd=f(t,xc,xd,iflag)',...
  ['if iflag==0 then '
   '  xcd=fc(t,xc,e(t)-hd(t,xd));'
   'else '
   '  xcd=fd(xd,hc(t,xc));'
   'end']);
A=[-10,2,3;4,-10,6;7,8,-10];B=[1;1;1];C=[1,1,1];
Ad=[1/2,1;0,1/20];Bd=[1;1];Cd=[1,1];
deff('st=e(t)','st=sin(3*t)')
deff('xdot=fc(t,x,u)','xdot=A*x+B*u')
deff('y=hc(t,x)','y=C*x')
deff('xp=fd(x,y)','xp=Ad*x + Bd*y')
```

```
deff('u=hd(t,x)','u=Cd*x')
h=0.1;t0=0;t=0:0.1:2;
x0c=[0;0;0];x0d=[0;0];nd=2;
xcd=odedc([x0c;x0d],nd,h,t0,t,f);
norm(xcd-odedc([x0c;x0d],nd,h,t0,t,'fcd1')) // Fast calculation (see fydot2.f)
plot2d([t',t',t'],xcd(1:3,:)');
xset("window",2);plot2d2("gnn",[t',t'],xcd(4:5,:)');
xset("window",0);
```

See Also

ode , odeoptions , csim , external

Name

odeoptions — set options for ode solvers

```
odeoptions()
```

Description

This function interactively displays a command which should be executed to set various options of ode solvers. The global variable `%ODEOPTIONS` sets the options.

CAUTION: the `ode` function checks if this variable exists and in this case it uses it. For using default values you should clear this variable. Note that `odeoptions` does not create this variable. To create it you must execute the command line displayed by `odeoptions`.

The variable `%ODEOPTIONS` is a vector with the following elements:

```
[itask,tcrit,h0,hmax,hmin,jactyp,mxstep,maxordn,maxords,ixpr,ml,mu]
```

The default value is:

```
[1,0,0,%inf,0,2,500,12,5,0,-1,-1]
```

The meaning of the elements is described below.

`itask` 1 : normal computation at specified times 2 : computation at mesh points (given in first row of output of `ode`) 3 : one step at one internal mesh point and return 4 : normal computation without overshooting `tcrit` 5 : one step, without passing `tcrit`, and return

`tcrit` assumes `itask` equals 4 or 5, described above

`h0` first step tried

`hmax` max step size

`hmin` min step size

`jactype` 0 : functional iterations, no jacobian used (`"adams"` or `"stiff"` only) 1 : user-supplied full jacobian 2 : internally generated full jacobian 3 : internally generated diagonal jacobian (`"adams"` or `"stiff"` only) 4 : user-supplied banded jacobian (see `ml` and `mu` below) 5 : internally generated banded jacobian (`see` ml and `mu` below)

`maxordn` maximum non-stiff order allowed, at most 12

`maxords` maximum stiff order allowed, at most 5

`ixpr` print level, 0 or 1

`ml`,`mu` If `jactype` equals 4 or 5, `ml` and `mu` are the lower and upper half-bandwidths of the banded jacobian: the band is the i,j's with i-ml <= j <= ny-1. If `jactype` equals 4 the jacobian function must return a matrix J which is ml+mu+1 x ny (where ny=dim of y in ydot=f(t,y)) such that column 1 of J is made of mu zeros followed by df1/dy1, df2/dy1, df3/dy1, ... (1+ml possibly non-zero entries) column 2 is made of mu-1 zeros followed by df1/dx2, df2/dx2, etc

See Also

ode

Dynamic/incremental Link

Name

G_make — call make or nmake

```
Rfiles=G_make(files,dllname)
```

Parameters

files
: a character string or a vector of character string.

dllname
: a character string.

Rfiles
: vector of character string. `Rfiles` can be used as a first argument when calling `addinter` function.

Description

On Unix like systems `G_make` calls the `make` utility for building target `files` and returns the value of `files` in the variable `Rfiles`. On windows platforms, `G_make` calls the `nmake` utility for building target `dllname` and it returns the value of `dllname` in the variable `Rfiles`. Of course `G_make` will work if apropriate Makefiles are provided in the current Scilab directory.

`G_make` can be used to provide OS independant call to addinter.

Examples

```
if MSDOS then
 txt = ['ex1c.dll:',
        '  @echo ------------------------------------------',
         '  @echo From Makefile.mak',
        '  @echo ------------------------------------------',
        ' '];
  mputl(txt,TMPDIR+'/makefile.mak')
  current_dir = pwd();
  cd TMPDIR
  files=G_make([TMPDIR+'/ex1cI.o',TMPDIR+'/ex1c.o'],'ex1c.dll');// compilation
  //
  //addinter(files,'foobar','foubare'); // link
  cd(current_dir);
end
```

See Also

addinter

Name

VCtoLCCLib — converts Ms VC libs to LCC-Win32 libs.

```
VCtoLCCLib()
```

Description

converts Ms VC libs to LCC-Win32 libs.

Examples

```
bOK=chooselcccompiler();VCtoLCCLib()
```

Authors

Allan CORNET

Name

addinter — new functions interface incremental/dynamic link at run time

```
addinter(files,spname,fcts)
```

Parameters

files
: a character string or a vector of character string contain object files used to define the new Scilab interface routine (interface code, user routines or libraries, system libraries).

spname
: a character string. Name of interface routine entry point

fcts
: vector of character strings. The name of new Scilab function implemented in the new interface (in `fin` the order).

Description

`addinter` performs incremental/dynamic link of a compiled C or Fortran new Scilab interface routine (see `intersci` documentation) and define corresponding scilab functions.

You can use the command `link('show')` to get the number of the shared libraries. And to reload a new version of an interface a call to `ulink` is necessary to get rid of the old version.

See `link` for more precision on use.

Number of 'addinter' in a scilab session can be limited by the operating system. On Windows, you cannot load more than 80 dynamic libraries at the same time.

See Also

link, intersci, newfun, clearfun

Name

c_link — check incremental/dynamic link

```
c_link(routine-name)
[test,ilib]=c_link(routine-name)
test=c_link(routine-name,num)
```

Parameters

routine-name
: a character string

num
: :

test
: boolean, indicates if there is a shared library which contains `routine-name`.

ilib
: a scalar, the number of the shared library which contains `routine-name`

Description

`c_link` is a boolean function which checks if the routine `routine-name` is currently linked. This function returns a boolean value true or false. When used with two return values, the function `c_link` returns a boolean value in `test` and the number of the shared library which contains `routine-name` in `ilib` (when `test` is true).

See Also

link , fort

Name

call — Fortran or C user routines call

```
// long form 'out' is present
[y1,...,yk]=call("ident",x1,px1,"tx1",...,xn,pxn,"txn",
"out",[ny1,my1],py1,"ty1",...,[nyl,myl],pyl,"tyl")
// short form : no 'out' parameter
[y1,....,yk]=call("ident",x1,...,xn)
```

Parameters

"ident"
: string.

xi
: real matrix or string

pxi, pyi
: integers

txi, tyi
: character string `"d"`, `"r"`, `"i"` or `"c"`.

Description

Interactive call of Fortran (or C) user program from Scilab. The routine must be previously linked with Scilab. This link may be done:

- with Scilab "`link`" command (incremental "soft" linking) during the Scilab session.(see `link`)
- by "hard" re-linking. Writing the routine call within Scilab routine `default/Ex-fort.f`, adding the entry point in the file `default/Flist` and then re_linking Scilab with the command `make bin/scilex` in main Scilab directory.

There are two forms of calling syntax, a short one and a long one. The short one will give faster code and an easier calling syntax but one has to write a small (C or Fortran) interface in order to make the short form possible. The long one make it possible to call a Fortran routine (or a C one) whitout modification of the code but the syntax is more complex and the interpreted code slower.

The meaning of each parameter is described now:

"ident"
: is the name of the called subroutine.

x1,...,xn
: are input variables (real matrices or strings) sent to the routine,

px1,...,pxn
: are the respective positions of these variables in the calling sequence of the routine `"ident"` and

tx1,...,txn
: are their types (`"r"`, `"i"`, `"d"` and `"c"` for real (float) , integer, double precision and strings)

"out"
: is a keyword used to separate input variables from output variables. when this key word is present it is assumes that the long form will be used and when it is not prsent, the short form is used.

[ny1, my1]
: are the size (# of rows and columns. For 'c' arguments,`m1*n1` is the number of charaters) of output variables and

py1, ...
: are the positions of output variables (possibly equal to `pxi`) in the calling sequence of the routine. The `pyi`'s integers must be in increasing order.

"ty1", ...
: are the Fortran types of output variables. The `k` first output variables are put in `y1,..., yk`.

If an output variable coincides with an input variable (i.e. `pyi=pxj`) one can pass only its position `pyi` . The size and type of `yi` are then the same as those of `xi`. If an output variable coincides with an input variable and one specify the dimensions of the output variable `[myl,nyl]` must follow the compatibility condition `mxk*nxk >= myl*nyl`.

In the case of short syntax , `[y1,....,yk]=call("ident",x1,...,xn)`, the input parameters `xi`'s and the name `"ident"` are sent to the interface routine `Ex-fort`. This interface routine is then very similar to an interface (see the source code in the directory `SCIDIR/default/Ex-fort.f`).

Examples

```
//Example 1 with  a simple C code
f1=['#include <math.h>'
    'void fooc(c,a,b,m,n)'
     'double a[],*b,c[];'
     'int *m,*n;'
     '{'
     '   int i;'
     '   for ( i =0 ; i < (*m)*(*n) ; i++) '
     '     c[i] = sin(a[i]) + *b; '
     '}'];

mputl(f1,'fooc.c')

//creating the shared library (a gateway, a Makefile and a loader are
//generated.

ilib_for_link('fooc','fooc.o',[],"c")

// load the shared library

exec loader.sce

//using the new primitive
a=[1,2,3;4,5,6];b= %pi;
[m,n]=size(a);

// Inputs:
// a is in position 2 and double
// b               3     double
// n               4     integer
// m               5     integer
// Outputs:
// c is in position 1 and double with size [m,n]
c=call("fooc",a,2,"d",b,3,"d",m,4,"i",n,5,"i","out",[m,n],1,"d");

//Example 2 with  a simple Fortran code
f1=['      subroutine foof(c,a,b,n,m)'
```

```
    '      integer n,m'
    '      double precision a(*),b,c(*)'
    '      do 10 i=1,m*n '
    '        c(i) = sin(a(i))+b'
    '   10 continue'
    '      end'];
mputl(f1,'foof.f')

//creating the shared library (a gateway, a Makefile and a loader are
//generated.

ilib_for_link('foof','foof.o',[],"f")

// load the shared library

exec loader.sce

//using the new primitive
a=[1,2,3;4,5,6];b= %pi;
[m,n]=size(a);
c=call("foof",a,2,"d",b,3,"d",m,4,"i",n,5,"i","out",[m,n],1,"d");
```

See Also

link , c_link , intersci , addinter

Name

chooselcccompiler — choose LCC-Win32 as the default C Compiler.

```
bOK=chooselcccompiler()
```

Parameters

bOK
: returns %T if LCC-Win32 is the default C Compiler.

Description

choose LCC-Win32 as the default C Compiler.

Examples

```
bOK=chooselcccompiler()
```

Authors

Allan CORNET

Name

configure_lcc — set environments variables for LCC-Win32 C Compiler.

```
bOK=configure_lcc()
```

Parameters

bOK
: returns %T if environments variables for LCC-Win32 C Compiler are OK.

Description

set environments variables for LCC-Win32 C Compiler.

Examples

```
bOK=configure_lcc()
```

Authors

Allan CORNET

Name

configure_ifort — set environments variables for Intel Fortran Compiler (Windows).

```
bOK=configure_msifort()
```

Parameters

bOK
: returns %T if environments variables for Intel fortran (9 or 10) Compiler are OK.

Description

set environments variables for Intel fortran (9 or 10) Compiler.

Examples

```
bOK = configure_msifort()
```

Authors

Allan CORNET

Name

configure_msvc — set environments variables for Microsoft C Compiler.

```
bOK=configure_msvc()
```

Parameters

bOK
: returns %T if environments variables for Ms C Compiler are OK.

Description

set environments variables for Microsoft C Compiler.

Examples

```
bOK=configure_msvc()
```

Authors

Allan CORNET

Name

dllinfo — provides information about the format and symbols provided in executable and DLL files (Windows).

```
infolist = dllinfo(filename,option)
```

Parameters

filename
: a string : a filename .dll or .exe file

option
: a string : 'machine' , 'exports' , 'imports'

infolist
: a list :

: infolist(1) : a string : name of dll or executable.

: infolist(2) : a string matrix : symbols (imported or exported) or machine type (x86 or x64).

Description

This tool provides information about the format and symbols (imported or exported) provided in executable and DLL files.

This tool is based on dumpbin.exe. A tool provided with Visual studio SDK.

Examples

```
if MSDOS then
  filename = SCI+'\bin\libscilab.dll';

  dllinfolist = dllinfo(filename,'machine');
  printf('Machine destination of %s: %s\n',dllinfolist(1),dllinfolist(2));

  dllinfolist = dllinfo(filename,'imports');
  printf('Dlls dependencies of %s:\n',filename);
  for i=1:size(dllinfolist)
    printf('%s\n',dllinfolist(i)(1));
  end

  dllinfolist = dllinfo(filename,'exports');
  printf('Dll exports of %s:\n',filename);
  disp(dllinfolist);
end
```

See Also

addinter, link, ilib_compile, ilib_gen_Make, ilib_gen_gateway, ilib_gen_loader, ilib_for_link

Authors

Allan CORNET

Name

findlcccompiler — detects LCC-Win32 C Compiler

```
ret=findlcccompiler()
```

Parameters

ret
: returns %T or %F

Description

detects LCC-Win32 C Compiler.

Examples

```
ret=findlcccompiler()
```

Authors

Allan CORNET

Name

findmsifortcompiler — detects Intel fortran Compiler

```
ifortv=findmsifortcompiler()
```

Parameters

ifortv
: returns 'ifort90','ifort10','unknown'

Description

detects Intel fortran Compiler (Windows).

Examples

```
ifortv = findmsifortcompiler()
```

Authors

Allan CORNET

Name

findmsvccompiler — detects Microsoft C Compiler

```
msvc=findmsvccompiler()
```

Parameters

msvc
: returns 'msvc70','msvc71','msvc80express','msvc80std,'msvc80pro','msvc90express','msvc90std,'msvc90pro','unknown'

Description

detects Microsoft C Compiler.

Examples

```
msvc=findmsvccompiler()
```

Authors

Allan CORNET

Name

fort — Fortran or C user routines call

```
// long form 'out' is present
[y1,...,yk]=fort("ident",x1,px1,"tx1",...,xn,pxn,"txn",
"out",[ny1,my1],py1,"ty1",...,[nyl,myl],pyl,"tyl")
// short form : no 'out' parameter
[y1,....,yk]=fort("ident",x1,...,xn)
```

Parameters

"ident"
: string.

xi
: real matrix or string

pxi, pyi
: integers

txi, tyi
: character string `"d"`, `"r"`, `"i"` or `"c"`.

Description

Interactive call of Fortran (or C) user program from Scilab. The routine must be previously linked with Scilab. This link may be done:

- with Scilab "`link`" command (incremental "soft" linking) during the Scilab session.(see `link`)
- by "hard" re-linking. Writing the routine call within Scilab routine `default/Ex-fort.f`, adding the entry point in the file `default/Flist` and then re_linking Scilab with the command `make bin/scilex` in main Scilab directory.

There are two forms of calling syntax, a short one and a long one. The short one will give faster code and an easier calling syntax but one has to write a small (C or Fortran) interface in order to make the short form possible. The long one make it possible to call a Fortran routine (or a C one) whitout modification of the code but the syntax is more complex and the interpreted code slower.

The meaning of each parameter is described now:

"ident"
: is the name of the called subroutine.

x1,...,xn
: are input variables (real matrices or strings) sent to the routine,

px1,...,pxn
: are the respective positions of these variables in the calling sequence of the routine `"ident"` and

tx1,...,txn
: are their types (`"r"`, `"i"`, `"d"` and `"c"` for real (float) , integer, double precision and strings)

"out"
: is a keyword used to separate input variables from output variables. when this key word is present it is assumes that the long form will be used and when it is not prsent, the short form is used.

[ny1, my1]
: are the size (number of rows and columns. For 'c' arguments,`m1*n1` is the number of charaters) of output variables and

py1, ...
: are the positions of output variables (possibly equal to `pxi`) in the calling sequence of the routine. The `pyi`'s integers must be in increasing order.

"ty1", ...
: are the Fortran types of output variables. The `k` first output variables are put in `y1, ..., yk`.

If an output variable coincides with an input variable (i.e. `pyi=pxj`) one can pass only its position `pyi` . The size and type of `yi` are then the same as those of `xi`. If an output variable coincides with an input variable and one specify the dimensions of the output variable `[myl,nyl]` must follow the compatibility condition `mxk*nxk >= myl*nyl`.

In the case of short syntax , `[y1,....,yk]=fort("ident",x1,...,xn)`, the input parameters `xi`'s and the name `"ident"` are sent to the interface routine `Ex-fort`. This interface routine is then very similar to an interface (see the source code in the directory `SCIDIR/default/Ex-fort.f`).

For example the following program:

```
 subroutine foof(c,a,b,n,m)
 integer n,m
 double precision a(*),b,c(*)
 do 10 i=1,m*n
   c(i) = sin(a(i))+b
 10 continue
 end

link("foof.o","foof")
a=[1,2,3;4,5,6];b= %pi;
[m,n]=size(a);
// Inputs:
// a is in position 2 and double
// b                3     double
// n                4     integer
// m                5     integer
// Outputs:
// c is in position 1 and double with size [m,n]
c=fort("foof",a,2,"d",b,3,"d",n,4,"i",m,5,"i","out",[m,n],1,"d");
```

returns the matrix `c=2*a+b`.

If your machine is a DEC Alpha, SUN Solaris or SGI you may have to change the previous command line `link("foo.o","foo")` by one of the followings:

```
link('foof.o -lfor -lm -lc','foof').
link('foof.o -lftn -lm -lc','foof').
link('foof.o -L/opt/SUNWspro/SC3.0/lib/lib77 -lm -lc','foof').
```

The same example coded in C:

```
```

```
  void fooc(c,a,b,m,n)
 double a[],*b,c[];
 int *m,*n;
      { double sin();
 int i;
 for ( i =0 ; i < (*m)*(*n) ; i++)
        c[i] = sin(a[i]) + *b;
 }

link("fooc.o","fooc","C") // note the third argument
a=[1,2,3;4,5,6];b= %pi;
[m,n]=size(a);
c=fort("fooc",a,2,"d",b,3,"d",m,4,"i",n,5,"i","out",[m,n],1,"d");
```

See Also

link , c_link , intersci , addinter

Name

getdynlibext — get the extension of dynamic libraries on your operating system.

```
ret=getdynlibext()
```

Description

get the extension of dynamic libraries on your operating system.

ret=getdynlibext() returns (.so on linux,.sl HP-UX,.dll on Windows, ...).

Examples

```
getdynlibext()
```

Authors

Allan CORNET

Name

haveacompiler — detect if you have a C compiler.

```
bOK=haveacompiler()
```

Parameters

bOK
: returns %T if you have a C compiler.

Description

detect if you have a C compiler.

Examples

```
bOK = haveacompiler();
```

See Also

findlcccompiler , findmsvccompiler

Name

ilib_build — utility for shared library management

```
ilib_build(lib_name,table,files,libs [,makename,ldflags,cflags,fflags,ismex, cc
```

Parameters

lib_name
: a character string, the generic name of the library without path and extension.

table
: 2 column string matrix giving the table of pairs 'scilab-name', 'interface name'

files
: string matrix giving source (from Scilab 5.0) or object files needed for shared library creation

libs
: string matrix giving extra libraries needed for shred library creation

makename
: character string. The path of the Makefile file without extension.

ldflags,cflags,fflags
: character strings to provide options for the loader, the C compiler and the Fortran compiler.

ismex
: Internal variable to specify if we are working with mex or not.

cc
: Provide the name of the C compiler.

Description

This tool is used to create shared libraries and to generate a loader file which can be used to dynamically load the shared library into Scilab with `addinter`

Many examples are provided in `SCI/modules/dynamic_link/examples` directory.

Note that a compiler must be available on the system to use this function.

Examples

```
//Here with give a complete example on adding new primitive to Scilab
//create the procedure files
f1=['extern double fun2();'
    'void fun1(double *x, double *y)'
    '{*y=fun2(*x)/(*x);}'];

mputl(f1,'fun1.c')

f2=['#include <math.h>'
    'double fun2(double x)'
    '{ return( sin(x+1.));}'];
mputl(f2,'fun2.c');
```

```
//creating the interface file
i=['#include ""stack-c.h""'
 '#include ""stackTypeVariable.h""'
   'extern int fun1 ( double *x, double *y);'
   'int intfun1(char *fname)'
   '{'
   '  int m1,n1,l1;'
   '  CheckRhs(1,1);'
   '  CheckLhs(1,1);'
   '  GetRhsVar(1, MATRIX_OF_DOUBLE_DATATYPE, &m1, &n1, &l1);'
   '  fun1(stk(l1),stk(l1));'
   '  LhsVar(1) = 1;'
   '  return 0;'
   '}'];
mputl(i,'intfun1.c')

//creating the shared library (a gateway, a Makefile and a loader are
//generated.

files=['fun1.c','fun2.c','intfun1.c'];
ilib_build('foo',['scifun1','intfun1'],files,[]);

// load the shared library

exec loader.sce

//using the new primitive
scifun1(33)
```

See Also

addinter , link , ilib_compile , ilib_gen_Make , ilib_gen_gateway , ilib_gen_loader , ilib_for_link

Name

ilib_compile — ilib_build utility: executes the Makefile produced by ilib_gen_Make

```
libn=ilib_compile(lib_name,makename [,files,ldflags,cflags,fflags,cc])
```

Parameters

lib_name
: a character string, the generic name of the library without path and extension.

makename
: character string. The path of the Makefile file without extension.

files
: optionnal vector of character strings. If files is given the make is performed on each target contained in files then a whole make is performed

libn
: character string. The path of the actual generated shared library file.

ldflags,cflags,fflags,cc
: character strings to provide options/flags for the loader, the C compiler, the Fortran compiler. cc provides the name of the compiler.

Description

Utility function used by `ilib_build`

This executes the Makefile produced by `ilib_gen_Make`, compiles the C and fortran files and generates the shared library.

Shared libraries can then be used with the `link` and `addinter` Scilab function for incremental/ dynamic link.

Note that a compiler must be available on the system to use this function.

See Also

addinter , link , ilib_build , ilib_gen_Make , ilib_gen_gateway , ilib_gen_loader , ilib_for_link

Name

ilib_for_link — utility for shared library management with link

```
libn=ilib_for_link(names,files,libs,flag [,makename [,loadername [,libname [,ld
```

Parameters

names
: a string matrix giving the entry names which are to be linked.

files
: string matrix giving objects files needed for shared library creation

libs
: string matrix giving extra libraries needed for shred library creation

flag
: a string flag ("c" or "f") for C or Fortran entry points.

makename
: character string. The pathname of the Makefile file without extension (default value `Makelib`).

loadername
: character string. The pathname of the loader file (default value is `loader.sce`).

libname
: optional character string. The name of the generated shared library (default value is '', and in this case the name is derived from `names(1)`).

ldflags
: optional character string. It can be used to add specific linker options in the generated Makefile. Default value is ''

cflags
: optional character string. It can be used to add specific C compiler options in the generated Makefile. Default value is ''

fflags
: optional character string. It can be used to add specific Fortran compiler options in the generated Makefile. Default value is ''

cc
: optional character string. It can be used to specify a C compiler. Default value is ''

libn
: character string. The path of the really generated shared library file.

Description

This tool is used to create shared libraries and to generate a loader file which can be used to dynamically load the shared library into Scilab with the `link` function. New entry points given by `names` are then accessible through the `call` function or with non linear tools `ode`, `optim`,...

The file to compile are supposed to be located given by `makename`. If `makename` sets a path different to the current directory, `loader` script must be located in the same directory using the `loadername` variable.

Many examples are provided in `examples/link-examples-so` directory.

Note that a compiler must be available on the system to use this function.

Examples

```
if haveacompiler() then

chdir(TMPDIR)
f1=['int ext1c(int *n, double *a, double *b, double *c)'
    '{int k;'
    '  for (k = 0; k < *n; ++k) '
    '      c[k] = a[k] + b[k];'
    '  return(0);}'];

mputl(f1,'fun1.c')

//creating the shared library (a gateway, a Makefile and a loader are
//generated.

ilib_for_link('ext1c','fun1.o',[],"c")

// load the shared library

exec loader.sce

//using the new primitive
a=[1,2,3];b=[4,5,6];n=3;
c=call('ext1c',n,1,'i',a,2,'d',b,3,'d','out',[1,3],4,'d');
if norm(c-(a+b)) > %eps then pause,end

end
```

See Also

addinter , link , ilib_compile , ilib_gen_Make , ilib_gen_gateway , ilib_gen_loader , ilib_for_link

Name

ilib_gen_Make — utility for ilib_build: produces a Makefile for building shared libraries

```
Makename=ilib_gen_Make(name,files,libs,makename [,with_gateway,ldflags,cflags,f
```

Parameters

lib_name
: a character string, the generic name of the library without path and extension.

files
: a vector of character string. The names of the C or Fortran files without the extension and the path part.

libs
: a vector of character string. additionnal libraries paths or [].

makename
: character string. The path of the Makefile file.

with_gateway
: a boolean. If true a file with name <lib_name>_gateway is added. Default value is %t

ldflags
: a string. It can be used to add specific linker options in the generated Makefile. Default value is "

cflags
: a string. It can be used to add specific C compiler options in the generated Makefile. Default value is "

fflags
: a string. It can be used to add specific Fortran compiler options in the generated Makefile. Default value is "

cc
: a string. The name of the C compiler. Default value is the C compiler detected on the host.

Makename
: character string. The path of the really generated Makefile file.

Description

Utility function used by `ilib_build`

This function generates a Makefile adapted to the Operating System for building shared libraries to be loaded in Scilab. Proper options and paths are set.

Shared libraries can then be used with the `link` and `addinter` scilab function for incremental/ dynamic linking.

The shared library is build from a set of C or Fortran routines stored in a directory and if required from a set of external libraries.

Files are not required to exist, when Makefile is generated, but of course are required for executing the Makefile.

Only use this function is you know what you are doing (it is a semi-private function).

See Also

addinter , link , ilib_build , ilib_compile , ilib_gen_gateway , ilib_gen_loader , ilib_for_link

Name

ilib_gen_gateway — utility for ilib_build, generates a gateway file.

```
ilib_gen_gateway(name,table)
```

Parameters

name
: a character string, the generic name of the library without path and extension.

table
: 2 column string matrix giving the table of pairs 'scilab-name' 'interface name'

Description

Utility function used by `ilib_build` This function generates a gateway file used by `addinter`.

See Also

addinter , link , ilib_build , ilib_compile , ilib_gen_Make , ilib_gen_loader , ilib_for_link

Name

ilib_gen_loader — utility for ilib_build: generates a loader file

```
ilib_gen_loader(name,table)
```

Parameters

name
: a character string, the generic name of the library without path and extension.

table
: 2 column string matrix giving the table of pairs 'scilab-name' 'interface name'

Description

Utility function used by `ilib_build` This function generates a loader file.

See Also

addinter , link , ilib_build , ilib_compile , ilib_gen_Make , ilib_gen_loader , ilib_for_link

Name

ilib_mex_build — utility for mex library management

```
ilib_mex_build(lib_name,table,files,libs [,makename,ldflags,cflags,fflags,cc])
```

Parameters

lib_name
: a character string, the generic name of the library without path and extension.

table
: 3 column string matrix giving the table of 'scilab-name', 'interface name', 'cmex' or 'fmex'

files
: string matrix giving objects files needed for shared library creation

libs
: string matrix giving extra libraries needed for shred library creation

makename
: character string. The path of the Makefile file without extension.

ldflags,cflags,fflags,cc
: character strings to provide options/flags for the loader, the C compiler, the Fortran compiler. cc provides the name of the compiler.

Description

This function is used to create mex libraries and to generate a loader file which can be used to dynamically load the mex shared library.

Note that the file name containing the mex code can be set in the third input argument (`files`) or the second value of the `table` input argument.

Note that a compiler must be available on the system to use this function.

Examples

```
cd(TMPDIR);

mputl([
'#include ""mex.h""'
'void mexFunction(int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[])'
'{'
'  int *dims = mxGetDimensions(prhs[0]);'
'  sciprint(""%d %d %d\n"",dims[0],dims[1],dims[2]);'
'}'
],'mexfunction16.c');
ilib_mex_build('libmex',['mexf16','mexfunction16','cmex'],[],[],'Makelib','','

exec(TMPDIR+'/loader.sce');
mexf16(rand(2,3,2));
```

See Also

addinter , link , ilib_compile , ilib_gen_Make , ilib_gen_gateway , ilib_gen_loader , ilib_for_link

Name

link — dynamic linker

```
x=link(files [, sub-names,flag]);
link(x , sub-names [, flag]);
ulink(x)
lst=link('show')
lst=link()
```

Parameters

files
: a character string or a vector of character strings, the files names used to define the new entry point (compiled routines, user libraries, system libraries,..)

sub-names
: a character string or a vector of character strings . Name of the entry points in `files` to be linked.

x
: an integer which gives the id of a shared library linked into Scilab with a previous call to `link`.

flag
: character string `'f'` or `'c'` for Fortran (default) or C code.

Description

`link` is a incremental/dynamic link facility: this command allows to add new compiled Fortran or C routines to Scilab executable code. Linked routines can be called interactively by the function `call`. Linked routines can also be used as "external" for e.g. non linear problem solvers (`ode`, `optim`, `intg`, `dassl`...).

`link()` returns a string matrix with linked functions.

a call to `link` returns an integer which gives the id of the shared library which is loaded into Scilab. This number can then be used as the first argument of the link function in order to link additional function from the linked shared library. The shared library is removed with the `ulink` command.

A routine can be unlinked with `ulink`. If the linked function has been modified between two links, it is required to `ulink` the previous instance before the new link.

`link('show')` returns the current linked routines.

To be able to link routines in a system independent way, it is convenient to use the `ilib_for_link` utility function instead of `link`.

(Experienced) users may also `link` a new Scilab interface routine to add a set of new functions. See ilib_build and addinter functions.

Number of 'link' in a scilab session can be limited by the operating system. On Windows, you cannot load more than 80 dynamic libraries at the same time.

Examples

```
//Example of the use of ilib_for_link with  a simple C code
f1=['#include <math.h>'
'void fooc(double c[],double a[],double *b,int *m,int *n)'
```

```
'int *m,*n;'
'{'
'   int i;'
'   for ( i =0 ; i < (*m)*(*n) ; i++) '
'     c[i] = sin(a[i]) + *b; '
'}'];

mputl(f1,'fooc.c')

//creating the shared library: a Makefile and a loader are
//generated, the code is compiled and a shared library built.
ilib_for_link('fooc','fooc.o',[],"c")

// display the loader.sce file which calls link
mprintf('%s\n',mgetl('loader.sce'))
// load the shared library
exec loader.sce

link('show')
// call the new linked entry point
a=linspace(0,%pi,10);b=5;
y1=call('fooc',a,2,'d',b,3,'d',size(a,1),4,'i',size(a,2),5,'i','out',size(a
// check
y1-(sin(a)+b)
```

See Also

call, external, c_link, addinter, ilib_for_link, ilib_build

Name

ulink — unlink a dynamically linked shared object

```
ulink(x)
ulink()
```

Description

see `link`

See Also

link

Name

with_lcc — returns if LCC-Win32 is the default C Compiler.

```
bOK=with_lcc()
```

Parameters

bOK
: returns %T if LCC-Win32 is the default C Compiler.

Description

checks if LCC-Win32 is the default C Compiler.

Examples

```
bOK=with_lcc()
```

Authors

Allan CORNET

Elementary Functions

Name

abs — absolute value, magnitude

```
t=abs(x)
```

Parameters

x
: real or complex vector or matrix

t
: real vector or matrix

Description

`abs(x)` is the absolute value of the elements of `x`. When `x` is complex, `abs(x)` is the complex modulus (magnitude) of the elements of `x`.

Examples

```
abs([1,%i,-1,-%i,1+%i])
```

Name

acos — element wise cosine inverse

```
t=acos(x)
```

Parameters

x
: real or complex vector

t
: real or complex vector

Description

The components of vector `t` are cosine inverse of the corresponding entries of vector `x`. Definition domain is [-1, 1].

Examples

```
x=[1,%i,-1,-%i]
cos(acos(x))
```

Name

acosd — element wise cosine inverse, result in degree.

```
t=acosd(x)
```

Parameters

x
: Real or complex array.

t
: Real or complex array.

Description

The components of vector `t` are cosine inverse, in degree, of the corresponding entries of vector `x` (t=acos(x)*180/%pi). For real data in `[-1, 1]`. The results are real in `[0 180].`

Examples

```
x=[-1 0 1 sqrt(2)/2 -sqrt(2)/2 sqrt(3)/2 -sqrt(3)/2];
acosd(x)
```

See Also

acos

Name

acosh — hyperbolic cosine inverse

```
[t]=acosh(x)
```

Parameters

x
: real or complex vector

t
: real or complex vector

Description

the components of vector `t` are the ArgCosh of the corresponding entries of vector `x`. Definition domain is]1,+infinity[.

Examples

```
x=[0,1,%i];
cosh(acosh(x))
```

Name

acoshm — matrix hyperbolic inverse cosine

```
t=acoshm(x)
```

Parameters

x,t
: real or complex square matrix

Description

acoshm is the matrix hyperbolic inverse cosine of the matrix x. Uses the formula `t=logm(x+(x+eye())*sqrtm((x-eye())/(x+eye())))` For non symmetric matrices result may be inaccurate.

Examples

```
A=[1,2;3,4];
coshm(acoshm(A))
A(1,1)=A(1,1)+%i;
coshm(acoshm(A))
```

See Also

acosh, logm, sqrtm

Name

acosm — matrix wise cosine inverse

```
t=acosm(x)
```

Parameters

x
: real or complex square matrix

t
: real or complex square matrix

Description

`t` are cosine inverse of the `x` matrix. Diagonalization method is used. For nonsymmetric matrices result may be inaccurate.One has `t=-%i*logm(x+%i*sqrtm(eye()-x*x))`

Examples

```
A=[1,2;3,4];
cosm(acosm(A))
```

See Also

acos, sqrtm, logm

Name

acot — computes the element-wise inverse cotangeant of the argument.

```
y = acot(x)
```

Parameters

x
: Real or complex array.

y
: Real or complex array.

Description

Computes the element-wise inverse cotangeant of the argument. For real argument the result is real.

The following equalities hold: `acot(z) = %pi-acot(-z)=%pi/2-atan(z)= %i*acoth(%i*z)+%pi/2*(1-csgn(z+%i))`

Examples

```
x=[1 2 -2 sqrt(2) -sqrt(2) 2/sqrt(3) -2/sqrt(3) -1];
acot(x)/%pi
```

See Also

cotg, acotd

References

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joing IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

Authors

Serge Steer, INRIA

Name

acotd — computes the element-wise inverse cotangeant of the argument result in degree.

```
y = acotd(x)
```

Parameters

x
: Real or complex array.

y
: Real or complex array.

Description

Computes the element-wise inverse cotangeant of the argument and returns the results in degree. For real argument w the result is real.

Examples

```
x=[1 2 -2 sqrt(2) -sqrt(2) 2/sqrt(3) -2/sqrt(3) -1];
acotd(x)
```

See Also

cotd, acot

References

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joing IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

Authors

Serge Steer, INRIA

Name

acoth — element wise hyperbolic cotangeant inverse.

```
y = acoth(x)
```

Parameters

x
: Real or complex array.

y
: Real or complex array.

Description

Computes the element wise hyperbolic cotangeant inverse of the argument.

Examples

```
x=-30:0.1:30;
x(abs(x)<=1)=%nan;
plot(x,acoth(x))
```

See Also

atanh, coth

Authors

Serge Steer, INRIA

Used Functions

this function is based on the `atanh` function.

Name

acsc — computes the element-wise inverse cosecant of the argument.

```
y = acsc(x)
```

Parameters

x
: Real or complex array.

y
: Real or complex array.

Description

Computes the element-wise inverse cotsecant of the argument. For real argument with absolute value greater than 1 the result is real.

The following equalities hold: `acsc(z) = -acsc(-z) = asin(1/z) = %pi/2-asec(z) = %i*acsch(%i*z)`

Examples

```
x=linspace(1,20,200);
x=[-x($:-1:1) %nan x];
plot(x,acsc(x))
```

See Also

csc, acscd

References

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joing IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

Authors

Serge Steer, INRIA

Name

acscd — computes the element-wise inverse cosecant of the argument, results in degree.

```
y = acscd(x)
```

Parameters

x
: Real array.

y
: Real or complex array.

Description

Computes the element-wise inverse cosecant of the argument an return the results in degree. For real argument with absolute value greater than 1 the result is real.

Examples

```
x=linspace(1,20,200);
x=[-x($:-1:1) %nan x];
plot(x,acscd(x))
```

See Also

cscd, acsc

References

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joing IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

Authors

Serge Steer, INRIA

Name

acsch — computes the element-wise inverse hyperbolic cosecant of the argument.

```
y = acsch(x)
```

Parameters

x
: Real or complex array.

y
: Real or complex array.

Description

Computes the element-wise inverse hyperbolic cotsecant of the argument. For real argument with absolute value greater than 1 the result is real.

The following equalities hold: `acsch(z) = -acsch(-z) = asinh(1/z) = csgn(%i+1/z)*asech(-i*z)-%i*%pi/2 = %i*acsc(%i*z)`

Examples

```
x=linspace(1,20,200);
x=[-x($:-1:1) %nan x];
plot(x,acsch(x))
```

See Also

csch

References

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joing IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

Authors

Serge Steer, INRIA

Name

adj2sp — converts adjacency form into sparse matrix.

Parameters

xadj
: integer vector of length (n+1).

adjncy
: integer vector of length nz containing the row indices for the corresponding elements in anz

anz
: column vector of length nz, containing the non-zero elements of A

mn
: row vector with 2 entries, `mn=size(A)` (optional).

A
: real or complex sparse matrix (nz non-zero entries)

Description

```
sp2adj converts an adjacency form representation of a matrix
into its standard Scilab representation (utility fonction).
xadj, adjncy, anz = adjacency representation of A i.e:
```

`xadj(j+1)-xadj(j)` = number of non zero entries in row j. `adjncy` = column index of the non zeros entries in row 1, row 2,..., row n. `anz` = values of non zero entries in row 1, row 2,..., row n. `xadj` is a (column) vector of size n+1 and `adjncy` is an integer (column) vector of size `nz=nnz(A)`. `anz` is a real vector of size `nz=nnz(A)`.

Examples

```
A = sprand(100,50,.05);
[xadj,adjncy,anz]= sp2adj(A);
[n,m]=size(A);
p = adj2sp(xadj,adjncy,anz,[n,m]);
A-p
```

See Also

sp2adj, spcompack

Name

amell — Jacobi's am function

```
[sn]=amell(u,k)
```

Parameters

u
: real scalar or vector

k
: scalar

sn
: real scalar or vector

Description

Computes Jacobi's elliptic function `am(u,k)` where `k` is the parameter and `u` is the argument. If `u` is a vector `sn` is the vector of the (element wise) computed values. Used in function `%sn`.

See Also

delip , %sn , %asn

Name

and — (&) logical and

```
b=and(A), b=and(A,'*')
b=and(A,'r'), b=and(A,1)
b=and(A,'c'), b=and(A,2)
A&B
```

Description

`and(A)` is the logical AND of elements of the boolean matrix `A`. `and(A)` returns `%T` ("true") iff all entries of `A` are `%T`.

`y=and(A,'r')` (or, equivalently, `y=and(A,1)`) is the rowwise and. It returns in each entry of the row vector `y` the and of the rows of `x` (The and is performed on the row index : `y(j)= and(A(i,j),i=1,m)`).

`y=and(A,'c')` (or, equivalently, `y=and(A,2)`) is the columnwise and. It returns in each entry of the column vector `y` the and of the columns of `x` (The and is performed on the column index: `y(i)= and(A(i,j),j=1,n)`)).

`A&B` gives the element-wise logical `and` of the booleans matrices `A` and `B` .`A` and `B` must be matrices with the same dimensions or one from them must be a single boolean.

See Also

not , or

Name

asec — computes the element-wise inverse secant of the argument.

```
y = asec(x)
```

Parameters

x
: Real or complex array.

y
: Real or complex array.

Description

Computes the element-wise inverse secant of the argument. For real argument with absolute value greater than 1 the result is real.

The following equalities hold: `asec(z) = -acsc(-z) = asin(1/z) = %pi/2-asec(x) = %i*acsch(%i*z)`

Examples

```
x=[1 2 -2 sqrt(2) -sqrt(2) 2/sqrt(3) -2/sqrt(3) -1];
asec(x)/%pi
```

See Also

sec, asecd

References

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joing IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

Authors

Serge Steer, INRIA

Name

asecd — computes the element-wise inverse secant of the argument, results in degree.

```
y = asecd(x)
```

Parameters

x
: Real or complex array

y
: Real or complex array

Description

Computes the element-wise inverse secant of the argument, results in degree. For real input data with absolute value greater than `1` the results are real and in `[-90 90]`.

`asecd(x)` is equal to `asec(x)*180/%pi`.

Examples

```
x=[1 2 -2 sqrt(2) -sqrt(2) 2/sqrt(3) -2/sqrt(3) -1];
asecd(x)
```

See Also

asec, secd

References

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joing IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

Authors

Serge Steer, INRIA

Name

asech — computes the element-wise inverse hyperbolic secant of the argument.

```
y = asech(x)
```

Parameters

x
: Real or complex array.

y
: Real or complex array.

Description

Computes the element-wise inverse hyperbolic secant of the argument. If the argument is real and have absolute value less than one the result is real.

The following equalities hold: `asech(x) = acosh(1 ./ x)= %i*csgn(%i*(1#1 ./ x))*asec(x)=csgn(%i*(1 # 1 ./ x))*(%pi/2*(%i+acsch(%i*x)))`

Examples

```
asech(1)
```

See Also

sech

References

Kahan, W., "Branch cuts for complex elementary functions, or, Much ado about nothing's sign bit", Proceedings of the joing IMA/SIAM conference on The State of the Art in Numerical Analysis, University of Birmingham, A. Iserles and M.J.D. Powell, eds, Clarendon Press, Oxford, 1987, 165-210.

Authors

Serge Steer, INRIA

Name

asin — sine inverse

```
[t]=asin(x)
```

Parameters

x
: real or complex vector/matrix

t
: real or complex vector/matrix

Description

The entries of `t` are sine inverse of the corresponding entries of `x`. Definition domain is [-1, 1].

Examples

```
A=[1,2;3,4]
sin(asin(A))
```

See Also

sin, sinm, asinm

Name

asind — sine inverse, results in degree

```
t=asind(x)
```

Parameters

x
: real vector/matrix. Elements must be in `[-1 1]`.

t
: real vector/matrix with same dimensions as `x`

Description

The entries of `t` are sine inverse of the corresponding entries of `x`. Definition domain is [-1, 1]. The results are in [-90 90];

Examples

```
x=[-1 0 1 sqrt(2)/2 -sqrt(2)/2 sqrt(3)/2 -sqrt(3)/2];
asind(x)
```

See Also

sin, sind, asinm

Name

asinh — hyperbolic sine inverse

```
[t]=asinh(x)
```

Parameters

x
: real or complex vector/matrix

t
: real or complex vector/matrix

Description

The entries of `t` are the hyperbolic sine inverse of the corresponding entries of `x`. Definition domain is]-1,i[.

Examples

```
A=[1,2;2,3]
sinh(asinh(A))
```

Name

asinhm — matrix hyperbolic inverse sine

```
t=asinhm(x)
```

Parameters

x,t
: real or complex square matrix

Description

asinhm is the matrix hyperbolic inverse sine of the matrix x. Uses the formula `t=logm(x +sqrtm(x*x+eye()))`. Results may be not reliable for non-symmetric matrix.

Examples

```
A=[1,2;2,3]
sinhm(asinhm(A))
```

See Also

asinh, logm, sqrtm

Name

asinm — matrix wise sine inverse

```
t=asinm(x)
```

Parameters

x
: real or complex square matrix

t
: real or complex square matrix

Description

`t` are sine inverse of the `x` matrix. Diagonalization method is used. For non symmetric matrices result may be inaccurate.

Examples

```
A=[1,2;3,4]
sinm(asinm(A))
asinm(A)+%i*logm(%i*A+sqrtm(eye()-A*A))
```

See Also

asin, sinm

Name

atan — 2-quadrant and 4-quadrant inverse tangent

```
phi=atan(x)
phi=atan(y,x)
```

Parameters

x
: real or complex scalar, vector or matrix

phi
: real or complex scalar, vector or matrix

x, y
: real scalars, vectors or matrices of the same size

phi
: real scalar, vector or matrix

Description

The first form computes the 2-quadrant inverse tangent, which is the inverse of `tan(phi)`. For real `x`, `phi` is in the interval (-pi/2, pi/2). For complex `x`, `atan` has two singular, branching points `+%i`,`-%i` and the chosen branch cuts are the two imaginary half-straight lines [i, i*oo) and (-i*oo, -i].

The second form computes the 4-quadrant arctangent (atan2 in Fortran), this is, it returns the argument (angle) of the complex number `x+i*y`. The range of `atan(y,x)` is (-pi, pi].

For real arguments, both forms yield identical values if `x>0`.

In case of vector or matrix arguments, the evaluation is done element-wise, so that `phi` is a vector or matrix of the same size with `phi(i,j)=atan(x(i,j))` or `phi(i,j)=tan(y(i,j),x(i,j))`.

Examples

```
// examples with the second form
x=[1,%i,-1,%i]
phasex=atan(imag(x),real(x))
atan(0,-1)
atan(-%eps,-1)

// branch cuts
atan(-%eps + 2*%i)
atan(+%eps + 2*%i)
atan(-%eps - 2*%i)
atan(+%eps - 2*%i)

// values at the branching points
ieee(2)
atan(%i)
atan(-%i)
```

See Also

tan, ieee

Authors

B.P.
L.V.D. (authors of the complex atan function).

Name

atand — 2-quadrant and 4-quadrant element-wise inverse tangent, result in degree.

```
phi=atand(x)
phi=atand(y,x)
```

Parameters

x
: real scalar, vector or matrix

phi
: real scalar, vector or matrix

x, y
: real scalars, vectors or matrices of the same size

phi
: real scalar, vector or matrix

Description

The first form computes the 2-quadrant inverse tangent, which is the inverse of `tand(phi)`. The `phi` elements are in the interval [-90, 90].

The second form computes the 4-quadrant arctangent (atan2 in Fortran), this is, it returns the argument (angle) of the complex number `x+i*y`. The range of `atand(y,x)` is [-180,180i].

Both forms yield identical values if `x>0`.

Examples

```
// example with the second form
x=[0,1/sqrt(3),1,sqrt(3),%inf,0]
atand(x)
```

See Also

tan, tand

Authors

Serge Steer, INRIA

Name

atanh — hyperbolic tangent inverse

```
t=atanh(x)
```

Parameters

x
: real or complex vector/matrix

t
: real or complex vector/matrix

Description

The components of vector `t` are the hyperbolic tangent inverse of the corresponding entries of vector `x`. Definition domain is `[-1,1]` for the real function (see Remark).

Remark

In Scilab (as in some others numerical software) when you try to evaluate an elementary mathematical function outside its definition domain in the real case, then the complex extension is used (with a complex result). The more famous example being the sqrt function (try `sqrt(-1)` !). This approach have some drawbacks when you evaluate the function at a singular point which may led to different results when the point is considered as real or complex. For the `atanh` this occurs for `-1` and `1` because the at these points the imaginary part do not converge and so `atanh(1) = +Inf + i NaN` while `atanh(1) = +Inf` for the real case (as lim x->1- of atanh(x)). So when you evaluate this function on the vector `[1 2]` then like `2` is outside the definition domain, the complex extension is used for all the vector and you get `atanh(1) = +Inf + i NaN` while you get `atanh(1) = +Inf` with `[1 0.5]` for instance.

Examples

```
// example 1
x=[0,%i,-%i]
tanh(atanh(x))

// example 2
x = [-%inf -3 -2 -1 0 1 2 3 %inf]
ieee(2)
atanh(tanh(x))

// example 3 (see Remark)
ieee(2)
atanh([1 2])
atanh([1 0.5])
```

See Also

tanh, ieee

Name

atanhm — matrix hyperbolic tangent inverse

```
t=atanhm(x)
```

Parameters

x
: real or complex square matrix

t
: real or complex square matrix

Description

`atanhm(x)` is the matrix hyperbolic tangent inverse of matrix `x`. Results may be inaccurate if `x` is not symmetric.

Examples

```
A=[1,2;3,4];
tanhm(atanhm(A))
```

See Also

atanh, tanhm

Name

atanm — square matrix tangent inverse

```
[t]=atanm(x)
```

Parameters

x
: real or complex square matrix

t
: real or complex square matrix

Description

`atanm(x)` is the matrix arctangent of the matrix x. Result may be not reliable if `x` is not symmetric.

Examples

```
tanm(atanm([1,2;3,4]))
```

See Also

atan

Name

base2dec — conversion from base b representation to integers

```
d=base2dec(s,b)
```

Parameters

d
: matrix of integers

s
: matrix of character strings corresponding to base b representation

b
: integer, the base

Description

`base2dec(x,b)` returns the matrix of numbers corresponding to the base `b` representation.

Examples

```
base2dec(['ABC','0','A'],16)
```

See Also

bin2dec, oct2dec, hex2dec, dec2bin, dec2oct, dec2hex

Name

bin2dec — integer corresponding to a binary form

```
y=bin2dec(str)
```

Parameters

str
: a string or a vector/matrix of strings containing only characters '1' and '0'

y
: a scalar or a vector/matrix of positives integers

Description

Given `str` a binary string, this function returns `y` the decimal number whose the binary representation is given by `str` (`y` and `str` have the same size).

Examples

```
// example 1 :
// '1010110' : is the binary representation of 86
str='1010110';
y=bin2dec(str)

// example 2 :
// '1011011' : is the binary representation of 91
// '1010010' : is the binary representation of 82
str=['1011011'; '1010010']
y=bin2dec(str)
```

See Also

base2dec, oct2dec, hex2dec, dec2bin, dec2oct, dec2hex

Name

binomial — binomial distribution probabilities

```
pr=binomial(p,n)
```

Parameters

pr
: row vector with n+1 components

p
: real number in [0,1]

n
: an integer >= 1

Description

`pr=binomial(p,n)` returns the binomial probability vector, i.e. `pr(k+1)` is the probability of `k` success in `n` independent Bernouilli trials with probability of success `p`. In other words : `pr(k+1) = probability(X=k)` , with X a random variable following the B(n,p) distribution, and numerically :

$$\mathrm{pr}(k+1) = \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k} \text{with} \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Examples

```
// first example
n=10;p=0.3; xbasc(); plot2d3(0:n,binomial(p,n));

// second example
n=50;p=0.4;
mea=n*p; sigma=sqrt(n*p*(1-p));
x=( (0:n)-mea )/sigma;
xbasc()
plot2d(x, sigma*binomial(p,n));
deff('y=Gauss(x)','y=1/sqrt(2*%pi)*exp(-(x.^2)/2)')
plot2d(x, Gauss(x), style=2);

// by binomial formula (Caution if big n)
function pr=binomial2(p,n)
x=poly(0,'x');pr=coeff((1-p+x)^n).*horner(x^(0:n),p);
endfunction
p=1/3;n=5;
binomial(p,n)-binomial2(p,n)

// by Gamma function: gamma(n+1)=n! (Caution if big n)
p=1/3;n=5;
Cnks=gamma(n+1)./(gamma(1:n+1).*gamma(n+1:-1:1));
x=poly(0,'x');
pr=Cnks.*horner(x.^(0:n).*(1-x)^(n:-1:0),p);
pr-binomial(p,n)
```

See Also

cdfbin, grand

Name

bitand — AND applied to binary representation of input arguments

```
[z]=bitand(x,y)
```

Parameters

x
: scalar/vector/matrix/hypermatrix of positives integers

y
: scalar/vector/matrix/hypermatrix of positives integers

z
: scalar/vector/matrix/hypermatrix of positives integers

Description

Given `x` and `y` two positives integers, this function returns `z` the decimal number whose the binary form is the AND of the binary representations of `x` and `y`. (`x`, `y`, `z` have the same size. If dimension of `x` (and `y`) is superior than 1 then `z(i)` is equal to `bitand(x(i),y(i)).`

Examples

```
// example 1 :
// '1010110' : is the binary representation of 86
// '1011011' : is the binary representation of 91
// '1010010' : is the binary representation for the AND of binary representatio
// so the decimal number corresponding to the AND  applied to binary forms 86 a
x=86; y=91
z=bitand(x,y)


// example 2 :
x=[12,45],y=[25,49]
z=bitand(x,y)
```

See Also

bitor, bin2dec, dec2bin

Name

bitor — OR applied to binary representation of input arguments

```
[z]=bitor(x,y)
```

Parameters

x
: scalar/vector/matrix/hypermatrix of positives integers

y
: scalar/vector/matrix/hypermatrix of positives integers

z
: scalar/vector/matrix/hypermatrix of positives integers

Description

Given `x` and `y` two positives integers, this function returns `z` the decimal number whose the binary form is the OR of the binary representations of `x` and `y` (`x`, `y` and `z` have the same size). If dimension of `x` is superior than 1 then `z(i)` is equal to `bitor(x(i),y(i))`.

Examples

```
// example 1 :
// '110000' : is the binary representation of 48
// '100101' : is the binary representation of 37
// '110101' : is the binary representation for the OR applied to the binary form
// so the decimal number corresponding to the OR  applied to binary forms 48 and
x=48; y=37
z=bitor(x,y)

// example 2 :
x=[12,45]; y=[25,49]
z=bitor(x,y)
```

See Also

bitand, bin2dec, dec2bin

Name

bloc2exp — block-diagram to symbolic expression

```
[str]=bloc2exp(blocd)
[str,names]=bloc2exp(blocd)
```

Parameters

blocd
: list

str
: string

names
: string

Description

given a block-diagram representation of a linear system `bloc2exp` returns its symbolic evaluation. The first element of the list `blocd` must be the string `'blocd'`. Each other element of this list `(blocd(2),blocd(3),...)` is itself a list of one the following types :

```
list('transfer','name_of_linear_system')
```

```
list('link','name_of_link',
               [number_of_upstream_box,upstream_box_port],
               [downstream_box_1,downstream_box_1_portnumber],
               [downstream_box_2,downstream_box_2_portnumber],
               ...)
```

The strings `'transfer'` and `'links'` are keywords which indicate the type of element in the block diagram.

Case 1 : the second parameter of the list is a character string which may refer (for a possible further evaluation) to the Scilab name of a linear system given in state-space representation (`syslin` list) or in transfer form (matrix of rationals).

To each transfer block is associated an integer. To each input and output of a transfer block is also associated its number, an integer (see examples)

Case 2 : the second kind of element in a block-diagram representation is a link. A link links one output of a block represented by the pair `[number_of_upstream_box,upstream_box_port]`, to different inputs of other blocks. Each such input is represented by the pair `[downstream_box_i,downstream_box_i_portnumber]`.

The different elements of a block-diagram can be defined in an arbitrary order.

For example

[1] `S1*S2` with unit feedback.

There are 3 transfers `S1` (number `n_s1=2`) , `S2` (number `n_s2=3`) and an adder (number `n_add=4`) with symbolic transfer function `['1','1']`.

There are 4 links. The first one (named 'U') links the input (port 0 of fictitious block -1, omitted) to port 1 of the adder. The second and third one link respectively (output)port 1 of the adder to (input)port 1 of system S1, and (output)port 1 of S1 to (input)port 1 of S2. The fourth link (named 'Y') links (output)port 1 of S2 to the output (port 0 of fictitious block -1, omitted) and to (input)port 2 of the adder.

```
//Initialization
syst=list('blocd'); l=1;
//
//Systems
l=l+1;n_s1=l;syst(l)=list('transfer','S1');  //System 1
l=l+1;n_s2=l;syst(l)=list('transfer','S2');  //System 2
l=l+1;n_adder=l;syst(l)=list('transfer',['1','1']);  //adder
//
//Links
// Inputs  -1 --> input 1
l=l+1;syst(l)=list('link','U',[-1],[n_adder,1]);
// Internal
l=l+1;syst(l)=list('link',' ',[n_adder,1],[n_s1,1]);
l=l+1;syst(l)=list('link',' ',[n_s1,1],[n_s2,1]);
// Outputs // -1 -> output 1
l=l+1;syst(l)=list('link','Y',[n_s2,1],[-1],[n_adder,2]);
//Evaluation call
w=bloc2exp(syst);
```

The result is the character string: w=-(s2*s1-eye())\s2*s1.

Note that invoked with two output arguments, [str,names]= blocd(syst) returns in names the list of symbolic names of named links. This is useful to set names to inputs and outputs.

[2] second example

```
//Initialization
syst=list('blocd'); l=1;

//System (2x2 blocks plant)
l=l+1;n_s=l;syst(l)=list('transfer',['P11','P12';'P21','P22']);

//Controller
l=l+1;n_k=l;syst(l)=list('transfer','k');

//Links
l=l+1;syst(l)=list('link','w',[-1],[n_s,1]);
l=l+1;syst(l)=list('link','z',[n_s,1],[-1]);
l=l+1;syst(l)=list('link','u',[n_k,1],[n_s,2]);
l=l+1;syst(l)=list('link','y',[n_s,2],[n_k,1]);

//Evaluation call
w=bloc2exp(syst);
```

In this case the result is a formula equivalent to the usual one:

```
P11+P12*invr(eye()-K*P22)*K*P21;
```

See Also

bloc2ss

Authors

S. S., F. D. (INRIA)

Name

bloc2ss — block-diagram to state-space conversion

```
[sl]=bloc2ss(blocd)
```

Parameters

blocd
: list

sl
: list

Description

Given a block-diagram representation of a linear system `bloc2ss` converts this representation to a state-space linear system. The first element of the list `blocd` must be the string `'blocd'`. Each other element of this list is itself a list of one the following types :

```
list('transfer','name_of_linear_system')
```

```
list('link','name_of_link',
             [number_of_upstream_box,upstream_box_port],
             [downstream_box_1,downstream_box_1_portnumber],
             [downstream_box_2,downstream_box_2_portnumber],
             ...)
```

The strings `'transfer'` and `'links'` are keywords which indicate the type of element in the block diagram.

Case 1 : the second parameter of the list is a character string which may refer (for a possible further evaluation) to the Scilab name of a linear system given in state-space representation (`syslin` list) or in transfer form (matrix of rationals).

To each transfer block is associated an integer. To each input and output of a transfer block is also associated its number, an integer (see examples)

Case 2 : the second kind of element in a block-diagram representation is a link. A link links one output of a block represented by the pair `[number_of_upstream_box,upstream_box_port]`, to different inputs of other blocks. Each such input is represented by the pair `[downstream_box_i,downstream_box_i_portnumber]`.

The different elements of a block-diagram can be defined in an arbitrary order.

For example

[1] `S1*S2` with unit feedback.

There are 3 transfers `S1` (number `n_s1=2`) , `S2` (number `n_s2=3`) and an adder (number `n_add=4`) with symbolic transfer function `['1','1']`.

There are 4 links. The first one (named `'U'`) links the input (port 0 of fictitious block -1, omitted) to port 1 of the adder. The second and third one link respectively (output)port 1 of the adder to (input)port

1 of system S1, and (output)port 1 of S1 to (input)port 1 of S2. The fourth link (named 'Y') links (output)port 1 of S2 to the output (port 0 of fictitious block -1, omitted) and to (input)port 2 of the adder.

```
//Initialization
syst=list('blocd'); l=1;

//Systems
l=l+1;n_s1=l;syst(l)=list('transfer','S1');  //System 1
l=l+1;n_s2=l;syst(l)=list('transfer','S2');  //System 2
l=l+1;n_adder=l;syst(l)=list('transfer',['1','1']);  //adder

//Links
// Inputs  -1 --> input 1
l=l+1;syst(l)=list('link','U1',[-1],[n_adder,1]);

// Internal
l=l+1;syst(l)=list('link',' ',[n_adder,1],[n_s1,1]);
l=l+1;syst(l)=list('link',' ',[n_s1,1],[n_s2,1]);

// Outputs // -1 -> output 1
l=l+1;syst(l)=list('link','Y',[n_s2,1],[-1],[n_adder,2]);
```

With s=poly(0,'s');S1=1/(s+1);S2=1/s; the result of the evaluation call sl=bloc2ss(syst); is a state-space representation for 1/(s^2+s-1).

[2] LFT example

```
//Initialization
syst=list('blocd'); l=1;

//System (2x2 blocks plant)
l=l+1;n_s=l;syst(l)=list('transfer',['P11','P12';'P21','P22']);

//Controller
l=l+1;n_k=l;syst(l)=list('transfer','k');

//Links
l=l+1;syst(l)=list('link','w',[-1],[n_s,1]);
l=l+1;syst(l)=list('link','z',[n_s,1],[-1]);
l=l+1;syst(l)=list('link','u',[n_k,1],[n_s,2]);
l=l+1;syst(l)=list('link','y',[n_s,2],[n_k,1]);
```

With

```
P=syslin('c',A,B,C,D);
P11=P(1,1);
P12=P(1,2);
P21=P(2,1);
P22=P(2,2);
K=syslin('c',Ak,Bk,Ck,Dk);
```

bloc2exp(syst) returns the evaluation the lft of P and K.

See Also

bloc2exp

Authors

S. S., F. D. (INRIA)

Name

cat — concatenate several arrays

```
y=cat(dims,A1,A2,...,An)
```

Parameters

dims
: a positive real scalar.

A1,A2,..An
: scalars, vectors, matrices or mutlti-arrays, or cells arrays. A1,A2,...,An must have the same size (excluding the dimension number dims). size(A1,i)=size(A2,i)=...=size(An,i) for i different of dims and size(A1,dims), size(A2,dims),...,size(An,dims) can be different.

y
: a scalar, vector, matrix or mutlti-array, y has the same type than A1,A2,...,An.

Description

`y=cat(dims,A1,A2,...,An)` : y is the result of the concatenation of the input arguments A1,A2,...,An. if dims=1then the concatenation is done according to the rows : if dims=2 then concatenation is done according to the columns of the input arguments,...

if dims=1, then the concatenation is done according to the rows

A1=[1 2 3 ; 4 5 6]; A2=[7 8 9 ; 10 11 12]; y=cat(1,A1,A2) => y=[1 2 3 ; 4 5 61 ;7 8 9; 10 11 12]

if dims=2, then the concatenation is done according to the columns of the input arguments

A1=[1 2 3;4 5 6]; A2=[7 8 9 ;10 11 12]; y=cat(2,A1,A2) => y=[1 2 3 7 8 9 ; 4 5 6 10 11 12]

Examples

```
// first example : concatenation according to the rows
dims=1; A1=[1 2 3]; A2=[4 5 6 ; 7 8 9]; A3=[10 11 12]; y=cat(dims,A1,A2,A3)

// second example :  concatenation according to the columns
dims=2; A1=[1 2 3]'; A2=[4 5;7 8;9 10]; y=cat(dims,A1,A2)

// third example : concatenation according to the 3th dimension
dims=3; A1=matrix(1:12,[2,2,3]); A2=[13 14;15 16]; A3=matrix(21:36,[2,2,4]); y=
```

See Also

permute, matrix

Authors

Farid Belahcene

Name

ceil — rounding up

```
[y]=ceil(x)
```

Parameters

x
: a real matrix

y
: integer matrix

Description

`ceil(x)` returns an integer matrix made of rounded up elements.

Examples

```
ceil([1.9 -2.5])-[2,-2]
ceil(-%inf)
x=rand()*10^20;ceil(x)-x
```

See Also

round, floor, int

Name

cell2mat — convert a cell array into a matrix

```
x=cell2mat(c)
```

Parameters

c
: cell, the components of c must have the same type and can be scalars or matrices

x
: matrix

Description

Returns a matrix which is the concatenation of all components of the cell c.

cell2mat(c)
: all components of c must have the same data type (strings or doubles or integers or booleans). For each row `i` of `c`, `cell2mat` concatenates all the components of the `ith` row of the cell `c`

Note that if the components of the cell input c are strings then `cell2mat(c)` returns a column vector of strings concatenation.

Examples

```
c=makecell([2,2],[1 2 3; 6 7 8],[4 5;9 10],[11 12;16 17],[14 13 15;18 19 20])
cell2mat(c)
```

See Also

cell

Name

cellstr — convert strings vector (or strings matrix) into a cell of strings

```
c=cellstr(s)
```

Parameters

s
: strings vector, or strings matrix

Description

returns a cell array of strings

- If `s` is a line vector of strings then `cellstr(s)` returns a `(one-by-one)` cell which contains one component (the concatenation of all columns components of `s`)
- If `s` is a column vector of strings then `cellstr(s)` convert s into a cell which have the same size : `(size(s,1)-by-one)` cell of strings
- If `s` is a matrix of strings then for each row `i` of `s`, `cellstr(s)` concatenates all the components of the ith rows of the matrix `s` (i.e s(i,1), s(i,2), s(i,3),...) and returns a `(size(s,1)-by-one)` cell of strings

Examples

```
cellstr("foo")
cellstr(["sci","lab"])
cellstr(["abc","def",'gh';"i","j","klm"])
```

See Also

cell, string

Name

char — char function

```
y=char( x)
y=char(st1,st2,st3,....)
```

Parameters

x
: a cell of strings arrays, or an array of ascii codes

st1,st2,st3
: strings arrays

y:
: a strings vector(column)

Description

`One input argument :`

Given `a cell of strings arrays x`, this function returns a strings vector `y` in which the rows are the components of the strings cell

Given `an array of ascii codes x,` this function returns a strings array `y` corresponding into ascii codes. If dims (`x`)=[n1,n2,n3,n4,....],then returned value have same size as input value instead of second dims, dims(`y`)=[n1,n3,n4,..]

`More than one input argument :`

Given `strings arrays st1,st2,st3,...`, this function returns a vector of strings in which the rows are the components of `st1,st2,st3,... In the vector y` the length of all strings sti is completed by blanks, to have the same length than the lengthmax of sti.

Examples

```
//Example with a hypermatrix of ascii codes :
x=hypermat([4,2,3],61:84);
y=char(x)
size(x)
size(y)

//Example with more than one argument :
st1="zeros";
st2=["one","two"];
st3=["three"];
y=char(st1,st2,st3)
size(y)

//all strings rows are completed by 'blanks' to have the same length : 6
length(y)
```

See Also

asciimat

Authors

F.B

Name

conj — conjugate

```
[y]=conj(x)
```

Parameters

x,y
: real or complex matrix.

Description

`conj(x)` is the complex conjugate of `x`.

Examples

```
x=[1+%i,-%i;%i,2*%i];
conj(x)
x'-conj(x)  //x'  is conjugate transpose
```

Name

cos — cosine function

```
[y]=cos(x)
```

Parameters

x
: real or complex vector/matrix

Description

For a vector or a matrix, `cos(x)` is the cosine of its elements . For matrix cosine use `cosm(X)` function.

Examples

```
x=[0,1,%i]
acos(cos(x))
```

See Also

cosm

Name

cosd — element-wise cosine function, argument in degree

```
y=cosd(x)
```

Parameters

x
: real vector/matrix

Description

For a vector or a matrix `x` of angles given in degree, `cosd(x)` is the cosine of its elements. The results are in `[-1 1]`. For input elements which are equal to `n*90` with `n` integer and odd, the result is exactly zero.

Examples

```
x=[0,30 45 60 90 360];
cosd(x)
```

See Also

cos

Authors

Serge Steer, INRIA

Name

cosh — hyperbolic cosine

```
[t]=cosh(x)
```

Parameters

x,t
: real or complex vectors/matrices

Description

The elements of `t` are the hyperbolic cosine of the corresponding entries of vector `x`.

Examples

```
x=[0,1,%i]
acosh(cosh(x))
```

See Also

cos, acosh

Name

coshm — matrix hyperbolic cosine

```
t=coshm(x)
```

Parameters

x,t
: real or complex square matrix

Description

coshm is the matrix hyperbolic cosine of the matrix x. `t=(expm(x)+expm(-x))/2`. Result may be inaccurate for nonsymmetric matrix.

Examples

```
A=[1,2;2,4]
acoshm(coshm(A))
```

See Also

cosh, expm

Name

cosm — matrix cosine function

```
t=cosm(x)
```

Parameters

x
: real or complex square matrix

Description

`cosm(x)` is the matrix cosine of the x matrix. `t=0.5*(expm(%i*x)+expm(-%i*x))`.

Examples

```
A=[1,2;3,4]
cosm(A)-0.5*(expm(%i*A)+expm(-%i*A))
```

See Also

cos, expm

Name

cotd — cotangent

```
y=cotd(x)
```

Parameters

x
: Real array.

y
: Real array with same dimensions as `x`.

Description

The entries of `y` are the cotangents of the corresponding entries of `x` supposed to be given in degree. `t=cos(x)./sin(x)`. For entries equal to `n*180` with `n` integers the results are infinite, whereas `cotg(n*%pi)` is large but finite because `%pi` cannot be represented exactly. For entries equal to `n*90` with `n` integers and odd the results are exactly 0.

Examples

```
x=[30 45 60 90];
cotd(x)
```

See Also

cotg

Authors

Serge Steer, INRIA

Name

cotg — cotangent

```
[t]=cotg(x)
```

Parameters

x,t
: real or complex vectors/matrices

Description

The elements of `t` are the cotangents of the corresponding entries of `x`. `t=cos(x)./sin(x)`

Examples

```
x=[1,%i];
cotg(x)-cos(x)./sin(x)
```

See Also

tan

Name

coth — hyperbolic cotangent

```
[t]=coth(x)
```

Description

the elements of vector `t` are the hyperbolic cotangent of elements of the vector `x`.

Examples

```
x=[1,2*%i]
t=exp(x);
(t-ones(x)./t).\(t+ones(x)./t)
coth(x)
```

See Also

cotg

Name

cothm — matrix hyperbolic cotangent

```
[t]=cothm(x)
```

Description

`cothm(x)` is the matrix hyperbolic cotangent of the square matrix x.

Examples

```
A=[1,2;3,4];
cothm(A)
```

See Also

coth

Name

csc — Computes the element-wise cosecant of the argument.

```
y = csc(x)
```

Parameters

x
: Real or complex array.

y
: Real or complex array with same dimensions as `x`.

Description

Computes the element-wise cosecant of the argument. The cosecant is a periodic function defined as 1/sin. For real data the results are real and in `]-%inf -1] U [1 %inf[`.

Examples

```
x=linspace(0.01,%pi-0.01,200)
clf();
plot(-x,csc(-x),x,csc(x))
```

See Also

sec, cscd

Authors

Serge Steer, INRIA

Name

cscd — Computes the element-wise cosecant of the argument given in degree.

```
x = cscd(x)
```

Parameters

x
: Real or complex array.

x
: Real or complex array.

Description

The entries of `y` are the cosecant `1/sin` of the entries of `x` given in degree. The results are real and in `]-%inf -1] U [1 %inf[`. or entries equal to `n*180` with `n` integer, the result is infinite (or an error depending on ieee mode). For entries equal to `n*90`with `n` integer and odd the result is exactly 1 or -1.

Examples

```
csc(%pi/4)
cscd(90)
```

See Also

secd, csc, sind

Authors

Serge Steer, INRIA

Name

csch — Computes the element-wise hyperbolic cosecant of the argument.

```
y = csch(x)
```

Parameters

x
: Real or complex array.

y
: Real or complex array with same dimensions as `x`.

Description

Computes the element-wise hyperbolic cosecant of the argument. For real data the results are real.

Examples

```
x=linspace(0.01,4,200);x=[-x($:-1:1) %nan x];
clf();
plot(x,csch(x))
```

See Also

csc, acsch

Authors

Serge Steer, INRIA

Name

cumprod — cumulative product

```
y=cumprod(x)
y=cumprod(x,'r') or y=cumprod(x,1)
y=cumprod(x,'c') or y=cumprod(x,2)
y=cumprod(x,'m')
```

Parameters

x
: vector or matrix (real or complex)

y
: vector or matrix (real or complex)

Description

For a vector or a matrix `x`, `y=cumprod(x)` returns in `y` the cumulative product of all the entries of `x` taken columnwise.

`y=cumprod(x,'c')` (or, equivalently, `y=cumprod(x,2)`) returns in `y` the cumulative elementwise product of the columns of `x`: `y(i,:)=cumprod(x(i,:))`

`y=cumprod(x,'r')` (or, equivalently, `y=cumprod(x,2)`) returns in `y` the cumulative elementwise product of the rows of `x`: `y(:,i)=cumprod(x(:,i))`.

`y=cumprod(x,'m')` is the cumulative product along the first non singleton dimension of `x` (for compatibility with Matlab).

Examples

```
A=[1,2;3,4];

cumprod(A)
cumprod(A,'r')
cumprod(A,'c')

rand('seed',0);
a=rand(3,4);
[m,n]=size(a);
w=zeros(a);
w(1,:)=a(1,:);
for k=2:m;
  w(k,:)=w(k-1,:).*a(k,:);
end;
w-cumprod(a,'r')
```

See Also

cumsum, sum, prod

Name

cumsum — cumulative sum

```
y=cumsum(x)
y=cumsum(x,'r') or y=cumsum(x,1)
y=cumsum(x,'c') or y=cumsum(x,2)
```

Parameters

x
: vector or matrix (real or complex)

y
: vector or matrix (real or complex)

Description

For a vector or a matrix `x`, `y=cumsum(x)` returns in `y` the cumulative sum of all the entries of `x` taken columnwise.

`y=cumsum(x,'c')` (or, equivalently, `y=cumsum(x,2)`) returns in `y` the cumulative sum of the columns of `x`: `y(i,:)=cumsum(x(i,:))`

`y=cumsum(x,'r')` (or, equivalently, `y=cumsum(x,1)`) returns in `y` the cumulative sum of the rows of `x`: `y(:,i)=cumsum(x(:,i))`

`y=cumsum(x,'m')` is the cumulative sum along the first non singleton dimension of `x` (for compatibility with Matlab).

Examples

```
A=[1,2;3,4];

cumsum(A)
cumsum(A,'r')
cumsum(A,'c')

a=rand(3,4)+%i;
[m,n]=size(a);
w=zeros(a);
w(1,:)=a(1,:);
for k=2:m;
  w(k,:)=w(k-1,:)+a(k,:);
end;
w-cumsum(a,'r')
```

See Also

cumprod, sum

Name

dec2bin — binary representation

```
[str]=dec2bin(x[,n])
```

Parameters

x
: scalar/vector/matrix/hypermatrix of positives integers

n
: a positive integer

str
: a string or a vector of string

Description

Given `x`, a positive (or a vector/matix of integers) integer, this function returns a string (or a column vector of strings) which is the binary representation of `x.` If dimension of `x` is superior than 1 then each component of the colums vector `str` is the binary representation of the `x` components (i.e `str(i)` is the binary representation of `x(i)`). If the components length of `str` is less than `n` (i.e length str(i) < n), then add to `str` components the characters '0' on the left in order to have componants length equal to `n.`

Examples

```
// example 1 :
x=86;
str=dec2bin(x)

// example 2 :
// the binary representation of 86 is: '1010110'
// its length is 7(less than n), so we add to str, 8 times the character '0'  (
x=86;n=15;
str=dec2bin(x,n)

// example 3 :
x=[12;45;135]
z=dec2bin(x)
```

See Also

base2dec, bin2dec, oct2dec, hex2dec, dec2oct, dec2hex

Name

dec2hex — hexadecimal representation of integers

```
h=dec2hex(d)
```

Parameters

d
: matrix of non negative integers

h
: matrix of character strings

Description

`dec2hex(x)` returns the hexadecimal representation of a matrix of integers.

Examples

```
dec2hex([2748 10;11 3])
```

See Also

base2dec, bin2dec, oct2dec, hex2dec, dec2bin, dec2oct

Name

dec2oct — octal representation of integers

```
o=dec2oct(d)
```

Parameters

d
: matrix of non negative integers

o
: matrix of character strings

Description

`dec2oct(x)` returns the octal representation of a matrix of integers.

Examples

```
dec2oct([2748 10;11 3])
```

See Also

base2dec, bin2dec, oct2dec, hex2dec, dec2bin, dec2hex

Name

delip — complete and incomplete elliptic integral of first kind

```
[r]=delip(x,ck)
```

Parameters

x
: real vector with non negative elements

ck
: real number between -1 and 1

r
: real or complex number (or vector) with the same size as `x`

Description

The elliptic integral of the first kind with parameter `ck` is defined as follow:

$$\int_0^x \frac{dt}{\sqrt{(1-t^2)(1-c_k^2 t^2)}}$$

Where `x` is real and positive, `ck` is in `[-1 1]`.

If x is less than 1 the result is real.

When called with `x` a vector `r` is evaluated for each entry of `x`.

Examples

```
ck=0.5;
delip([1,2],ck)
deff('y=f(t)','y=1/sqrt((1-t^2)*(1-ck^2*t^2))')
intg(0,1,f)    //OK since real solution!
```

See Also

amell, %asn, %sn

Name

diag — diagonal including or extracting

```
[y]=diag(vm, [k])
```

Parameters

vm
: vector or matrix (full or sparse storage)

k
: integer (default value 0)

y
: vector or matrix

Description

for `vm` a (row or column) n-vector `diag(vm)` returns a diagonal matrix with entries of `vm` along the main diagonal.

`diag(vm,k)` is a `(n+abs(k))x(n+abs(k))` matrix with the entries of `vm` along the kth diagonal. `k=0` is the main diagonal `k>0` is for upper diagonals and `k<0` for lower diagonals.

For a matrix `vm`, `diag(vm,k)` is the column vector made of entries of the kth diagonal of `vm`. `diag(vm)` is the main diagonal of `vm`. `diag(diag(x))` is a diagonal matrix.

If `vm` is a sparse matrix `diag(vm,k)` returns a sparse matrix.

To construct a diagonal linear system, use `sysdiag`.

Note that `eye(A).*A` returns a diagonal matrix made with the diagonal entries of `A`. This is valid for any matrix (constant, polynomial, rational, state-space linear system,...).

Examples

```
diag([1,2])

A=[1,2;3,4];
diag(A)  // main diagonal
diag(A,1)

diag(sparse(1:10))  // sparse diagonal matrix

// form a tridiagonal matrix of size 2*m+1
m=5;diag(-m:m) +  diag(ones(2*m,1),1) +diag(ones(2*m,1),-1)
```

See Also

sysdiag, sparse

Name

diff — Difference and discrete derivative

```
y=diff(x)
y=diff(x [,n [,dim]])
```

Parameters

x
: vector or matrix (real, complex, sparse or polynomial)

n
: integer the order of differentiation

dim
: integer or character string with values "r","c" and "*"

y
: scalar or vector

Description

`y=diff(x)` compute the difference function y=x(2:$)-x(1:$-1)

`diff(x,n,dim)` is the `n` th difference function along dimension `dim`.

`diff(x,n,"*")` is equivalent to diff(x(:),n).

Default value for `n` is 1. Default value for `dim` is `"*"`.

`dim='r'` is equivalent to `dim=1` and `dim='c'` is equivalent to `dim=2`.

Examples

```
v=(1:8)^3;
diff(v)
diff(v,3)

A=[(1:8)^2
   (1:8)^3
   (1:8)^4];

diff(A,3,2)

//approximate differentiation
step=0.001
t=0:step:10;
y=sin(t);
dy=diff(sin(t))/step; //approximate differentiation of sine function
norm(dy-cos(t(1:$-1)),%inf)
```

Name

double — conversion from integer to double precision representation

```
y=double(X)
y=int16(X)
y=int32(X)
y=uint8(X)
y=uint16(X)
y=uint32(X)
```

Parameters

X
: matrix of floats or integers

y
: matrix of floats

Description

converts data stored using one, two or four bytes integers into double precision floating point representation. If X entries are already double precision floats, nothing is done.

Examples

```
x=int8([0 12 140])
double(x)
```

See Also

int8, inttype, type

Name

dsearch — binary search (aka dichotomous search in french)

```
[ind, occ, info] = dsearch(X, val [, ch ])
```

Parameters

X
: a real vector or matrix

val
: a real (row or column) vector with n components in strictly increasing order val(1) < val(2) < ... < val(n)

ch
: (optional) a character "c" or "d" (default value "c")

ind
: a real vector or matrix with the same dimensions than X

occ
: a real vector with the same format than val (but with n-1 components in the case ch="c")

info
: integer

Description

This function is useful to search in an ordered table and/or to count the number of components of a vector falling in some classes (a class being an interval or a value).

By default or when `ch="c"`, this is the interval case, that is, for each X(i) search in which of the n-1 intervals it falls, the intervals being defined by:

```
I1 = [val(1), val(2)]
Ik = (val(k), val(k+1)] for 1 < k <= n-1 ;
```

and:

ind(i)
: is the interval number of X(i) (0 if X(i) is not in [val(1),val(n)])

occ(k)
: is the number of components of X which are in Ik

info
: is the number of components of X which are not in [val(1),val(n)]

When `ch="d"` case, this is the discrete case, that is, for each X(i) search if it is equal to one val(k) and:

ind(i)
: is equal to the index of the component of val which matches X(i) (ind(i) = k if X(i)=val(k)) or 0 if X(i) is not in val.

occ(k)
: is the number of components of X equal to val(k)

info
is the number of components of X which are not in the set {val(1),...,val(n)}

Examples

```
// example #1 (elementary stat for U(0,1))
m = 50000 ; n = 10;
X = grand(m,1,"def");
val = linspace(0,1,n+1)';
[ind, occ] = dsearch(X, val);
xbasc() ; plot2d2(val, [occ/m;0])  // no normalisation : y must be near 1/n


// example #2 (elementary stat for B(N,p))
N = 8 ; p = 0.5; m = 50000;
X = grand(m,1,"bin",N,p); val = (0:N)';
[ind, occ] = dsearch(X, val, "d");
Pexp = occ/m; Pexa = binomial(p,N);
xbasc() ; hm = 1.1*max(max(Pexa),max(Pexp));
plot2d3([val val+0.1], [Pexa' Pexp],[1 2],"111",  ...
        "Pexact@Pexp", [-1 0 N+1 hm],[0 N+2 0 6])
xtitle(  "binomial distribution B("+string(N)+","+string(p)+") :" ...
        +" exact probability versus experimental ones")


// example #3 (piecewise Hermite polynomial)
x = [0 ; 0.2 ; 0.35 ; 0.5 ; 0.65 ; 0.8 ;  1];
y = [0 ; 0.1 ;-0.1  ; 0   ; 0.4  ;-0.1 ;  0];
d = [1 ; 0   ; 0    ; 1   ; 0    ; 0   ; -1];
X = linspace(0, 1, 200)';
ind = dsearch(X, x);

// define Hermite base functions
deff("y=Ll(t,k,x)","y=(t-x(k+1))./(x(k)-x(k+1))")   // Lagrange left on Ik
deff("y=Lr(t,k,x)","y=(t-x(k))./(x(k+1)-x(k))")     // Lagrange right on Ik
deff("y=Hl(t,k,x)","y=(1-2*(t-x(k))./(x(k)-x(k+1))).*Ll(t,k,x).^2")
deff("y=Hr(t,k,x)","y=(1-2*(t-x(k+1))./(x(k+1)-x(k))).*Lr(t,k,x).^2")
deff("y=Kl(t,k,x)","y=(t-x(k)).*Ll(t,k,x).^2")
deff("y=Kr(t,k,x)","y=(t-x(k+1)).*Lr(t,k,x).^2")

// plot the curve
Y = y(ind).*Hl(X,ind) + y(ind+1).*Hr(X,ind) + d(ind).*Kl(X,ind) + d(ind+1).*Kr(
xbasc(); plot2d(X,Y,2) ; plot2d(x,y,-9,"000")
xtitle("an Hermite piecewise polynomial")
// NOTE : you can verify by adding these ones :
// YY = interp(X,x,y,d); plot2d(X,YY,3,"000")
```

See Also

find, tabul

Authors

B.P.

Name

eval — evaluation of a matrix of strings

```
[H]= eval(Z)
```

Description

returns the evaluation of the matrix of character strings Z.

Examples

```
a=1; b=2; Z=['a','sin(b)'] ; eval(Z)  //returns the matrix [1,0.909];
```

See Also

evstr, execstr

Name

exp — element-wise exponential

```
exp(X)
```

Parameters

X
: scalar,vector or matrix with real or complex entries.

Description

`exp(X)` is the (element-wise) exponential of the entries of `X`.

Examples

```
x=[1,2,3+%i];
log(exp(x))  //element-wise
2^x
exp(x*log(2))
```

See Also

coff, log, expm

Name

eye — identity matrix

```
X=eye(m,n)
X=eye(A)
X=eye()
```

Parameters

A,X
: matrices or `syslin` lists

m,n
: integers

Description

according to its arguments defines an `mxn` matrix with 1 along the main diagonal or an identity matrix of the same dimension as `A` .

Caution: `eye(10)` is interpreted as `eye(A)` with `A=10` i.e. `1`. (It is NOT a ten by ten identity matrix!).

If `A` is a linear system represented by a `syslin` list, `eye(A)` returns an `eye` matrix of appropriate dimension: (number of outputs x number of inputs).

`eye()` produces a identity matrix with undefined dimensions. Dimensions will be defined when this identity matrix is added to a matrix with fixed dimensions.

Examples

```
eye(2,3)
A=rand(2,3);eye(A)
s=poly(0,'s');A=[s,1;s,s+1];eye(A)
A=[1/s,1;s,2];eye(A);
A=ssrand(2,2,3);eye(A)
[1 2;3 4]+2*eye()
```

See Also

ones, zeros

Name

factor — factor function

```
[y]=factor(x)
```

Parameters

x
: real scalar

y
: vector of primes numbers

Description

Given a real `x`, `factor(x)` returns in a vector `y` the primes numbers decomposition of `x`. Particular cases: `factor(0)` returns `0`, and `factor(1)` returns `1`.

Examples

```
x=620
y=factor(x)
```

See Also

primes

Name

fix — rounding towards zero

```
[y]=fix(x)
```

Parameters

x
: a real matrix

y
: integer matrix

Description

`fix(x)` returns an integer matrix made of nearest rounded integers toward zero,i.e, `y=sign(x).*floor(abs(x))`. Same as `int`.

See Also

round , floor , ceil

Name

flipdim — flip x components along a given dimension

```
y=flipdim(x,dim)
```

Parameters

x
: a scalar, a vector or an array of reals

dim
: a positive integer

y
: a scalar, a vector or an array of reals

Description

Given `x`, a scalar/vector/array of reals and `dim` a positive integer, this function flips the x components along the dimension number `dim` of `x` (`x` and `y` have the same size)

Examples

```
// example 1: flip x components along the first dimension
x=[1 2 3 4; 5 6 7 8];
dim=1;
y=flipdim(x,dim)

// example 2: flip x components along the second dimension
dim=2;
y=flipdim(x,dim)

// example 3: flip x components along the third dimension
x=matrix(1:48,[3 2,4,2]);
dim=3;
y=flipdim(x,dim)
```

Authors

F.Belahcene

Name

floor — rounding down

```
[y]=floor(x)
```

Parameters

x
: a real matrix

y
: integer matrix

Description

`floor(x)` returns an integer matrix made of nearest rounded down integers.

Examples

```
floor([1.9 -2.5])-[1,-3]
floor(-%inf)
x=rand()*10^20;floor(x)-x
```

See Also

round, fix, ceil

Name

frexp — dissect floating-point numbers into base 2 exponent and mantissa

```
[f,e]=frexp(x)
```

Parameters

x
: real vector or matrix

f
: array of real values, usually in the range 0.5 <= abs(f) < 1.

e
: array of integers that satisfy the equation: `x = f.*2.^e`

Description

This function corresponds to the ANSI C function frexp(). Any zeros in `x` produce `f=0` and `e=0`.

Examples

```
[f,e]=frexp([1,%pi,-3,%eps])
```

See Also

log, hat, ieee, log2

Name

gsort — sorting by quick sort agorithm

```
[B [,k]]=gsort(A)
[B [,k]]=gsort(A,option)
[B [,k]]=gsort(A,option,direction)
```

Parameters

A
: a real,an integer or a character string vector/matrix.

option
: a character string. It gives the type of sort to perform:

  - 'r' : each column of A is sorted
  - 'c': each row of A is sorted
  - 'g': all elements of A are sorted. It is the default value.
  - 'lr': lexicographic sort of the rows of A
  - 'lc': lexicographic sort of the columns of A

direction
: a character string. It gives the ordering direction: `'i'` stand for increasing and `'d'` for decreasing order.

B
: an array with same type and dimensions as A.

k
: a real array with integer values and same dimensions as A. Contains the origin indices.

Description

`gsort` implements a "quick sort" algorithm for various data types.

- `B=gsort(A,'g')`, `B=gsort(A,'g','d')` and `B=gsort(A)` sort the elements of the array A, seen as `A(:)` in a decreasing order.

  `B=gsort(A,'g','i')` sort the elements of the array A in the increasing order.

- `B=gsort(A,'lr')` sorts the rows of A in lexical decreasing order. B is obtained by a permutation of the rows of matrix A in such a way that the rows of B verify `B(i,:)>=B(j,:)` if `i<j`.

  `B=gsort(A,'lr','i')` works similarily for increasing lexical order.

- `B=gsort(A,'lc')` sorts the columns of A in lexical decreasing order. B is obtained by a permutation of the columns of matrix A in such a way that the columns of B verify `B(:,i)>=B(:,j)` if `i<j`.

  `B=gsort(A,'lc','i')` works similarily for increasing lexical order.

If required the second return argument k contains the indices of the sorted values in A. If `[B,k]=gsort(A,'g')` one has `B==A(k)`. **The algorithme preserve the relative order of records with equal values.**

When v is complex, the elements are sorted by magnitude, i.e., abs(v) . Only 'g' as second argument is managed with complex.

if v have %nan or %inf as elements. gsort places these at the beginning with 'i' or at the end with 'd' argument.

Examples

```
alr=[1,2,2;
     1,2,1;
     1,1,2;
     1,1,1];
[alr1,k]=gsort(alr,'lr','i')
[alr1,k]=gsort(alr,'lc','i')

v=int32(alr)

gsort(v)
gsort(v,'lr','i')
gsort(v,'lc','i')

v=['Scilab' '2.6'
   'Scilab' '2.7'
   'Scicos' '2.7'
   'Scilab' '3.1'
   'Scicos' '3.1'
   'Scicos' '4.0'
   'Scilab' '4.0']

gsort(v,'lr','i')
gsort(v,'lc','i')
```

See Also

find, sort

Bibliography

Quick sort algorithm from Bentley & McIlroy's "Engineering a Sort Function". Software---Practice and Experience, 23(11):1249-1265

http://www.enseignement.polytechnique.fr/profs/informatique/Jean-Jacques.Levy/poly/main2/node10.html

Name

hex2dec — conversion from hexadecimal representation to integers

```
d=hex2dec(h)
```

Parameters

d
: matrix of integers

h
: matrix of character strings corresponding to hexadecimal representation

Description

`hex2dec(x)` returns the matrix of numbers corresponding to the hexadecimal representation.

Examples

```
hex2dec(['ABC','0','A'])
```

See Also

base2dec, bin2dec, oct2dec, dec2bin, dec2oct, dec2hex

Name

imag — imaginary part

```
[y]=imag(x)
```

Parameters

x
: real or complex vector or matrix.

y
: real vector or matrix.

Description

`imag(x)` is the imaginary part of `x`. (See `%i` to enter complex numbers).

See Also

real

Name

imult — multiplication by i the imaginary unitary

```
y=imult(x)
```

Parameters

x
: real or complex scalar, vector or matrix

y
: complex scalar, vector or matrix

Description

`imult(x)` is a more efficient way to multiply x by i than y = %i*x, without the problems occuring when x comprises "special" floating point numbers as %inf and %nan.

Examples

```
z1 = imult(%inf)
z2 = %i * %inf
```

Authors

B.P.

Name

ind2sub — linear index to matrix subscript values

```
[i1,i2,..] =ind2sub(dims,I)
Mi = ind2sub(dims,I)
```

Parameters

dims
: vector: the matrix dimensions

I
: vector: the given linear index

i1,i2,..
: the subscript values (same matrix shape as `I`)

Mi
: matrix whose columns contains the subscript values.

Description

`ind2sub` is used to determine the equivalent subscript values corresponding to a given single index into an array. `[i1,i2,..] = ind2sub(dims,I)` returns the arrays `i1`, `i2`, ... containing the equivalent row, column, ... subscripts corresponding to the index matrix `I` for a matrix of size `dims`. `Mi=ind2sub(dims,I)` returns a matrix `Mi` whose columns are the arrays `i1(:)`, `i2(:)`, ...

Examples

```
ind2sub([2,3,2],1:12)
[i,j,k]=ind2sub([2,3,2],1:12)
```

See Also

sub2ind, extraction, insertion

Authors

Serge Steer, INRIA

Name

int — integer part

```
[y]=int(X)
```

Parameters

X
: real matrix

y
: integer matrix

Description

`int(X)` returns the integer part of the real matrix `X`. Same as `fix`.

See Also

round , floor , ceil

Name

intc — Cauchy integral

```
[y]=intc(a,b,f)
```

Parameters

a,b
: two complex numbers

f
: "external" function

Description

If `f` is a complex-valued function, `intc(a,b,f)` computes the integral from `a` to `b` of `f(z)dz` along the straight line `a b` of the complex plane.

See Also

intg , intl

Authors

F. D.

Name

integrate — integration of an expression by quadrature

```
x=integrate(expr,v,x0,x1 [,atol [,rtol]])
```

Parameters

expr
: Character string defining a Scilab expression.

v
: Character string, the integration variable name)

x0
: real number, the lower bound of integration

x1
: vector of real numbers, upper bounds of integration

atol
: real number (absolute error bound) Default value: 1.-8

rtol
: real number, (relative error bound) Default value: 1e-14

x
: vector of real numbers, the integral value for each `x1(i)`.

Description

computes : $x(i) = \int_{x_0}^{x1(i)} f(v)dv$ for `i=1:size(x1,'*')`

Where $f(v)$ is given by the expression `expr`.

The evaluation hopefully satisfies following claim for accuracy: `abs(I-x)<= max(atol,rtol*abs(I))` where `I` stands for the exact value of the integral.

Restriction

the given expression should not use variable names with a leading %.

Examples

```
x0=0;x1=0:0.1:2*%pi;
X=integrate('sin(x)','x',x0,x1);
norm(cos(x1)-(1-X))

 x1=-10:0.1:10;
X=integrate(['if x==0 then 1,';
             'else sin(x)/x,end'],'x',0,x1)
```

See Also

intg, inttrap, intsplin, ode

Name

interp1 — one_dimension interpolation function

```
[yp]=interp1(x, y,xp [, method,[extrapolation]])
```

Parameters

xp
: reals scalar, vector or matrix (or hypermatrix)

x
: reals vector

method
: (optional) string defining the interpolation method

extrapolation
: (optional) string, or real value defining the yp(j) components for xp(j) values outside [x1,xn] interval.

yp
: vector, or matrix (or hypermatrix)

Description

Given `(x,y,xp)`, this function performs the yp components corresponding to xp by the interpolation(linear by default) defined by x and y.

If `yp` is a vector then the length of xp is equal to the length of `yp,` if `yp` is a matrix then `xp` have the same length than the length of each columns of yp, if `yp` is a hypermatrix then the length of `xp` have the same length than the first dimension of `yp.`

If size(y)=[C,M1,M2,M3,...,Mj] and size(xp)=[N1,N2,N3,...,Nk] then size(yp)=[N1,N2,..,Nk,M1,M2,...Mj] and length of x must be equal to size(y,1)

The `method` parameter sets the evaluation rule for interpolation

"linear"
: the interpolation is defined by linear method (see interpln)

"spline"
: the interpolation is defined by cubic spline interpolation (see splin , interp)

"nearest"
: for each value xp(j), yp(j) takes the value or y(i) corresponding to x(i) the nearest neighbor of xp(j)

The `extrapolation` parameter sets the evaluation rule for extrapolation, i.e for `xp(i)` not in [x1,xn] interval

"extrap"
: the extrapolation is performed by the defined method. yp=interp1(x,y,xp,method,"extrap")

real value
: you can choose a real value for extrapolation, in this way yp(i) takes this value for xp(i) not in [x1,xn] interval, for example 0 (but also nan or inf). yi=interp1(x,y,xp,method, 0)

by default
: the extrapolation is performed by the defined method (for spline method), and by nan for linear and nearest methods. yp=interp1(x,y,xp,method)

Examples

```
x=linspace(0,3,20);
y=x^2;
xx=linspace(0,3,100);
yy1=interp1(x,y,xx,'linear');
yy2=interp1(x,y,xx,'spline');
yy3=interp1(x,y,xx,'nearest');
plot(xx,[yy1;yy2;yy3],x,y,'*')
xtitle('interpolation of square function')
legend(['linear','spline','nearest'],a=2)
```

See Also

interp, interpln, splin

Authors

F.B

Name

interp2d — bicubic spline (2d) evaluation function

```
[zp[,dzpdx,dzpdy[,d2zpdxx,d2zpdxy,d2zpdyy]]]=interp2d(xp,yp,x,y,C [,out_mode])
```

Parameters

xp, yp
: real vectors or matrices of same size

x,y,C
: real vectors defining a bicubic spline or sub-spline function (called `s` in the following)

out_mode
: (optional) string defining the evaluation of `s` outside [x(1),x(nx)]x[y(1),y(ny)]

zp
: vector or matrix of same format than `xp` and `yp`, elementwise evaluation of `s` on these points.

dzpdx, dzpdy
: vectors (or matrices) of same format than `xp` and `yp`, elementwise evaluation of the first derivatives of `s` on these points.

d2zpdxx, d2zpdxy, d2zpdyy
: vectors (or matrices) of same format than `xp` and `yp`, elementwise evaluation of the second derivatives of `s` on these points.

Description

Given three vectors `(x,y,C)` defining a bicubic spline or sub-spline function (see splin2d) this function evaluates *s* (and *ds/dx*, *ds/dy*, *d2s/dxx*, *d2s/dxy*, *d2s/dyy* if needed) at *(xp(i),yp(i))* :

$$zp(i) = s(xp(i),yp(i))$$

$$dzpdx(i) = \frac{\partial}{\partial x} s(xp(i),yp(i))$$

$$dzpdy(i) = \frac{\partial}{\partial y} s(xp(i),yp(i))$$

$$d2zpdxx(i) = \frac{\partial^2}{\partial x^2} s(xp(i),yp(i))$$

$$d2zpdxy(i) = \frac{\partial^2}{\partial x \partial y} s(xp(i),yp(i))$$

$$d2zpdyy(i) = \frac{\partial^2}{\partial y^2} s(xp(i),yp(i))$$

The `out_mode` parameter defines the evaluation rule for extrapolation, i.e. for *(xp(i),yp(i)) not in [x(1),x(nx)]x[y(1),y(ny)]*:

"by_zero"
: an extrapolation by zero is done

"by_nan"
: extrapolation by Nan

"C0"
: the extrapolation is defined as follows :

```
s(x,y) = s(proj(x,y)) where proj(x,y) is nearest point
                      of [x(1),x(nx)]x[y(1),y(ny)] from (x,y)
```

"natural"
: the extrapolation is done by using the nearest bicubic-patch from (x,y).

"periodic"
: s is extended by periodicity.

Examples

```
// see the examples of splin2d

// this example shows some different extrapolation features
// interpolation of cos(x)cos(y)
n = 7;  // a n x n interpolation grid
x = linspace(0,2*%pi,n); y = x;
z = cos(x')*cos(y);
C = splin2d(x, y, z, "periodic");

// now evaluate on a bigger domain than [0,2pi]x [0,2pi]
m = 80; // discretisation parameter of the evaluation grid
xx = linspace(-0.5*%pi,2.5*%pi,m); yy = xx;
[XX,YY] = ndgrid(xx,yy);
zz1 = interp2d(XX,YY, x, y, C, "C0");
zz2 = interp2d(XX,YY, x, y, C, "by_zero");
zz3 = interp2d(XX,YY, x, y, C, "periodic");
zz4 = interp2d(XX,YY, x, y, C, "natural");
xbasc()
subplot(2,2,1)
  plot3d(xx, yy, zz1, flag=[2 6 4])
  xtitle("extrapolation with the C0 outmode")
subplot(2,2,2)
  plot3d(xx, yy, zz2, flag=[2 6 4])
  xtitle("extrapolation with the by_zero outmode")
subplot(2,2,3)
  plot3d(xx, yy, zz3, flag=[2 6 4])
  xtitle("extrapolation with the periodic outmode")
subplot(2,2,4)
  plot3d(xx, yy, zz4, flag=[2 6 4])
  xtitle("extrapolation with the natural outmode")
xselect()
```

See Also

splin2d

Authors

B. Pincon

Name

intersect — returns the vector of common values of two vectors

```
[v [,ka,kb]]=intersect(a,b)
[v [,ka,kb]]=intersect(a,b,orient)
```

Parameters

a
: vector of numbers or strings

b
: vector of numbers or strings

orient
: flag with possible values : 1 or "r", 2 or "c"

v
: row vector of numbers or strings

ka
: row vector of integers

kb
: row vector of integers

Description

`intersect(a,b)` returns a sorted row vector of common values of two vectors of `a` and `b`.

`[v,ka,kb]=intersect(a,b)` also returns index vectors `ka` and `kb` such that `v=a(ka)` and `v=b(kb)`.

`intersect(a,b,"r")` or `intersect(a,b,1)` returns the matrix formed by the intersection of the unique rows of `a` and `b` sorted in lexicographic ascending order. In this case matrices `a` and `b` must have the same number of columns.

`[v,ka,kb]=intersect(a,b,"r")` also returns index vectors `ka` and `kb` such that `v=a(ka,:)` and `v=b(kb,:)`.

`intersect(a,b,"c")` or `intersect(a,b,2)` returns the matrix formed by the intersection of the unique columns of `a` and `b` sorted in lexicographic ascending order. In this case matrices `a` and `b` must have the same number of rows.

`[v,ka,kb]=intersect(a,b,"c")` also returns index vectors `ka` and `kb` such that `v=a(:,ka)` and `v=b(:,kb)`.

Remark

NaN are considered as different from themselves so they are excluded out of intersection in case of vector intersection.

Examples

```
A=round(5*rand(10,1));
B=round(5*rand(7,1));

intersect(A,B)
[N,ka,kb]=intersect(A,B)

intersect('a'+string(A),'a'+string(B))

intersect(int16(A),int16(B))

//with matrices
 A = [0,0,1,1 1;
      0,1,1,1,1;
      2,0,1,1,1;
      0,2,2,2,2;
      2,0,1,1,1;
      0,0,1,1,%nan];
B = [1,0,1;
     1,0,2;
     1,2,3;
     2,0,4;
     1,2,5;
     %nan,0,6];

[v,ka,kb] = intersect(A,B,'c')
A(:,ka)
```

See Also

unique, gsort, union

Name

intl — Cauchy integral

```
[y]=intl(a,b,z0,r,f)
```

Parameters

z0
: complex number

a,b
: two real numbers

r
: positive real number

f
: "external" function

Description

If `f` is a complex-valued function, `intl(a,b,z0,r,f)` computes the integral of `f(z)dz` along the curve in the complex plane defined by `z0 + r.exp(%i*t)` for `a<=t<=b` .(part of the circle with center `z0` and radius `r` with phase between `a` and `b`).

See Also

intc

Authors

F. D.

Name

inttrap — integration of experimental data by trapezoidal interpolation

```
v = inttrap([x,] s)
```

Parameters

x
: vector of increasing x coordinate data. Default value is `1:size(y,'*')`

s
: vector of y coordinate data

v
: value of the integral

Description

computes :

Where f is a function described by a set of experimental value:

`s(i)=f(x(i))` and `x0=x(1)`, `x1=x(n)`

Between mesh points function is interpolated linearly.

Examples

```
t=0:0.1:%pi
inttrap(t,sin(t))
```

See Also

intg, intc, intl, integrate, intsplin, splin

Name

isdef — checks variable existence

```
isdef(name [,where])
```

Parameters

name
: a character string

where
: an optional character string with default value `'all'`

Description

`isdef(name)` returns `%T` if the variable named`name` exists and `%F` otherwise.

Caveats: a function which uses `isdef` may return a result which depends on the environment!

`isdef(name,'local')` returns `%T` if the variable named`name` exists in the local environment of the current function and `%F` otherwise.

`isdef(name,'nolocal')` returns `%T` if the variable named `name` exists in the full calling environment (including the global level) of the current function and `%F` otherwise.

Examples

```
A=1;
isdef('A')
clear A
isdef('A')

function level1()
  function level2()
    disp(isdef("a","all"));
    disp(isdef("a","local"));
    disp(isdef("a","nolocal"));
  endfunction
  level2()
endfunction
function go()
  a=1;
  level1()
endfunction
go()
```

See Also

exists isglobal, whereis, type, typeof, clear

Name

isempty — check if a variable is an empty matrix or an empty list

```
t=isempty(x)
```

Parameters

x
: vector or matrix or list

t
: a boolean

Description

`isempty(x)` returns true if x is an empty matrix or an empty list.

Examples

```
a=1
isempty(a(2:$))
isempty(find(rand(1:10)==5))
```

Name

isequal — objects comparison

```
t=isequal(a,b)
t=isequal(a,b,..)
```

Parameters

a, b , ...
: variables of any types

t
: boolean

Description

`isequal` compares its arguments. If all of them are equals function returns `%t` and in the other case it returns `%f` .

When comparing list's, structures,... the comparison is made recursively, the order of the fields matters.

floating point data are compared according to IEEE rule, i.e. NaN values are not equal. See isequalbitwise for bitwise comparisons.

Examples

```
a=[1 2]
isequal(a,[1 2])
isequal(a,1)
```

See Also

isequalbitwise, equal, less

Name

isequalbitwise — bitwise comparison of variables

```
t=isequalbitwise(a,b)
t=isequalbitwise(a,b,..)
```

Parameters

a, b , ...
: variables of any types

t
: boolean

Description

`isequalbitwise` compares its arguments. If all of them are equals function returns `%t` and in the other case it returns `%f`.

When comparing list's, structures,... the comparison is made recursively, the order of the fields matters.

floating point data are compared bitwise, i.e. NaN values are not equal, double(1) and int32(1) are not equal. See isequal for IEEE comparisons.

Examples

```
a=list(1:5,%s+1,'ABCDEFG');
isequalbitwise(a,a)
```

See Also

isequal

Name

isinf — check for infinite entries

```
r=isinf(x)
```

Parameters

x
: real or complex vector or matrix r : boolean vector or matrix

Description

`isinf(x)` returns a boolean vector or matrix which contains true entries corresponding with infinite `x` entries and false entries corresponding with finite `x` entries.

Examples

```
isinf([1 0.01 -%inf %inf])
```

See Also

isnan

Name

isnan — check for "Not a Number" entries

```
r=isnan(x)
```

Parameters

x
: real or complex vector or matrix r : boolean vector or matrix

Description

`isnan(x)` returns a boolean vector or matrix which contains true entries corresponding with "Not a Number" `x` entries and false entries corresponding with regular `x` entries.

Examples

```
isnan([1 0.01 -%nan %inf-%inf])
```

See Also

isinf

Name

isreal — check if a variable as real or complex entries

```
t=isreal(x)
t=isreal(x,eps)
```

Parameters

x
: vector or matrix with floating point entries or coefficients

t
: a boolean

Description

`isreal(x)` returns true if `x` is stored as a real variable and false if `x` is stored with an (eventually zero) imaginary part.

`isreal(x,eps)` returns true if `x` is stored as a real variable or if maximum absolute value of imaginary floating points is less or equal than `eps`.

Examples

```
isreal([1 2])
isreal(1+0*%i)
isreal(1+0*%i,0)
isreal(1+%s)
isreal(sprand(3,3,0.1))
```

Name

kron — Kronecker product (.*.)

```
kron(A,B)
A.*.B
```

Description

`kron(A,B)` or `A.*.B` returns the Kronecker tensor product of two matrices `A` and `B`. The resulting matrix has the following block form:

$$A.*.B = \begin{pmatrix} A(1,1)\cdot B & \cdots & A(1,n)\cdot B \\ \vdots & & \vdots \\ A(m,1)\cdot B & \cdots & A(m,n)\cdot B \end{pmatrix}$$

If `A` is a `m x n` matrix and `B` a `p x q` matrix then `A.*.B` is a `(m*p) x (n*q)` matrix.

`A` and `B` can be sparse matrices.

Examples

```
A=[1,2;3,4];
kron(A,A)
A.*.A
sparse(A).*.sparse(A)
A(1,1)=%i;
kron(A,A)
```

Name

lex_sort — lexicographic matrix rows sorting

```
[N, [k]]=lex_sort(M [,sel] [,'unique'])
```

Parameters

M
: real matrix

N
: real matrix

k
: column vector of integers

Description

the `lex_sort` function is now obsolete. It can be replaced by functions gsort and unique.

`N=lex_sort(M)` sorts the rows (as a group) of the matrix `M` in ascending order. If required the output argument `k` contains the ordering: `[N,k]=lex_sort(M)` returns `k` such as `N` is uequal to `M(k,:)`.

`N=lex_sort(M,sel [,'unique'])` produces the same result as the following sequence of instructions:

```
[N,k]=lex_sort(M(:,sel) [,'unique']);
N=M(k,:)
```

The `'unique'` flag has to be given if one wants to retain only unique rows in the result. Note that `lex_sort(M,sel,'unique')` retains only rows such that M(:,sel) are unique.

Examples

```
M=round(2*rand(20,3));

lex_sort(M)
lex_sort(M,'unique')
[N,k]=lex_sort(M,[1 3],'unique')
```

See Also

gsort, unique

Name

linspace — linearly spaced vector

```
[v]=linspace(x1,x2 [,n])
```

Parameters

x1,x2
: real or complex scalars

n
: integer (number of values) (default value = 100)

v
: real or complex row vector

Description

Linearly spaced vector. `linspace(x1, x2)` generates a row vector of n (default value=100) linearly equally spaced points between `x1` and `x2`. If `x1` or `x2` are complex then `linspace(x1,x2)` returns a row vector of n complexes, the real (resp. imaginary) parts of the n complexes are linearly equally spaced between the real (resp. imaginary) parst of `x1` and `x2`.

Examples

```
linspace(1,2,10)
linspace(1+%i,2+2*%i,10)
```

See Also

logspace

Name

log — natural logarithm

```
y=log(x)
```

Parameters

x
: constant vector or constant matrix

Description

`log(x)` is the "element-wise" logarithm. `y(i,j)=log(x(i,j))`. For matrix logarithm see `logm`.

Examples

```
exp(log([1,%i,-1,-%i]))
```

See Also

exp, logm, log10, ieee

Name

log10 — logarithm

```
y=log10(x)
```

Parameters

x
: vector or matrix

Description

decimal logarithm.If `x` is a vector `log10(x)=[log10(x1),...,log10(xn)]`.

Examples

```
10.^log10([1,%i,-1,-%i])
```

See Also

log, logm, hat, ieee

Name

log1p — computes with accuracy the natural logarithm of its argument added by one

```
y=log1p(x)
```

Parameters

x
: real scalar, vector or matrix

y
: real scalar, vector or matrix

Description

`logp1(x)` is the "element-wise" log(1+x) function. `y(i,j)=log(1 + x(i,j))`. This function, defined for x > -1, must be used if we want to compute log(1+x) with accuracy for |x| << 1.

Examples

```
format("e",24)
log(1.001)
log1p(0.001)
log(1 + 1.e-7)
log1p(1.e-7)
log(1 + 1.e-20)
log1p(1.e-20)
format("v") //reset default format
```

See Also

log

Authors

B.P.

Name

log2 — base 2 logarithm

```
y=log2(x)
```

Parameters

x
: vector or matrix

Description

decimal logarithm.If `x` is a vector `log2(x)=[log2(x1),...,log2(xn)]`.

Examples

```
2.^log2([1,%i,-1,-%i])
```

See Also

log, hat, ieee, log10, frexp

Name

logm — square matrix logarithm

```
y=logm(x)
```

Parameters

x
: square matrix

Description

`logm(x)` is the matrix logarithm of `x`. The result is complex if `x` is not positive or definite positive. If `x` is a symmetric matrix, then calculation is made by schur form. Otherwise, `x` is assumed diagonalizable. One has `expm(logm(x))=x`.

Examples

```
A=[1,2;3,4];
logm(A)
expm(logm(A))

A1=A*A';
logm(A1)
expm(logm(A1))

A1(1,1)=%i;
expm(logm(A1))
```

See Also

expm, log

Name

logspace — logarithmically spaced vector

```
logspace(d1,d2, [n])
```

Parameters

d1,d2
: real or complex scalar (special meaning for `%pi`)

n
: integer (number of values) (default value = 50)

Description

returns a row vector of `n` logarithmically equally spaced points between `10^d1` and `10^d2`. If `d2=%pi` then the points are between `10^d1` and `pi`.

Examples

```
logspace(1,2,10)
```

See Also

linspace

Name

lstsize — list, tlist, mlist numbers of entries

```
n=lstsize(x)
```

Parameters

l
: a list, tlist or mlist object

n
: an integer, the number of entries

Description

`lstsize(x)` returns the number of entries for list, list, mlist objects. This function is more efficient than the `size` function and works similarily with all list types while `size` is overloaded for `mlist` objects.

Examples

```
lstsize(list(1,'aqsdf'))
x=ssrand(3,2,4);
[ny,nu]=size(x)
lstsize(x)
```

See Also

length, size, list, tlist, mlist

Name

max — maximum

```
[m [,k]]=max(A)
[m [,k]]=max(A,'c')
[m [,k]]=max(A,'r')
[m [,k]]=max(A,'m')
[m [,k]]=max(A1,A2,...,An)
[m [,k]]=max(list(A1,A2,...,An))
```

Parameters

A
: real vector or matrix.

A1,...,An
: a set of real vectors or matrices, all of the same size or scalar.

Description

For `A`, a real vector or matrix, `max(A)` is the largest element `A`. `[m,k]=max(A)` gives in addition the index of the maximum. A second argument of type string `'r'` or `'c'` can be used : `'r'` is used to get a row vector `m` such that `m(j)` contains the maximum of the `j` th column of A (`A(:,j)`), `k(j)` gives the row indice which contain the maximum for column `j`. `'c'` is used for the dual operation on the rows of `A`. `'m'` is used for compatibility with Matlab.

`m=max(A1,A2,...,An)`, where all the `Aj` are matrices of the same sizes,returns a vector or a matrix `m` of size `size(m)=size(A1)` such that `m(i)= max( Aj(i)), j=1,...,n`. `[m,k]=max(A1,A2,...,An)` gives in addition the vector or matrix `k`. for a fixed `i`, `k(i)` is the number of the first `Aj(i)` achieving the maximum.

`[m,k]=max(list(A1,...,An))` is an equivalent syntax of `[m,k]=max(A1,A2,...,An)`

Examples

```
[m,n]=max([1,3,1])
[m,n]=max([3,1,1],[1,3,1],[1,1,3])
[m,n]=max([3,-2,1],1)
[m,n]=max(list([3,1,1],[1,3,1],[1,1,3]))
[m,n]=max(list(1,3,1))
```

See Also

sort, find, mini

Name

maxi — maximum

```
[m [,k]]=maxi(A)
[m [,k]]=maxi(A,'c')
[m [,k]]=maxi(A,'r')
[m [,k]]=maxi(A,'m')
[m [,k]]=maxi(A1,A2,...,An)
[m [,k]]=maxi(list(A1,A2,...,An))
```

Parameters

A
: real vector or matrix.

A1,...,An
: a set of real vectors or matrices, all of the same size or scalar.

Description

For `A`, a real vector or matrix, `maxi(A)` is the largest element `A`. `[m,k]=maxi(A)` gives in addition the index of the maximum. A second argument of type string `'r'` or `'c'` can be used : `'r'` is used to get a row vector `m` such that `m(j)` contains the maximum of the `j` th column of A (`A(:,j)`), `k(j)` gives the row indice which contain the maximum for column `j`. `'c'` is used for the dual operation on the rows of `A`. `'m'` is used for compatibility with Matlab.

`m=maxi(A1,A2,...,An)`, where all the `Aj` are matrices of the same sizes,returns a vector or a matrix `m` of size `size(m)=size(A1)` such that `m(i)= maxi( Aj(i)), j=1,...,n`. `[m,k]=maxi(A1,A2,...,An)` gives in addition the vector or matrix `k`. for a fixed `i`, `k(i)` is the number of the first `Aj(i)` achieving the maximum.

`[m,k]=maxi(list(A1,...,An))` is an equivalent syntax of `[m,k]=maxi(A1,A2,...,An)`

Examples

```
[m,n]=maxi([1,3,1])
[m,n]=maxi([3,1,1],[1,3,1],[1,1,3])
[m,n]=maxi([3,-2,1],1)
[m,n]=maxi(list([3,1,1],[1,3,1],[1,1,3]))
[m,n]=maxi(list(1,3,1))
```

See Also

sort, find, mini

Name

meshgrid — create matrices or 3-D arrays

```
[X, Y] = meshgrid(x)
[X, Y] = meshgrid(x,y)
[X, Y, Z] = meshgrid(x,y,z)
```

Parameters

x, y, z
: vectors

X, Y, Z
: matrices in case of 2 input arguments, else 3-D arrays in case of 3 input arguments

Description

Create matrices or 3-D arrays.

Examples

```
x = -1:0.1:1;
y = -1:0.1:1;

[X,Y] = meshgrid(x,y);

for i=1:size(X,1)
  for j=1:size(X,2)
    Z(i,j) = sinc(2*%pi*X(i,j)*Y(i,j));
  end
end

surf(X,Y,Z)
```

See Also

ndgrid

Authors

Farid Belahcene

Name

min — minimum

```
[m [,k]]=min(A)
[m [,k]]=min(A,'c')
[m [,k]]=min(A,'r')
[m [,k]]=min(A,'m')
[m [,k]]=min(A1,A2,...,An)
[m [,k]]=min(list(A1,A2,...,An))
```

Parameters

A
: real vector or matrix.

A1,...,An
: a set of real vectors or matrices, all of the same size or scalar.

Description

For `A`, a real vector or matrix, `min(A)` is the largest element `A`. `[m,k]=min(A)` gives in addition the index of the minimum. A second argument of type string `'r'` or `'c'` can be used : `'r'` is used to get a row vector `m` such that `m(j)` contains the minimum of the `j` th column of A (`A(:,j)`), `k(j)` gives the row indice which contain the minimum for column `j`. `'c'` is used for the dual operation on the rows of `A`. `'m'` is used for compatibility with Matlab.

`m=min(A1,A2,...,An)`, where all the `Aj` are matrices of the same sizes,returns a vector or a matrix `m` of size `size(m)=size(A1)` such that `m(i)= min( Aj(i)), j=1,...,n`. `[m,k]=min(A1,A2,...,An)` gives in addition the vector or matrix `k`. for a fixed `i`, `k(i)` is the number of the first `Aj(i)` achieving the minimum.

`[m,k]=min(list(A1,...,An))` is an equivalent syntax of `[m,k]=min(A1,A2,...,An)`

Examples

```
[m,n]=min([1,3,1])
[m,n]=min([3,1,1],[1,3,1],[1,1,3])
[m,n]=min([3,-2,1],1)
[m,n]=min(list([3,1,1],[1,3,1],[1,1,3]))
[m,n]=min(list(1,3,1))
```

See Also

sort, find, mini

Name

mini — minimum

```
[m [,k]]=mini(A)
[m [,k]]=mini(A,'c')
[m [,k]]=mini(A,'r')
[m [,k]]=mini(A,'m')
[m [,k]]=mini(A1,A2,...,An)
[m [,k]]=mini(list(A1,A2,...,An))
```

Parameters

A
: real vector or matrix.

A1,...,An
: a set of real vectors or matrices, all of the same size or scalar.

Description

For `A`, a real vector or matrix, `mini(A)` is the largest element `A`. `[m,k]=mini(A)` gives in addition the index of the minimum. A second argument of type string `'r'` or `'c'` can be used : `'r'` is used to get a row vector `m` such that `m(j)` contains the minimum of the `j` th column of A (`A(:,j)`), `k(j)` gives the row indice which contain the minimum for column `j`. `'c'` is used for the dual operation on the rows of `A`. `'m'` is used for compatibility with Matlab.

`m=mini(A1,A2,...,An)`, where all the `Aj` are matrices of the same sizes,returns a vector or a matrix `m` of size `size(m)=size(A1)` such that `m(i)= mini( Aj(i)), j=1,...,n`. `[m,k]=mini(A1,A2,...,An)` gives in addition the vector or matrix `k`. for a fixed `i`, `k(i)` is the number of the first `Aj(i)` achieving the minimum.

`[m,k]=mini(list(A1,...,An))` is an equivalent syntax of `[m,k]=mini(A1,A2,...,An)`

Examples

```
[m,n]=mini([1,3,1])
[m,n]=mini([3,1,1],[1,3,1],[1,1,3])
[m,n]=mini([3,-2,1],1)
[m,n]=mini(list([3,1,1],[1,3,1],[1,1,3]))
[m,n]=mini(list(1,3,1))
```

See Also

sort, find, min

Name

minus — (-) substraction operator, sign changes

```
X-Y
-Y
```

Parameters

X
: scalar or vector or matrix of numbers, polynomials or rationals. It may also be a `syslin` list

Y
: scalar or vector or matrix of numbers, polynomials or rationals. It may also be a `syslin` list

Description

Substraction

For numeric operands substraction as its usual meaning. If one of the operands is a matrix and the other one a scalar the the operation is performed element-wise. if `Y==[] X` is returned; if `X==[] -Y` is returned.

Substraction may also be defined for other data types through "soft-coded" operations.

Examples

```
[1,2]-1
[]-2

%s-2
1/%s-2
"cat"+"enate"
```

See Also

addf, mtlb_mode

Name

modulo — symetric arithmetic remainder modulo m
pmodulo — positive arithmetic remainder modulo m

```
i=modulo(n,m)
i=pmodulo(n,m)
```

Parameters

n,m
integers

Description

`modulo` computes `i= n (modulo m)` i.e. remainder of `n` divided by `m` (`n` and `m` integers).

i = n - m .* int (n ./ m). Here the answer may be negative if `n` or `m` are negative.

`pmodulo` computes `i = n - m .* floor (n ./ m)`, the answer is positive or zero.

Examples

```
n=[1,2,10,15];m=[2,2,3,5];
modulo(n,m)

modulo(-3,9)
pmodulo(-3,9)
```

Name

ndgrid — arrays for multidimensional function evaluation on grid

```
[X, Y] = ndgrid(x,y)
[X, Y, Z] = ndgrid(x,y,z)
[X, Y, Z, T] = ndgrid(x,y,z,t)
[X1, X2, ..., Xm] = ndgrid(x1,x2,...,xm)
```

Parameters

x, y, z, ...
: vectors

X, Y, Z, ...
: matrices in case of 2 input arguments, or else hypermatrices

Description

This is an utility routine useful to create arrays for function evaluation on 2, 3, ..., n dimensional grids. For instance in 2d, a grid is defined by two vectors, `x` and `y` of length nx and ny, and you want to evaluate a function (says *f*) on all the grid points, that is on all the points of coordinates *(x(i),y(j))* with *i=1,..,nx* and *j=1,..,ny*. In this case, this function can compute the two matrices `X,Y` of size *nx x ny* such that :

```
X(i,j) = x(i)   for all i in [1,nx]
Y(i,j) = y(j)       and j in [1,ny]
```

and the evaluation may be done with `Z=f(X,Y)` (at the condition that you have coded `f` for evaluation on vector arguments, which is done (in general) by using the element-wise operators `.*`, `./` and `.^` in place of `*`, `/` and `^`).

In the 3d case, considering 3 vectors `x,y,z` of length nx, ny and nz, `X,Y,Z` are 3 hypermatrices of size *nx x ny x nz* such that :

```
X(i,j,k) = x(i)
Y(i,j,k) = y(j)   for all (i,j,k) in [1,nx]x[1,ny]x[1,nz]
Z(i,j,k) = z(k)
```

In the general case of m input arguments `x1, x2, .., xm`, then the m output arguments `X1, X2, .., Xm` are hypermatrices of size *nx1 x nx2 x ... x nxm* and :

```
Xj(i1,i2,...,ij,...,im) = xj(ij)
for all (i1,i2,...,im) in [1,nx1]x[1,nx2]x...x[1,nxm]
```

Examples

```
// create a simple 2d grid
nx = 40; ny = 40;
x = linspace(-1,1,nx);
y = linspace(-1,1,ny);
```

```
[X,Y] = ndgrid(x,y);

// compute a function on the grid and plot it
//deff("z=f(x,y)","z=128*x.^2 .*(1-x).^2 .*y.^2 .*(1-y).^2");
deff("z=f(x,y)","z=x.^2 + y.^3")
Z = f(X,Y);
xbasc()
plot3d(x,y,Z, flag=[2 6 4]); xselect()

// create a simple 3d grid
nx = 10; ny = 6; nz = 4;
x = linspace(0,2,nx);
y = linspace(0,1,ny);
z = linspace(0,0.5,nz);
[X,Y,Z] = ndgrid(x,y,z);

// try to display this 3d grid ...
XF=[]; YF=[]; ZF=[];

for k=1:nz
   [xf,yf,zf] = nf3d(X(:,:,k),Y(:,:,k),Z(:,:,k));
   XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf];
end

for j=1:ny
   [xf,yf,zf] = nf3d(matrix(X(:,j,:),[nx,nz]),...
                     matrix(Y(:,j,:),[nx,nz]),...
                     matrix(Z(:,j,:),[nx,nz]));
   XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf];
end

xbasc()
plot3d(XF,YF,ZF, flag=[0 6 3], leg="X@Y@Z")
xtitle("A 3d grid !"); xselect()
```

See Also

kron

Authors

B. Pincon

Name

ndims — number of dimensions of an array

Parameters

A
: an array

n
: integer, the number of dimensions of the array

Description

`n=ndims(A)` return the number of dimension of the array `A`. `n` is greater than or equal to 2.

Examples

```
A=rand(2,3);
ndims(A)

A=rand(2,3,2);
size(A),ndims(A)

H=[1/%s,1/(%s+1)]
ndims(H)
```

See Also

size

Authors

S. Steer

Name

nearfloat — get previous or next floating-point number

```
xnear = nearfloat(dir, x)
```

Parameters

dir
: string ("succ" or "pred")

x
: real scalar, vector or matrix

xnear
: real scalar, vector or matrix

Description

This function computes, in the element wise meaning, the corresponding neighbours of the elements of `x` (in the underlying floating point set, see number_properties), the successors if `dir = "succ"` and the predecessors if `dir = "pred".`

Examples

```
format("e",22)
nearfloat("succ",1) - 1
1 - nearfloat("pred",1)
format("v") //reset default format
```

See Also

number_properties, frexp

Authors

B.P.

Name

nextpow2 — next higher power of 2.

```
t=nextpow2(x)
```

Parameters

x
: real vector or matrix

p
: integer vector or matrix

Description

If `x` is a scalar, `nextpow2(x)` returns the first `p` such that `2^p >= abs(x)` . if `x` is a vector or a matrix `nextpow2(x)` applies element-wise.

Examples

```
nextpow2(127)
nextpow2(128)
nextpow2(0:10)
```

See Also

frexp

Name

norm — matrix norms

```
[y]=norm(x [,flag])
```

Parameters

x
: real or complex vector or matrix (full or sparse storage)

flag
: string (type of norm) (default value =2)

Description

For matrices

norm(x)
: or `norm(x,2)` is the largest singular value of `x` (`max(svd(x))`).

norm(x,1)
: The l_1 norm `x` (the largest column sum : `maxi(sum(abs(x),'r'))`).

norm(x,'inf'),norm(x,%inf)
: The infinity norm of `x` (the largest row sum : `maxi(sum(abs(x),'c'))`).

norm(x,'fro')
: Frobenius norm i.e. `sqrt(sum(diag(x'*x)))`

For vectors

norm(v,p)
: l_p norm (`sum(v(i)^p))^(1/p)` .

norm(v)
: : `=norm(v,2)` : l_2 norm

norm(v,'inf')
: : `max(abs(v(i)))`.

Examples

```
A=[1,2,3];
norm(A,1)
norm(A,'inf')
A=[1,2;3,4]
max(svd(A))-norm(A)

A=sparse([1 0 0 33 -1])
norm(A)
```

See Also

h_norm, dhnorm, h2norm, abs

Name

not — (~) logical not

```
~A
```

Description

~A gives the element-wise negation of the elements of the boolean matrix A.

Examples

```
~[%t %t %f]
```

See Also

and, or, find

Name

number_properties — determine floating-point parameters

```
pr = number_properties(prop)
```

Parameters

prop
: string

pr
: real or boolean scalar

Description

This function may be used to get the characteristic numbers/properties of the floating point set denoted here by `F(b,p,emin,emax)` (usually the 64 bits float numbers set prescribe by IEEE 754). Numbers of `F` are of the form:

```
sign * m * b^e
```

`e` is the exponent and `m` the mantissa:

```
m = d_1 b^(-1) + d_2 b^(-2) + .... + d_p b^(-p)
```

`d_i` the digits are in `[0, b-1]` and `e` in `[emin, emax]`, the number is said "normalised" if `d_1 ~= 0`. The following may be gotten:

prop = "radix"
: then `pr` is the radix `b` of the set `F`

prop = "digits"
: then `pr` is the number of digits `p`

prop = "huge"
: then `pr` is the max positive float of `F`

prop = "tiny"
: then `pr` is the min positive normalised float of `F`

prop = "denorm"
: then `pr` is a boolean (%t if denormalised numbers are used)

prop = "tiniest"
: then if denorm = %t, `pr` is the min positive denormalised number else `pr` = tiny

prop = "eps"
: then `pr` is the epsilon machine (generally `b^(1-p))/2`) which is the relative max error between a real `x` (such than `|x|` in `[tiny, huge]`) and `fl(x)`, its floating point approximation in `F`

prop = "minexp"
: then `pr` is `emin`

prop = "maxexp"
then `pr` is `emax`

Remarks

This function uses the lapack routine dlamch to get the machine parameters (the names (radix, digit, huge, etc...) are those recommended by the LIA 1 standard and are different from the corresponding lapack's ones) ; CAUTION: sometimes you can see the following definition for the epsilon machine : `eps = b^(1-p)` but in this function we use the traditionnal one (see prop = "eps" before) and so `eps = (b^(1-p))/2` if normal rounding occurs and `eps = b^(1-p)` if not.

Examples

```
b = number_properties("radix")
eps = number_properties("eps")
```

See Also

nearfloat, frexp

Authors

Bruno Pincon

Name

oct2dec — conversion from octal representation to integers

```
d=oct2dec(o)
```

Parameters

d
: matrix of integers

o
: matrix of character strings corresponding to octal representation

Description

`oct2dec(x)` returns the matrix of numbers corresponding to the octal representation.

Examples

```
oct2dec(["1" "756115"; "0" "23"])
```

See Also

base2dec, bin2dec, hex2dec, dec2bin, dec2oct, dec2hex

Name

ones — matrix made of ones

```
y=ones(m1,m2,...)
y=ones(x)
y=ones()
```

Parameters

x,y
: matrices

m1, m2,..
: integers

Description

Returns a matrix made of ones.

ones(m1,m2)
: returns a `(m1,m2)` matrix full of ones.

ones(m1,m2,..,mn)
: creates a `(m1,m2,..,mn)` matrix full of ones.

ones(x)
: returns a matrix full of ones with the same size that `x`.

ones(x)
: is also valid for `x` a `syslin` list.

Note that `ones(3)` is `ones(a)` with `a=3` i.e it is NOT a 3x3 matrix!

ones() is eqivalent to ones(1,1).

Examples

```
ones(3)
ones(3,3)
ones(2,3,2)
```

See Also

eye, zeros

Name

or — (|) logical or

```
or(A), or(A,'*')
or(A,'r'), or(A,1)

or(A,'c'), or(A,2)
A|B
```

Description

`or(A)` gives the `or` of the elements of the boolean matrix `A`. `or(A)` is true (`%t`) iff at least one entry of `A` is `%t`.

`y=or(A,'r')` (or, equivalently, `y=or(A,1)`) is the rowwise or. It returns in each entry of the row vector `y` the or of the rows of `x` (The or is performed on the row index : `y(j)= or(A(i,j),i=1,m)`).

`y=or(A,'c')` (or, equivalently, `y=or(A,2)`) is the columnwise or. It returns in each entry of the column vector `y` the or of the columns of `x` (The or is performed on the column index: `y(i)= or(A(i,j),j=1,n)`)).

`A|B` gives the element-wise logical `or` of the booleans matrices `A` and `B` .`A` and `B` must be matrices with the same dimensions or one from them must be a single boolean.

Examples

```
or([%t %t %f])
[%t %t %f] | [%f %t %t]
[%t %t %f] | %f
```

See Also

and, not, find

Name

pen2ea — pencil to E,A conversion

```
[E,A]=pen2ea(Fs)
```

Parameters

Fs
: matrix pencil `s*E-A`

E,A
: two matrices such that `Fs=s*E-A`

Description

Utility function. Given the pencil `Fs=s*E-A`, returns the matrices `E` and `A`.

Examples

```
E=[1,0];A=[1,2];s=poly(0,'s');
[E,A]=pen2ea(s*E-A)
```

Name

perms — all permutations of vector components

```
y=perms(x)
```

Parameters

x
: scalar or vector

y
: matrix

Description

Given a vector `x` of length `n`, `perms` returns all the permutations of the `n` components of `x` (i.e `n!` permutations). The size of `y` is `n! x n.`

Examples

```
x=[4, 7, 10]
y=perms(x)
x=[1, 5, 2, 5]
y=perms(x)
```

Name

permute — permute the dimensions of an array

```
y=permute(x,dims)
```

Parameters

dims
: a scalar or a vector of positive reals.

x
: a scalar, a vector, a matrix or a mutlti-array.

Description

Permute the dimensions of an array.

Examples

```
//example 1:
x=[1 2 3;4 5 6];
y=permute(x,[2 1]);

//example 2:
x=matrix(1:12,[2,3,2]);
y=permute(x,[3 1 2]);
```

See Also

pertrans, quote, cat

Authors

Farid Belahcene

Name

pertrans — pertranspose

```
[Y]=pertrans(X)
```

Parameters

X
: real or complex matrix

Y
: real or complex matrix

Description

`Y=pertrans(X)` returns the pertranspose of `X`, i.e. the symmetric of `X` w.r.t the second diagonal (utility function).

Examples

```
A=[1,2;3,4]
pertrans(A)
```

Name

primes — primes function

```
[y]=primes(x)
```

Parameters

x
: real scalar

y
: vector

Description

Given a real `x`, `primes(x)` returns in a vector `y` all the primes numbers included between 1 and `x`. If `x<2` then `primes(x)` returns an empty matrix.

Examples

```
x=35
y=primes(x)
```

See Also

factor

Name

prod — product

```
y=prod(x)
y=prod(x,'r') or y=prod(x,1)
y=prod(x,'c') or y=prod(x,2)
y=prod(x,'m')
```

Parameters

x
: real or complex vector or matrix

y
: real or complex scalar or matrix

Description

For a vector or a matrix `x`, `y=prod(x)` returns in the scalar `y` the prod of all the entries of `x`, , e.g. `prod(1:n)` is n!

`y=prod(x,'r')` (or, equivalently, `y=prod(x,1)`)computes the rows elementwise product of `x`. `y` is the row vector: `y(1,j)=prod(x(:,j))`.

`y=prod(x,'c')` (or, equivalently, `y=prod(x,2)`) computes the columns elementwise product of `x`. `y` is the column vector: `y(i,1)=prod(x(i,:))`.

`y=prod(x,'m')` is the product along the first non singleton dimension of `x` (for compatibility with Matlab).

`prod` is not implemented for sparse matrices.

Examples

```
A=[1,2;0,100];
prod(A)
prod(A,'c')
prod(A,'r')
```

See Also

sum, cumprod

Name

rand — random number generator

```
rand(m1,m2,.. [,key])
rand(x [, key])
rand()

rand(key)
rand("seed" [,n])
rand("info")
```

Parameters

mi
: integers

key
: character string with value in `"uniform"`, `"normal"`

x
: a matrix. Only its dimensions are taken into account.

Description

random matrix generator.

Without key argument the syntaxes below produce random matrices with the current random generator (default is "uniform")

rand(m1,m2)
: is a random matrix of dimension `m1` by `m2`.

rand(m1,m2,..,mn)
: is a random matrix of dimension `m1` by `m2`,.. by `mn`.

rand(a)
: is a random matrix of same size as a. `rand(a)` is complex if `a` is a complex matrix.

rand() : with no arguments gives a scalar whose value changes
: each time it is referenced.

If present, the key argument allows to specifie an other random distribution.

rand('uniform')
: The current random generator is set to a uniform random generator. Random numbers are uniformly distributed in the interval (0,1).

rand('normal')
: The current random generator is set to a Gaussian (with mean 0 and variance 1) random number generator.

str=rand('info')
: return the type of the default random generator ('uniform' or 'normal')

IT is possible to (re-)initialize the seed of the rand generator:

rand('seed')
: returns the current value of the seed.

rand('seed',n)
puts the seed to `n`. (by default `n=0` at first call).

Remark

Use the more powerful function grand instead.

Examples

```
x=rand(10,10,'uniform')
rand('normal')
rand('info')
y=rand(x,'normal');
x=rand(2,2,2)
```

See Also

grand, ssrand

Name

rat — Floating point rational approximation

```
[N,D]=rat(x [,tol])
y=rat(x [,tol])
```

Parameters

x
: real vector or matrix

n
: integer vector or matrix

d
: integer vector or matrix

y
: real vector or matrix

Description

`[N,D] = rat(x,tol)` returns two integer matrices so that `N./D` is close to`x` in the sense that `abs(N./D - X) <= tol*abs(x)`. The rational approximations are generated by truncating continued fraction expansions. `tol = 1.e-6*norm(X,1)` is the default. `y = rat(x,tol)` return the quotient `N./D`

Examples

```
[n,d]=rat(%pi)
[n,d]=rat(%pi,1.d-12)
n/d-%pi
```

See Also

int, round

Name

real — real part

```
[y]=real(x)
```

Parameters

x
: real or complex vector or matrix

y
: real matrix

Description

`real(x)` is the real part of `x` (See `%i` to enter complex numbers).

See Also

imag

Name

resize_matrix — create a new matrix with a different size

```
resMat = resize_matrix(mat,nbRow,nbCol,[typeOfMat])
```

Parameters

mat
: input matrix from which the resized matrix will be created.

nbRow
: number of row of the resized matrix.

nbCol
: number of column of the resized matrix.

typeOfMat
: caracter string, type name of the resized matrix.

resMat
: resized matrix.

Description

Create a matrix of size `nbRow` x `nbCol` and whose elements `(i,j)` are `mat(i,j)` if `(i,j)` is in the range of the input matrix. Otherwise elements `(i,j)` are `0` for real or integer matrices, `%f` for boolean matrices and an empty string for string matrices.

The type of the output matrix may be modified by specifying the `typeOfMat` argument. In this case, be sure that the input matrix type is compatible with this one.

For now, only real, integer matrices, boolean and character string matrices are supported. This means that `typeOfMat` must be chosen within: 'constant', 'boolean', 'string' or any integer type ('int8', 'int16',...).

Examples

```
// number matrix
myMat = 5 * rand( 3, 4 )
myMat = resize_matrix( myMat, 3, 3 ) // reduce the matrix size
myMatInteger = resize_matrix( myMat, 4, 4, 'int32' ) // create a integer matrix
myMatBoolean = resize_matrix( myMat, 2, 2, 'boolean' )
myMatBoolean = resize_matrix( myMatBoolean, 3, 5 )

// string matrix
myMatString = ["Scilab","the";"Open Source","Scientific";"Software","Package"]
myMatString = resize_matrix( myMatString, 3, 1 )
```

See Also

matrix, size, typeof

Authors

Jean-Baptiste Silvy

Name

round — rounding

```
y=round(x)
```

Parameters

x
: real or complex matrix

y
: integer or complex (with integer real and imag) matrix

Description

`round(x)` rounds the elements of `x` to the nearest integers.

Examples

```
round([1.9 -2.5])-[2,-3]
round(1.6+2.1*%i)-(2+2*%i)
round(-%inf)
x=rand()*10^20;round(x)-x
```

See Also

int, floor, ceil

Name

sec — Compute the element-wise secant of the argument.

```
y = sec(x)
```

Parameters

x
: Real or complex array.

y
: Real or complex array.

Description

Compute the element-wise secant of the argument. The secant is a periodic finction defined as 1/cos. For real data the results are real and in `]-%inf -1] U [1 %inf[`.

Examples

```
x=[0 %pi/3 2*%pi/3 %pi/4 3*%pi/4 %pi/6 5*%pi/6 %pi];
sec(x)
x=linspace(-%pi,%pi,100)
plot(x,sec(x))
```

See Also

cos, secd

Authors

Serge Steer, INRIA

Used Functions

This function uses the `cos` function.

Name

secd — Compute the element-wise secant of the argument given in degree.

```
y = secd(x)
```

Parameters

x
: Real array.

y
: Real array.

Description

The entries of `y` are thesecant `1/cos` of the entries of `x` given in degree. The results are real and in `]-%inf -1] U [1 %inf[`. For entries equal to `n*180` with `n` integer, the result is exactly `-1` or `+1`. For entries equal to `n*90`with `n` integer and odd the result is infinite(or an error depending on ieee mode).

Examples

```
secd(90)
sec(%pi/2)
```

See Also

cosd, sec

Authors

Serge Steer, INRIA

Name

sech — Compute the element-wise hyperbolic secant of the argument.

```
y = sech(x)
```

Parameters

x
: Real or complex array.

y
: Real or complex array.

Description

Compute the element-wise hyperbolic secant of the argument. The hyperbolic secant is defined as 1/cosh. For real data the results are real and in `[0 1]`.

Examples

```
x=linspace(-10,10,1000)
plot(x,sech(x))
```

See Also

cosh, asech

Authors

Serge Steer, INRIA

Name

setdiff — returns components of a vector which do not belong to another one

```
v=setdiff(a,b)
[v,ka]=setdiff(a,b)
```

Parameters

a
: vector of real numbers or strings

b
: vector of real numbers or strings

v
: vector of real numbers or strings with same orientation than a

ka
: row vector of integers, ka(i) is the location of v(i) in a

Description

`setdiff(a,b)` returns a sorted vector which retains the a entries which are not in b

`[v,ka]=setdiff(a,b)` returns a sorted vector which retains the a entries which are not in b and the location of these entries in a.

Examples

```
a = [223;111;2;4;2;2];
b = [2;3;21;223;123;22];
setdiff(a,b)
[v,k]=setdiff(string(a),string(b))
```

See Also

unique, sort, union

Name

sign — sign function

Description

`X=sign(A)` returns the matrix made of the signs of `A(i,j)`.For complex `A`, `sign(A) = A./abs(A)`. function.

Examples

```
sign(rand(2,3))
sign(1+%i)
```

See Also

abs

Name

signm — matrix sign function

Description

For square and Hermitian matrices `X=signm(A)` is matrix sign function.

Examples

```
A=rand(4,4);B=A+A';X=signm(B);spec(B),spec(X)
```

See Also

sign

Name

sin — sine function

```
[t]=sin(x)
```

Parameters

x
: real or complex vector or matrix

Description

For a vector or a matrix, `sin(x)` is the sine of its elements. For matrix sine use `sinm(X)` function.

Examples

```
asin(sin([1,0,%i]))
```

See Also

sinm

Name

sinc — sinc function

```
t=sinc(x)
```

Parameters

x
: real or complex vector or matrix

t
: real or complex vector or matrix

Description

If `x` is a vector or a matrix, `t=sinc(x)` is the vector or matrix such that `t(i)=sin(x(i))/x(i)` if `x(i)~=0` and `t(i)=1` if `x(i)==0.`

Examples

```
x=linspace(-10,10,3000);
plot2d(x,sinc(x))
```

See Also

sin, cos

Name

sind — sine function, argument in degree.

```
t=sind(x)
```

Parameters

x
: real vector or matrix

t
: real vector or matrix with same dimensions as `x`

Description

For a vector or a matrix x , `sind(x)` is the sine of its elements supposed to be given in degree. The results are in [-1 1]. For integers `n`, `sind(n*180)` is exactly zero.

Examples

```
x=[0,30 45 60 90 360];
sind(x)
```

See Also

sin

Authors

Serge Steer, INRIA

Name

sinh — hyperbolic sine

```
[t]=sinh(x)
```

Parameters

x,t
: real or complex vectors/matrices

Description

the elements of vector `t` are the hyperbolic sine of elements of vector `x`.

Examples

```
asinh(sinh([0,1,%i]))
```

See Also

asinh

Name

sinhm — matrix hyperbolic sine

```
t=sinhm(x)
```

Parameters

x,t
: real or complex square matrix

Description

sinhm(x) is the matrix hyperbolic sine of the matrix x. `t=(expm(x)-expm(-x))/2.`

Examples

```
A=[1,2;2,3]
asinhm(sinhm(A))
A(1,1)=%i;sinhm(A)-(expm(A)-expm(-A))/2   //Complex case
```

See Also

sinh

Name

sinm — matrix sine function

```
t=sinm(x)
```

Parameters

x
: real or complex square matrix

Description

`sinm(x)` is matrix sine of x matrix.

Examples

```
A=[1,2;2,4];
sinm(A)+0.5*%i*(expm(%i*A)-expm(-%i*A))
```

See Also

sin, asinm

Name

size — size of objects

```
y=size(x [,sel])
[nr,nc]=size(x)
```

Parameters

x
: matrix (including transfer matrix) or list or linear system (`syslin`)

y
: `1x2` integer vector or integer number

sel
: a scalar or a character string

nr,nc
: two integers

Description

Applied to :
: a matrix (constant, polynomial, string, boolean, rational) `x`, with only one lhs argument `size` returns a `1x2` vector [number of rows, number of columns]. Called with LHS=2, returns `nr,nc` = [number of rows, number of columns]. `sel` may be used to specify what dimension to get:

1 or 'r'
: to get the number of rows

2 or 'c'
: to get the number of columns

'*'
: to get the product of rows and column numbers

Applied to:
: a list it returns the number of elements. In this case only `y=size(x)` syntax can be used

Applied to:
: a linear system, `y=size(x)` returns in `y` the (row) vector [number of outputs, number if inputs] i.e. the dimension of the corresponding transfer matrix. The syntax `[nr,nc]=size(x)` is also valid (with `(nr,nc)=(y(1),y(2))`). If `x` is a linear system in state-space form, then `[nr,nc,nx]=size(x)` returns in addition the dimension `nx` of the `A` matrix of `x`.

  label='Applied to:'> an hypermatrix `y=size(x)` returns the vector of hypermatrix dimensions. `[n1,n2,...nn]=size(x)` returns the hypermatrix dimensions. `ni=size(x,i)` returns the ith dimension and `size(x,'*')` returns the product of dimensions.

Examples

```
[n,m]=size(rand(3,2))
[n,m]=size(['a','b';'c','d'])
x=ssrand(3,2,4);[ny,nu]=size(x)
[ny,nu]=size(ss2tf(x))
[ny,nu,nx]=size(x)
```

See Also

length, syslin

Name

solve — symbolic linear system solver

```
[x]=solve(A,b)
```

Parameters

A,b,x
: matrix (resp. vectors) of character strings

Description

solves `A*x = b` when `A` is an upper triangular matrix made of character strings.

Examples

```
A=['1','a';'0','2'];   //Upper triangular
b=['x';'y'];

w=solve(A,b)

a=1;x=2;y=5;
evstr(w)
inv([1,1;0,2])*[2;5]
```

See Also

trianfml

Name

sort — stable sorting by "quick sort" algorithm

```
[s, [k]]=sort(v)
[s, [k]]=sort(v,'r')
[s, [k]]=sort(v,'c')
```

Parameters

v
: real or complex vector/matrix; sparse vector; character string vector/matrix

s
: real or complex vector or matrix; sparse vector; character string vector/matrix

k
: vector or matrix of integers

Description

the `sort` implements a "bubble sort algorithm".

`s=sort(v)` sorts `v` in decreasing order. If `v` is a matrix, sorting is done columnwise, `v` being seen as the stacked vector `v(:)`. If `v` is a string, sort is increasing order. `[s,k]=sort(v)` gives in addition the indices of entries of `s` in `v`, i.e. `v(k(:))` is the vector `s`.

`s=sort(v,'r')` sorts the rows of `v` in decreasing order i.e. each column of `s` is obtained from each column of `v` by reordering it in decreasing order. `[s,k]=sort(v,'r')` returns in addition in each column of `k` the indices such that `v(k(:,i),i)=s(:,i)` for each column index `i`.

`s=sort(v,'c')` sorts the columns of `v` in decreasing order i.e. each row of `s` is obtained from each row of `v` by reordering it in decreasing order. `[s,k]=sort(v,'c')` returns in addition in each row of `k` the indices such that `v(i,k(i,:))=s(i,:)` for each row index `i`.

Complex matrix or vectors are sorted by their magnitude. Column/row sorting is not implemented for complex matrices.

`y=sort(A)` is valid when `A` is a sparse vector. Column/row sorting is not implemented for sparse matrix.

Remark : `sort` is now obsolete it may be replaced by **gsort .**

Examples

```
[s,p]=sort(rand(1,10));
//p  is a random permutation of 1:10
A=[1,2,5;3,4,2];
[Asorted,q]=sort(A);A(q(:))-Asorted(:)
v=1:10;
sort(v)
sort(v')
sort(v,'r')  //Does nothing for row vectors
sort(v,'c')
```

See Also

find, gsort

Name

sp2adj — converts sparse matrix into adjacency form

Parameters

A
: real or complex sparse matrix (nz non-zero entries)

xadj
: integer vector of length (n+1).

adjncy
: integer vector of length nz containing the row indices for the corresponding elements in anz

anz
: column vector of length nz, containing the non-zero elements of A

Description

```
sp2adj converts a sparse matrix into its adjacency form (utility fonction).
A =  n x m sparse matrix. xadj, adjncy, anz = adjacency representation of A i.e
```

`xadj(j+1)-xadj(j)` = number of non zero entries in row j. `adjncy` = column index of the non zeros entries in row 1, row 2,..., row n. `anz` = values of non zero entries in row 1, row 2,..., row n. `xadj` is a (column) vector of size n+1 and `adjncy` is an integer (column) vector of size `nz=nnz(A)`. `anz` is a real vector of size `nz=nnz(A)`.

Examples

```
A = sprand(100,50,.05);
[xadj,adjncy,anz]= sp2adj(A);
[n,m]=size(A);
p = adj2sp(xadj,adjncy,anz,[n,m]);
A-p
```

See Also

adj2sp, sparse, spcompack, spget

Name

speye — sparse identity matrix

```
Isp=speye(nrows,ncols)
Isp=speye(A)
```

Parameters

nrows
: integer (number of rows)

ncols
: integer (number os columns)

A
: sparse matrix

sp
: sparse identity matrix

Description

`Isp=speye(nrows,ncols)` returns a sparse identity matrix `Isp` with `nrows` rows, `ncols` columns. (Non square identity matrix have a maximal number of ones along the main diagonal).

`Isp=speye(A)` returns a sparse identity matrix with same dimensions as `A`. If `[m,n]=size(A)`, `speye(m,n)` and `speye(A)` are equivalent. In particular `speye(3)` is not equivalent to `speye(3,3)`.

Examples

```
eye(3,3)-full(speye(3,3))
```

See Also

sparse, full, eye, spzeros, spones

Name

splin2d — bicubic spline gridded 2d interpolation

```
C = splin2d(x, y, z, [,spline_type])
```

Parameters

x,y
: strictly increasing row vectors (with at least 2 components) defining the interpolation grid

z
: nx x ny matrix (nx being the length of `x` and ny the length of `y`)

spline_type
: (optional) a string selecting the kind of bicubic spline to compute

C
: a big vector with the coefficients of the bicubic patches (see details in Remarks)

Description

This function computes a bicubic spline or sub-spline *s* which interpolates the *(xi,yj,zij)* points, ie, we have *s(xi,yj)=zij* for all *i=1,..,nx* and *j=1,..,ny*. The resulting spline *s* is defined by the triplet `(x,y,C)` where `C` is the vector (of length 16(nx-1)(ny-1)) with the coefficients of each of the (nx-1)(ny-1) bicubic patches : on *[x(i) x(i+1)]x[y(j) y(j+1)]*, *s* is defined by :

$$s(x,y) = \sum_{k=1}^{4}\sum_{l=1}^{4} c_{ij}(k,l)\cdot(x-x_i)^{k-1}\cdot(y-y_i)^{l-1}$$

The evaluation of *s* at some points must be done by the interp2d function. Several kind of splines may be computed by selecting the appropriate `spline_type` parameter. The method used to compute the bicubic spline (or sub-spline) is the old fashionned one 's, i.e. to compute on each grid point *(xi,yj)* an approximation of the first derivatives *ds/dx(xi,yj)* and *ds/dy(xi,yj)* and of the cross derivative *d2s/dxdy(xi,yj)*. Those derivatives are computed by the mean of 1d spline schemes leading to a C2 function (*s* is twice continuously differentiable) or by the mean of a local approximation scheme leading to a C1 function only. This scheme is selected with the `spline_type` parameter (see splin for details) :

"not_a_knot"
: this is the default case.

"periodic"
: to use if the underlying function is periodic : you must have *z(1,j) = z(nx,j) for all j in [1,ny] and z(i,1) = z(i,ny) for i in [1,nx]* but this is not verified by the interface.

Remarks

From an accuracy point of view use essentially the **not_a_knot** type or **periodic** type if the underlying interpolated function is periodic.

The **natural**, **monotone**, **fast** (or **fast_periodic**) type may be useful in some cases, for instance to limit oscillations (**monotone** being the most powerfull for that).

To get the coefficients of the bi-cubic patches in a more friendly way you can use `c = hypermat([4,4,nx-1,ny-1],C)` then the coefficient *(k,l)* of the patch *(i,j)* (see equation here

before) is stored at `c(k,l,i,j)`. Nevertheless the interp2d function wait for the big vector `C` and not for the hypermatrix `c` (note that one can easily retrieve `C` from `c` with `C=c(:)`).

Examples

```
// example 1 : interpolation of cos(x)cos(y)
n = 7;  // a regular grid with n x n interpolation points
        // will be used
x = linspace(0,2*%pi,n); y = x;
z = cos(x')*cos(y);
C = splin2d(x, y, z, "periodic");
m = 50; // discretisation parameter of the evaluation grid
xx = linspace(0,2*%pi,m); yy = xx;
[XX,YY] = ndgrid(xx,yy);
zz = interp2d(XX,YY, x, y, C);
emax = max(abs(zz - cos(xx')*cos(yy)));
xbasc()
plot3d(xx, yy, zz, flag=[2 4 4])
[X,Y] = ndgrid(x,y);
param3d1(X,Y,list(z,-9*ones(1,n)), flag=[0 0])
str = msprintf(" with %d x %d interpolation points. ermax = %g",n,n,emax)
xtitle("spline interpolation of cos(x)cos(y)"+str)

// example 2 : different interpolation functions on random datas
n = 6;
x = linspace(0,1,n); y = x;
z = rand(n,n);
np = 50;
xp = linspace(0,1,np); yp = xp;
[XP, YP] = ndgrid(xp,yp);
ZP1 = interp2d(XP, YP, x, y, splin2d(x, y, z, "not_a_knot"));
ZP2 = linear_interpn(XP, YP, x, y, z);
ZP3 = interp2d(XP, YP, x, y, splin2d(x, y, z, "natural"));
ZP4 = interp2d(XP, YP, x, y, splin2d(x, y, z, "monotone"));
xset("colormap", jetcolormap(64))
xbasc()
subplot(2,2,1)
   plot3d1(xp, yp, ZP1, flag=[2 2 4])
   xtitle("not_a_knot")
subplot(2,2,2)
   plot3d1(xp, yp, ZP2, flag=[2 2 4])
   xtitle("bilinear interpolation")
subplot(2,2,3)
   plot3d1(xp, yp, ZP3, flag=[2 2 4])
   xtitle("natural")
subplot(2,2,4)
   plot3d1(xp, yp, ZP4, flag=[2 2 4])
   xtitle("monotone")
xselect()

// example 3 : not_a_knot spline and monotone sub-spline
//             on a step function
a = 0; b = 1; c = 0.25; d = 0.75;
// create interpolation grid
n = 11;
x = linspace(a,b,n);
ind = find(c <= x & x <= d);
```

```
z = zeros(n,n); z(ind,ind) = 1;  // a step inside a square
// create evaluation grid
np = 220;
xp = linspace(a,b, np);
[XP, YP] = ndgrid(xp, xp);
zp1 = interp2d(XP, YP, x, x, splin2d(x,x,z));
zp2 = interp2d(XP, YP, x, x, splin2d(x,x,z,"monotone"));
// plot
xbasc()
xset("colormap",jetcolormap(128))
subplot(1,2,1)
   plot3d1(xp, xp, zp1, flag=[-2 6 4])
   xtitle("spline (not_a_knot)")
subplot(1,2,2)
   plot3d1(xp, xp, zp2, flag=[-2 6 4])
   xtitle("subspline (monotone)")
```

See Also

cshep2d, linear_interpn, interp2d

Authors

B. Pincon

Name

spones — sparse matrix

```
sp=spones(A)
```

Parameters

A
: sparse matrix

sp
: sparse matrix

Description

`sp=spones(A)` generates a matrix with the same sparsity structure as A, but with ones in the nonzero positions.

Examples

```
A=sprand(10,12,0.1);
sp=spones(A)
B = A~=0
bool2s(B)
```

See Also

sparse, full, eye, speye, spzeros

Name

sprand — sparse random matrix

```
sp=sprand(nrows,ncols,fill [,typ])
```

Parameters

nrows
: integer (number of rows)

ncols
: integer (number of columns)

fill
: filling coefficient (density)

typ
: character string (`'uniform'` (default) or `'normal'`)

sp
: sparse matrix

Description

`sp=sprand(nrows,ncols,fill)` returns a sparse matrix `sp` with `nrows` rows, `ncols` columns and approximately `fill*nrows*ncols` non-zero entries.

If `typ='uniform'` uniformly distributed values on [0,1] are generated. If `typ='normal'` normally distributed values are generated (mean=0 and standard deviation=1).

Examples

```
W=sprand(100,1000,0.001);
```

See Also

sparse, full, rand, speye

Name

spzeros — sparse zero matrix

```
sp=spzeros(nrows,ncols)
sp=spzeros(A)
```

Parameters

nrows
: integer (number of rows)

ncols
: integer (number os columns)

A
: sparse matrix

sp
: sparse zero matrix

Description

`sp=spzeros(nrows,ncols)` returns a sparse zero matrix `sp` with `nrows` rows, `ncols` columns. (Equivalent to `sparse([],[],[nrow,ncols])`)

`sp=spzeros(A)` returns a sparse zero matrix with same dimensions as `A`. If `[m,n]=size(A)`, `spzeros(m,n)` and `spzeros(A)` are equivalent. In particular `spzeros(3)` is not equivalent to `spzeros(3,3)`.

Examples

```
sum(spzeros(1000,1000))
```

See Also

sparse, full, eye, speye, spones

Name

sqrt — square root

```
y=sqrt(x)
```

Parameters

x
: real or complex scalar or vector

Description

`sqrt(x)` is the vector of the square root of the `x` elements. Result is complex if `x` is negative.

Examples

```
sqrt([2,4])
sqrt(-1)
```

See Also

hat, sqrtm

Name

sqrtm — matrix square root

```
y=sqrtm(x)
```

Parameters

x
: real or complex square matrix

Description

`y=sqrt(x)` is the matrix square root of the `x` x matrix (`x=y^2`) Result may not be accurate if `x` is not symmetric.

Examples

```
x=[0 1;2 4]
w=sqrtm(x);
norm(w*w-x)
x(1,2)=%i;
w=sqrtm(x);norm(w*w-x,1)
```

See Also

expm, sqroot

Name

squarewave — generates a square wave with period 2*%pi

```
x=squarewave(t [,percent])
```

Parameters

t
: real vector, time discretization

x
: real vector, the wave value at each time point in set (-1,+1)

percent
: real positive scalar, the percent of the period in which the signal is positive. Defaut value is 50

Description

`squarewave(t)` generates the vector of the values of the square wave with period 2*%pi at each date given in the `t` vector.

`squarewave(t,%)` generates a square wave such that `%` is the percent of the period in which the signal is positive.

Examples

```
t=(0:0.1:5*%pi)';
plot2d1('onn',t,[2*sin(t),1.5*squarewave(t),squarewave(t,10)])
```

See Also

sin, cos

Name

ssprint — pretty print for linear system

```
ssprint(sl [,out])
```

Parameters

sl
: list (`syslin` list)

out
: output (default value `out=%io(2)`)

Description

pretty print of a linear system in state-space form `sl=(A,B,C,D) syslin` list.

Examples

```
a=[1 1;0 1];b=[0 1;1 0];c=[1,1];d=[3,2];
ssprint(syslin('c',a,b,c,d))
ssprint(syslin('d',a,b,c,d))
```

See Also

texprint

Name

ssrand — random system generator

```
sl=ssrand(nout,nin,nstate)
[sl,U]=ssrand(nout,nin,nstate,flag)
```

Parameters

nout
: integer (number of output)

nin
: integer (number of input)

nstate
: integer (dimension of state-space)

flag
: list made of one character string and one or several integers

sl
: list (`syslin` list)

U
: square `(nstate x nstate)` nonsingular matrix

Description

`sl=ssrand(nout,nin,nstate)` returns a random strictly proper (`D=0`) state-space system of size `[nout,nint]` represented by a `syslin` list and with `nstate` state variables.

`[sl,U]=ssrand(nout,nin,nstate,flag)` returns a test linear system with given properties specified by `flag`. `flag` can be one of the following:

```
flag=list('co',dim_cont_subs)
flag=list('uo',dim_unobs_subs)
flag=list('ncno',dim_cno,dim_ncno,dim_co,dim_nco)
flag=list('st',dim_cont_subs,dim_stab_subs,dim_stab0)
flag=list('dt',dim_inst_unob,dim_instb0,dim_unobs)
flag=list('on',nr,ng,ng0,nv,rk)
flag=list('ui',nw,nwu,nwui,nwuis,rk)
```

The complete description of the `Sys` is given in the code of the `ssrand` function (in `SCIDIR/macros/util`). For example with `flag=list('co',dim_cont_subs)` a non-controllable system is return and `dim_cont_subs` is the dimension of the controllable subspace of `Sys`. The character strings `'co','uo','ncno','st','dt','on','ui'` stand for "controllable", "unobservable", "non-controllable-non-observable", "stabilizable","detectable","output-nulling","unknown-input".

Examples

```
//flag=list('st',dim_cont_subs,dim_stab_subs,dim_stab0)
//dim_cont_subs<=dim_stab_subs<=dim_stab0
//pair (A,B) U-similar to:
```

```
//    [*,*,*,*;     [*;
//    [0,s,*,*;     [0;
//A=  [0,0,i,*;   B=[0;
//    [0,0,0,u]     [0]
//
// (A11,B1) controllable  s=stable matrix i=neutral matrix u=unstable matrix
[Sl,U]=ssrand(2,3,8,list('st',2,5,5));
w=ss2ss(Sl,inv(U)); //undo the random change of basis => form as above
[n,nc,u,sl]=st_ility(Sl);n,nc
```

See Also

syslin

Name

sub2ind — matrix subscript values to linear index

```
I = sub2ind(dims,i1,i2,...)
J = sub2ind(dims,Mi)
```

Parameters

dims
: vector: the matrix dimensions

i1,i2,...
: the subscript value arrays(same matrix shape as `I`)

Mi
: matrix whose columns contains the subscript values.

I
: the linear index array

Description

`sub2ind` is used to determine the equivalent single index corresponding to a given set of subscript values. `I = sub2ind(dims,i1,i2,..)` returns the linear index equivalent to the row, column, ... subscripts in the arrays `i1`, `i2`,.. for an matrix of size `dims`. In this case `i1`, `i2`,.. must have the same shape and the result `I` has the same matrix shape. `I = sub2ind(dims,Mi)` returns the linear index equivalent to the subscripts in the columns of the matrix `Mi` for a matrix of size `dims`. in this case `I` is a column vector.

Examples

```
i=[1 2 1 1 2 1 1];
j=[1 2 3 1 2 3 3];
k=[1 2 1 2 1 2 1];
sub2ind([2,3,2],i,j,k)

sub2ind([2,3,2],[i',j',k'])
```

See Also

ind2sub, extraction, insertion

Authors

Serge Steer, INRIA

Name

sum — sum (row sum, column sum) of vector/matrix entries

```
y=sum(x)
y=sum(x,'r') or y=sum(x,1)

y=sum(x,'c') or y=sum(x,2)

y=sum(x,'m')
```

Parameters

x
: vector or matrix (real, complex, sparse or polynomial)

y
: scalar or vector

Description

For a vector or a matrix `x`, `y=sum(x)` returns in the scalar `y` the sum of all the entries of `x`.

`y=sum(x,'r')` (or, equivalently, `y=sum(x,1)`) is the rowwise sum: `y(j)= sum(x(:,j))`. `y` is a row vector

`y=sum(x,'c')` (or, equivalently, `y=sum(x,2)`) is the columnwise sum. It returns in each entry of the column vector `y` the sum : `y(i)= sum(x(i,:))`).

`y=sum(x,'m')` is the sum along the first non singleton dimension of `x` (for compatibility with Matlab).

Examples

```
A=[1,2;3,4];
trace(A)-sum(diag(A))
sum(A,'c')-A*ones(2,1)
sum(A+%i)
A=sparse(A);sum(A,'c')-A*ones(2,1)
s=poly(0,'s');
M=[s,%i+s;s^2,1];
sum(M),sum(M,2)
```

See Also

cumsum, prod

Name

sysconv — system conversion

```
[s1,s2]=sysconv(s1,s2)
```

Parameters

s1,s2
: list (linear `syslin` systems)

Description

Converts `s1` and `s2` into common representation in order that system interconnexion operations can be applied. Utility function for experts. The conversion rules in given in the following table.

"c"
: continuous time system

"d"
: discrete time system

n
: sampled system with sampling period n

[]
: system with undefined time domain For mixed systems `s1` and `s2` are put in state-space representation.

```
s1\s2 |    "c"     |     "d"     |      n2        |     []     |
-------------------------------------------------------------------
"c"   | nothing    |uncompatible | c2e(s1,n2)     |  c(s2)     |
-------------------------------------------------------------------
"d"   |uncompatible| nothing     | e(s1,n2)       |  d(s2)     |
-------------------------------------------------------------------
n1    | c2e(s2,n1) | e(s2,n1)    | n1<>n2 uncomp  |  e(s2,n1)  |
      |            |             | n1=n2  nothing |            |
-------------------------------------------------------------------
[]    | c(s1)      | d(s1)       | e(s1,n2)       |  nothing   |
-------------------------------------------------------------------
```

With the following meaning:

n1,n2
: sampling period

c2e(s,n)
: the continuous-time system s is transformed into a sampled system with sampling period n.

c(s)
: conversion to continuous (time domain is `"c"`)

d(s)
: conversion to discrete (time domain is `"d"`)

e(s,n)
conversion to samples system with period n

Examples

```
s1=ssrand(1,1,2);
s2=ss2tf(s1);
[s1,s2]=sysconv(s1,s2);
```

See Also

syslin, ss2tf, tf2ss

Name

sysdiag — block diagonal system connection

```
r=sysdiag(a1,a2,...,an)
```

Description

Returns the block-diagonal system made with subsystems put in the main diagonal

ai
: subsystems (i.e. gains, or linear systems in state-space or transfer form)

Used in particular for system interconnections.

Remark

At most 17 arguments.

Examples

```
s=poly(0,'s')
sysdiag(rand(2,2),1/(s+1),[1/(s-1);1/((s-2)*(s-3))])
sysdiag(tf2ss(1/s),1/(s+1),[1/(s-1);1/((s-2)*(s-3))])

s=poly(0,'s')
sysdiag(rand(2,2),1/(s+1),[1/(s-1);1/((s-2)*(s-3))])
sysdiag(tf2ss(1/s),1/(s+1),[1/(s-1);1/((s-2)*(s-3))])
```

See Also

brackets, insertion, feedback

Name

syslin — linear system definition

```
[sl]=syslin(dom,A,B,C [,D [,x0] ])
[sl]=syslin(dom,N,D)
[sl]=syslin(dom,H)
```

Parameters

dom
: character string (`'c'`, `'d'`), or `[]` or a scalar.

A,B,C,D
: matrices of the state-space representation (`D` optional with default value zero matrix). For improper systems `D` is a polynomial matrix.

x0
: vector (initial state; default value is `0`)

N, D
: polynomial matrices

H
: rational matrix or linear state space representation

sl
: tlist ("`syslin`" list) representing the linear system

Description

`syslin` defines a linear system as a list and checks consistency of data.

`dom` specifies the time domain of the system and can have the following values:

`dom='c'` for a continuous time system, `dom='d'` for a discrete time system, `n` for a sampled system with sampling period `n` (in seconds).

`dom=[]` if the time domain is undefined

State-space representation:

```
sl=syslin(dom,A,B,C [,D [,x0] ])
```

represents the system :

$$s \cdot x = A \cdot x + B \cdot u$$
$$y = C \cdot x + D \cdot u$$
$$x(0) = x0$$

The output of `syslin` is a list of the following form: `sl=tlist(['lss','A','B','C','D','X0','dt'],A,B,C,D,x0,dom)` Note that `D` is allowed to be a polynomial matrix (improper systems).

Transfer matrix representation:

```
sl=syslin(dom,N,D)
sl=syslin(dom,H)
```

The output of `syslin` is a list of the following form : `sl=tlist(['r','num','den','dt'],N,D,dom)` or `sl=tlist(['r','num','den','dt'],H(2),H(3),dom)`.

Linear systems defined as `syslin` can be manipulated as usual matrices (concatenation, extraction, transpose, multiplication, etc) both in state-space or transfer representation.

Most of state-space control functions receive a `syslin` list as input instead of the four matrices defining the system.

Examples

```
A=[0,1;0,0];B=[1;1];C=[1,1];
S1=syslin('c',A,B,C)   //Linear system definition
S1("A")    //Display of A-matrix
S1("X0"), S1("dt") // Display of X0 and time domain
s=poly(0,'s');
D=s;
S2=syslin('c',A,B,C,D)
H1=(1+2*s)/s^2, S1bis=syslin('c',H1)
H2=(1+2*s+s^3)/s^2, S2bis=syslin('c',H2)
S1+S2
[S1,S2]
ss2tf(S1)-S1bis
S1bis+S2bis
S1*S2bis
size(S1)
```

See Also

tlist, lsslist, rlist, ssrand, ss2tf, tf2ss, dscr, abcd

Name

tan — tangent

```
[t]=tan(x)
```

Parameters

x
: vector or matrix

t
: vector or matrix

Description

The elements of `t` are the tangent of the elements of `x`.

Examples

```
x=[1,%i,-1,-%i]
tan(x)
sin(x)./cos(x)
```

See Also

atan, tanm

Name

tand — tangent, argument in degree.

```
t=tand(x)
```

Parameters

x
: real vector or matrix

t
: real vector or matrix

Description

The elements of `t` are the tangent of the elements of `x`.

Examples

```
mod=ieee();ieee(2);
x=[0,30 45 60 90 360];
tand(x)
ieee(mod)
```

See Also

atand, tan

Name

tanh — hyperbolic tangent

```
t=tanh(x)
```

Description

the elements of `t` are the hyperbolic tangents of the elements of `x.`

Examples

```
x=[1,%i,-1,-%i]
tanh(x)
sinh(x)./cosh(x)
```

See Also

atanh, tan, tanhm

Name

tanhm — matrix hyperbolic tangent

```
t=tanhm(x)
```

Parameters

x,t
: real or complex square matrix

Description

tanhm is the matrix hyperbolic tangent of the matrix x.

Examples

```
A=[1,2;3,4];
tanhm(A)
```

See Also

tan, tanm, expm, sinm, cosm, atanhm

Name

tanm — matrix tangent

```
[t]=tanm(x)
```

Parameters

x
: square real or complex matrix

t
: square matrix

Description

`tanm(x)` is the matrix tangent of the square matrix x.

Examples

```
A=[1,2;3,4];
tanm(A)
```

See Also

tan, expm, sinm, atanm

Name

toeplitz — toeplitz matrix

```
A=toeplitz(c [,r])
```

Parameters

a,c,r
: constant, polynomial or string matrices

Description

returns the Toeplitz matrix whose first row is `r` and first column is `c`. `c(1)` must be equal to `r(1)`. `toeplitz(c)` returns the symmetric Toeplitz matrix.

Examples

```
A=toeplitz(1:5);

T=toeplitz(1:5,1:2:7);T1=[1 3 5 7;2 1 3 5;3 2 1 3;4 3 2 1;5 4 3 2];
T-T1

s=poly(0,'s');
t=toeplitz([s,s+1,s^2,1-s]);
t1=[s,1+s,s*s,1-s;1+s,s,1+s,s*s;s*s,1+s,s,1+s;1-s,s*s,1+s,s]
t-t1

t=toeplitz(['1','2','3','4']);
t1=['1','2','3','4';'2','1','2','3';'3','2','1','2';'4','3','2','1']
```

See Also

matrix

Name

trfmod — poles and zeros display

```
[hm]=trfmod(h [,job])
```

Description

To visualize the pole-zero structure of a SISO transfer function `h` .

job='p'
: visualization of polynomials (default)

job='f'
: visualization of natural frequencies and damping

Interactive simplification of `h`. `trfmod` opens a dialog window.

See Also

poly , simp

Name

trianfml — symbolic triangularization

```
[f [,sexp]]=trianfml(f [,sexp])
```

Description

Symbolic triangularization of the matrix `f` ; triangularization is performed by elementary row operations; `sexp` is a set of common expressions stored by the algorithm.

Examples

```
A=['1','2';'a','b']
W=trianfml([A,string(eye(2,2))])
U=W(:,3:4)
a=5;b=6;
A=evstr(A)
U=evstr(U)
U*A
evstr(W(:,1:2))
```

See Also

addf, mulf, solve, trisolve

Name

tril — lower triangular part of matrix

```
tril(x [,k])
```

Parameters

x
: matrix (real, complex, polynomial, rational)

k
: integer (default value 0)

Description

Lower triangle part of a matrix. `tril(x,k)` is made by entries below the kth diagonal : k>0 (upper diagonal) and k<0 (diagonals below the main diagonal).

Examples

```
s=poly(0,'s');
tril([s,s;s,1])
tril([1/s,1/s;1/s,1])
```

See Also

triu, ones, eye, diag

Name

trisolve — symbolic linear system solver

```
[x [,sexp]] = trisolve(A,b [,sexp])
```

Parameters

A,b
: matrices of strings

Description

symbolically solves `A*x =b` , `A` being assumed to be upper triangular.

`sexp` is a vector of common subexpressions in `A, b, x`.

Examples

```
A=['x','y';'0','z'];b=['0';'1'];
w=trisolve(A,b)
x=5;y=2;z=4;
evstr(w)
inv(evstr(A))*evstr(b)
```

See Also

trianfml, solve

Authors

F.D, S.S

Name

triu — upper triangle

Description

Upper triangle. See tril.

Examples

```
s=poly(0,'s');
triu([s,s;s,1])
triu([1/s,1/s;1/s,1])
```

See Also

tril, ones, eye, diag

Name

typeof — object type

```
[t]=typeof(object)
```

Parameters

object
: Scilab object

t
: string

Description

`t=typeof(object)` returns one of the following strings:

"constant"
: if object is a real or complex constant matrix.

"polynomial"
: if object is a polynomial matrix.

"function"
: if object is a function (Scilab code).

"handle"
: if object is an handle.

"string"
: if object is a matrix made of character strings.

"boolean"
: if object is a boolean matrix.

"list"
: if object is a list.

"rational"
: if object is a rational matrix (transfer matrix).

"state-space"
: if object is a state-space model (see syslin).

"sparse"
: if object is a (real) sparse matrix.

"boolean sparse"
: if object is a boolean sparse matrix.

"hypermat"
: if object is an hypermatrix (N-dimension array with N >=3).

"st"
: if object is a structure array.

"ce"
: if object is a cell array.

the first string in the first list entry
if object is a tlist or mlist.

"fptr"
if object is a Scilab intrinsic (C or Fortran code).

"pointer"
if object is a pointer (See lufact).

"size implicit"
if object is a size implicit polynom used for indexing.

Examples

```
typeof(1)
typeof(poly(0,'x'))

typeof(1/poly(0,'x'))
typeof(%t)

w=sprand(100,100,0.001);
typeof(w)
typeof(w==w)

deff('y=f(x)','y=2*x');
typeof(f)

L=tlist(['V','a','b'],18,'Scilab');
typeof(L)
```

See Also

type, strings, syslin, poly

Name

union — extract union components of a vector

```
[v [,ka, kb] ] = union(a,b)
[v [,ka, kb] ] = union(a,b,orient)
```

Parameters

a
: vector or matrix of numbers or strings

b
: vector of real numbers or strings

orient
: flag with possible values : 1 or "r", 2 or "c".

v
: row vector or matrix of numbers or strings

ka
: row vector of integers

kb
: row vector of integers

Description

`union(a,b)` returns a sorted row vector which retains the unique entries of `[a(:);b(:)]`.

`union(a,b,"r")` or `union(a,b,1)`returns the matrix formed by the union of the unique rows of `a` and `b` sorted in lexicographic ascending order. In this case matrices `a` and `b` must have the same number of columns.

`union(a,b,"c")` or `union(a,b,2)`returns the matrix formed by the union of the unique columns of `a` and `b` sorted in lexicographic ascending order. In this case matrices `a` and `b` must have the same number of rows.

`[v,ka,kb]=union(a,b)` also returns index vectors `ka` and `kb` such that `v` is a sorted combination of the entries `a(ka)` and `b(kb)`.

Examples

```
A=round(5*rand(10,1));
B=round(5*rand(7,1));

union(A,B)
[N,ka,kb]=union(A,B)

union('a'+string(A),'b'+string(B))
```

See Also

unique, gsort

Name

unique — extract unique components of a vector or matrices

```
[N [,k]]=unique(M)
[N [,k]]=unique(M ,orient)
```

Parameters

M
: vector or matrix of numbers or strings

orient
: flag with possible values : 1 or "r", 2 or "c"

N
: vector or matrix of numbers or strings

k
: vector of integers

Description

`unique(M)` returns a vector which retains the unique entries of `M` in ascending order.

`unique(M,"r")` or `unique(M,1)`returns the unique rows of `M` in lexicographic ascending order.

`unique(M,"c")` or `unique(M,2)`returns the unique columns of `M` in lexicographic ascending order.

If required the output argument `k` contains the position of the first encountered unique entries.

Examples

```
M=round(2*rand(20,1));

unique(M)
[N,k]=unique(M)

unique(string(M))
[N,k]=unique(string(M))


A = [0,0,1,1;
     0,1,1,1;
     2,0,1,1;
     0,2,2,2;
     2,0,1,1;
     0,0,1,1];
T='x'+string(A);

//unique rows

[m,k]=unique(A,'r')
unique(T,'r')
```

```
//unique columns
[m,k]=unique(T,'c')
unique(A,'c')
```

See Also

union, intersect, gsort, lex_sort

Name

vectorfind — finds in a matrix rows or columns matching a vector

```
ind = vectorfind(m,v,job)
```

Parameters

m
: a matrix of any type

v
: vector of any type

job
: string flag with possible values "r" to look for matching rows or "c" to look for matching columns

ind
: row vector containing indices of matching rows or columns

Description

finds in a matrix rows or columns matching a vector.

Examples

```
alr=[1,2,2;
     1,2,1;
     1,1,2;
     1,1,1;
     1,2,1];

ind = vectorfind(alr,[1,2,1],'r')
ind = vectorfind(string(alr),string([1,2,1]),'r')
```

See Also

find, gsort

Authors

R. Nikoukhah, S. Steer INRIA

Name

zeros — matrix made of zeros

```
y=zeros()
y=zeros(x)
y=zeros(m1,m2,..)
```

Parameters

x,y
: matrices

m1, m2,..
: integers

Description

Creates matrix of zeros (same as `0*ones`).

zeros(m1,m2)
: for an `(m1,m2)` matrix.

zeros(m1,m2,..,mn)
: creates a `(m1,m2,..,mn)` matrix filled with zeros

zeros(A)
: for a matrix of same size of `A`.

zeros(3)
: is `zeros(a)` with `a=3` i.e it is NOT a 3x3 matrix!

zeros()
: returns a single zero

If `x` is a `syslin` list (linear system in state-space or transfer form), `zeros(x)` is also valid and returns a zero matrix.

Examples

```
zeros(3)
zeros(3,3)
zeros(2,3,2)
```

See Also

eye, ones, spzeros

FFTW

Name

fftw — fast fourier transform that use fftw library

```
[y]=fftw(x)
[y]=fftw(x,sign)
[y]=fftw(x,sign,dim,incr)
[y]=fftw(x,sign,[dim1 dim2 ...dimN],[incr1 incr2 ...incrN])
```

Parameters

y,x
: matrix/vector of real/complex data. Input/output data to be transformed.

sign
: Integer. 1 or -1. Set direct or inverse transform.

dim
: integer. Set the dimension (the length) of the transform.

incr
: integer. Set the stride (the span) of the transform.

Description

This function realizes direct/inverse Discrete Fourier Transform (DFT) with the help of the FFTW library.

One can compute vector, 2D, M-D transform with this function.

For more details of fftw syntax see fft scilab function.

For more details about FFTW library see FFTW Web site : http://www.fftw.org

Remark : fftw function automaticaly stores his last parameters in memory to re-use it in a second time.

This results on a time computation improvement when consecutives calls (with same parameters) are used.

Examples

```
//simple vector direct transform
a = rand(50,1)+%i*rand(50,1);
y = fftw(a);
y = fftw(a,-1);
//inverse transform
b = fftw(y,1);

//2D transform
a = rand(512,512)+%i*rand(512,512);
y = fftw(a);

//M-D transform -old calling sequence-
a = rand(120,1);
y = a;
dim=[5 6 4];incr=[1 5 30];
```

```
for i=1:3
  y = fftw(y,-1,dim(i),incr(i));
end

//M-D transform -new calling sequence-
//More efficient than old
y = fftw(a,-1,[5 6 4],[1 5 30]);
b = fftw(y,1,[5 6 4],[1 5 30]);
```

See Also

fftw_flags, get_fftw_wisdom, set_fftw_wisdom, fftw_forget_wisdom

Bibliography

Matteo Frigo and Steven G. Johnson, "FFTW Manual fo version 3.1.2" June 2006. Available : http://www.fftw.org

Name

fftw_flags — set computation method of fast fourier transform of the fftw function

```
[a,[S]]=fftw_flags([x1;x2;...])
```

Parameters

[x1;x2;...]
: Matrix of string or integers. Entry to switch the method of fft computation for fftw.

a
: Integer. Give the current value of the flag of the fftw function.

S
: String matrix. Give the string value of the fftw flag.

Description

This function enables the change of the `unsigned flags` parameter of the `fftw_plan_guru_split_dft` function that is used in fftw function.

Default value is FFTW_ESTIMATE

Accepted entries are :

- FFTW_MEASURE or 0
- FFTW_DESTROY_INPUT or 1
- FFTW_UNALIGNED or 2
- FFTW_CONSERVE_MEMORY or 4
- FFTW_EXHAUSTIVE or 8
- FFTW_PRESERVE_INPUT or 16
- FFTW_PATIENT or 32
- FFTW_ESTIMATE or 64
- FFTW_ESTIMATE_PATIENT or 128
- FFTW_BELIEVE_PCOST or 256
- FFTW_NO_DFT_R2HC or 512
- FFTW_NO_NONTHREADED or 1024
- FFTW_NO_BUFFERING or 2048
- FFTW_NO_INDIRECT_OP or 4096
- FFTW_ALLOW_LARGE_GENERIC or 8192
- FFTW_NO_RANK_SPLITS or 16384
- FFTW_NO_VRANK_SPLITS or 32768
- FFTW_NO_VRECURSE or 65536

- FFTW_NO_SIMD or 131072
- FFTW_NO_SLOW or 262144
- FFTW_NO_FIXED_RADIX_LARGE_N or 524288
- FFTW_ALLOW_PRUNING or 1048576

Rmk : when using FFTW_MEASURE/FFTW_PATIENT/FFTW_EXHAUSTIVE you must call two times fftw. (first call for initialisation, second and others calls for computation)

Examples

```
//return the integer value of the flag
fftw_flags()

//change flags
fftw_flags(["FFTW_MEASURE";"FFTW_CONSERVE_MEMORY"]);

//change flags and display current value of fftw flags (both integer and strin
[a,S]=fftw_flags("FFTW_PATIENT")
```

See Also

fftw

Name

fftw_forget_wisdom — Reset fftw wisdom

```
fftw_forget_wisdom()
```

Description

This function reset the current fftw wisdom.

Examples

```
//return fftw wisdom
txt=get_fftw_wisdom();
//set fftw wisdom
set_fftw_wisdom(txt);
//reset fftw wisdom
fftw_forget_wisdom();
```

See Also

fftw , get_fftw_wisdom , set_fftw_wisdom

Name

get_fftw_wisdom — return fftw wisdom

```
[txt]=get_fftw_wisdom()
```

Parameters

txt
: String matrix that contains fftw wisdom.

Description

This function return the fftw wisdom in a string matrix.

Examples

```
//return fftw wisdom
txt=get_fftw_wisdom();
//set fftw wisdom
set_fftw_wisdom(txt);
//reset fftw wisdom
fftw_forget_wisdom();
```

See Also

fftw , set_fftw_wisdom , fftw_forget_wisdom

Name

set_fftw_wisdom — set fftw wisdom

```
set_fftw_wisdom(txt)
```

Parameters

txt
: String matrix that contains fftw wisdom.

Description

This function set the fftw wisdom with a string matrix.

Examples

```
//return fftw wisdom
txt=get_fftw_wisdom();
//set fftw wisdom
set_fftw_wisdom(txt);
//reset fftw wisdom
fftw_forget_wisdom();
```

See Also

fftw , get_fftw_wisdom , fftw_forget_wisdom

Files : Input/Output functions

Name

basename — strip directory and suffix from filenames

```
files= basename(files[,flag [,flagexpand]])
```

Parameters

files
: a string matrix giving a set of file names.

flag,flagexpand
: boolean optional parameters. (default value `%t`).

files
: a string matrix.

Description

`basename` return the basename of the file entries given in files.

If `flag` is true the files are first converted to the target type given by the `MSDOS` variable. Moreover, if `flagexpand` is true leading strings like `HOME`, `SCI` or ~ are expanded using environment variables.

Note that `basename(files,%f)` can give erroneous results if pathnames given in files do not follow the convention given by the `MSDOS` variable.

Examples

```
files=basename('SCI/modules/fileio/macros/poo.sci')
files=basename('SCI/modules\fileio/macros/poo.sci')
files=basename('SCI/modules\fileio/macros/poo.sci.k')
```

See Also

listfiles , pathconvert

Name

copyfile — Copy file

```
copyfile('source','destination')
[status,message] = copyfile('source','destination')
```

Description

copyfile('source','destination') copies the file to the file or directory destination.

[status,message] = copyfile('source','destination') copies source to destination, returning the status and a message.

Examples

```
copyfile(SCI+"/etc/scilab.start",TMPDIR+"/scilab.start")
[status,message] = copyfile(SCI+"/etc/scilab.start",TMPDIR);
```

See Also

mdelete

Authors

Allan CORNET

Name

createdir — Make new directory

```
createdir('dirname')
status = createdir('dirname')
```

Description

createdir('dirname') creates the directory dirname in the current directory, if dirname is not in the current directory, specify the relative path to the current directory or the full path for dirname.

[status] = createdir('dirname') creates the directory dirname in the existing directory parentdir, returning the status, a message. Here, status is %T for success and %F otherwise.

createdir is used by mkdir.

Examples

```
createdir(SCIHOME+'/Directory_test')
removedir(SCIHOME+'/Directory_test')
```

See Also

mkdir , rmdir

Authors

A.C

Name

deletefile — delete a file

```
f = deletefile(filename)
```

Parameters

filename
: a file name

f
: %t or %f

Description

delete a file

Examples

```
fd=mopen(TMPDIR+'/filetodelete.txt','wt');
mclose(fd);
if (fileinfo(TMPDIR+'/filetodelete.txt') <> []) then deletefile(TMPDIR+'/fileto
```

Authors

A.C

Name

dir — get file list

```
dir path
S=dir([path])
```

Parameters

path
: a string matrix giving a directory pathname (eventually ended by a pattern built with `*`). Default value is `.`

S
: a `tlist` of type `dir` with fields : `name`, `date` and `isdir`

Description

`dir` can be used to get the files which match the patterns given by the path argument. Patterns are given to the unix `ls` or to the windows `dir` commands in order to get information on files. Thus in order to write portable Scilab script valid wildcard patterns for both os are to be given. Note that Pathname conversion is performed and for example `SCI/modules/core/macros/*.sci` is a valid pattern for both unix and windows.

The `name` field of the returned variable is the column vector of the file names.

The `date` field of the returned variable is the column vector of integers containing a last modification date coded in second from 1 Jan 1970).

The `isdir` field of the returned variable is the column vector of boolean true if the corresponding name is a directory.

The default display of the returned structure is a column formatted list of files. It can be changed redefining the function `%dir_p`

Examples

```
dir
dir SCI/modules/core/macros/*.bin
x=dir('SCI/modules/core/macros/*.bin')
dt=getdate(x.date);
mprintf("%s: %04d-%02d-%02d %02d:%02d:%02d\n",x.name,dt(:,[1 2 6 7:9]))
```

See Also

listfiles , findfiles , ls , fileinfo , date

Name

dirname — get directory from filenames

```
dirs= dirname(files[,flag [,flagexpand]])
```

Parameters

files
: a string matrix giving a set of file names.

flag,flagexpand
: boolean optional parameters. (default value `%t`).

files,dir
: string matrices.

Description

`dirname` return the dirname of the file entries given in files.

If `flag` is true the files are first converted to the target type given by the `MSDOS` variable. Moreover, if `flagexpand` is true leading strings like `HOME`, `SCI` or ~ are expanded using environment variables.

Note that `dirname(files,%f)` can give erroneous results if pathnames given in files do not follow the convention given by the `MSDOS` variable.

Examples

```
files=dirname('SCI/modules/fileio/macros/poo.sci')
files=dirname('SCI/modules\fileio/macros/poo.sci')
files=dirname('SCI/modules\fileio/macros/poo.sci.k')
```

See Also

basename , listfiles , pathconvert

Name

dispfiles — display opened files properties

```
dispfiles([units])
```

Parameters

units
: a vector of numbers, the file's logical units. By default all opened files.

Description

dispfiles displays properties of currently opened files.

Examples

```
dispfiles()
```

See Also

file , mopen

Authors

S. Steer

Name

fileext — returns extension for a file path

```
extension = fileext(fullpath)
```

Parameters

fullpath
: a character string, the given file path

extension
: a character string, the extension part is any or ''

Description

`extension=fileext(fullpath)` splits the fullpath character string in the extension part including the dot.

Examples

```
extension = fileext('SCI/etc/scilab.start')
   extension = fileext(['SCI/etc/scilab.start';'SCI/etc/scilab.quit'])
```

See Also

fileparts

Authors

Allan CORNET

Name

fileparts — returns the path, filename and extension for a file path

```
[path,fname,extension]=fileparts(fullpath)
v=fileparts(fullpath,sel)
```

Parameters

fullpath
: a character string, the given file path

sel
: a optional character string selector, with posible values: 'path' 'fname' or 'extension'

path
: a character string, the path of the directory pointed to by fullpath

fname
: a character string, the filename part is any or ''

extension
: a character string, the extension part is any or ''

value
: a character string, depending on sel value

Description

`[path,fname,extension]=fileparts(fullpath)` splits the fullpath character string in its three parts: the path of the directory pointed to, the filename part, the extension part including the dot.

Examples

```
[path,fname,extension]=fileparts('SCI/etc/scilab.start')
fileparts('SCI/etc/scilab.start','extension')
```

See Also

pathconvert , basename , fullfile

Authors

Serge Steer, INRIA

Name

filesep — returns directory separator for current platform

```
s = filesep()
```

Parameters

s
: a string

Description

returns directory separator. ('/' on Linux or '\' on Windows)

Examples

```
filesep()
```

Authors

A.C

Name

findfiles — Finding all files with a given filespec

```
f = findfiles()
f=findfiles(path)
f=findfiles(path,filespec)
```

Parameters

path
: a path

filespec
: a spec file. example "*.sce"

f
: returns a string matrix of filenames

Description

Finding all files with a given filespec

Examples

```
f=findfiles()
f=findfiles(SCI)
f=findfiles(SCI+'/modules/core/macros','*.sci')
```

See Also

listfiles

Authors

A.C

Name

fprintf — Emulator of C language fprintf function

```
fprintf(file,format,value_1,..,value_n)
```

Parameters

format
: a Scilab string. Specifies a character string combining literal characters with conversion specifications.

value_i
: Specifies the data to be converted according to the format parameter.

str
: column vector of character strings

file
: a Scilab string specifying a file name or a logical unit number (see `file`)

  Note that is file=0, the message will be display on standard error stream (stderr).

Description

Obsolete function, use preferabily the mfprintf function which is much more compatible with the C `fprintf` functionalities.

The `fprintf` function converts, formats, and writes its `value` parameters, under control of the `format` parameter, to the file specified by its `file` parameter.

The `format` parameter is a character string that contains two types of objects:

Literal characters
: which are copied to the output stream.

Conversion specifications
: each of which causes zero or more items to be fetched from the `value` parameter list. see printf_conversion for details

If any `value`s remain after the entire `format` has been processed, they are ignored.

Examples

```
u=file('open','results','unknown') //open the result file
t=0:0.1:2*%pi;
for tk=t
 fprintf(u,'time = %6.3f value = %6.3f',tk,sin(tk)) // write a line
end
file('close',u) //close the result file


fprintf(0,'My error which is going to be displayed on the stderr')
```

See Also

mfprintf , string , print , write , format , disp , file , printf , sprintf , printf_conversion

Name

fprintfMat — Write a matrix in a file.

```
fprintfMat(file,M [,format,text])
```

Parameters

fil
: a string, the pathname of the file to be written.

M
: A matrix of real numbers.

format
: a character string, a C like format. This is an optional parameter, the default value is `"%f"`

text
: a string matrix giving non numerical comments stored at the beginning of the file.

Description

The `fprintfMat` function writes a matrix in a formated file. Each row of the matrix give a line in the file. If `text` is given then the elements of `text` are inserted columnwise at the beginning of the file one element per line.

Examples

```
n=50;
a=rand(n,n,'u');
fprintfMat(TMPDIR+'/Mat',a,'%5.2f');
a1=fscanfMat(TMPDIR+'/Mat');
```

See Also

mclose , meof , mfprintf , mfscanf , fscanfMat , mget , mgetstr , mopen , mprintf , mput , mputstr , mscanf , mseek , mtell , mdelete

Name

fscanf — Converts formatted input read on a file

```
[v_1,...v_n]=fscanf (file,format)
```

Parameters

format
: Specifies the format conversion.

file
: Specifies the input file name or file number.

Description

The fscanf functions read character data on the file specified by the `file` argument , interpret it according to a format, and returns the converted results.

The format parameter contains conversion specifications used to interpret the input.

The format parameter can contain white-space characters (blanks, tabs, newline, or formfeed) that, except in the following two cases, read the input up to the next nonwhite-space character. Unless there is a match in the control string, trailing white space (including a newline character) is not read.

- Any character except % (percent sign), which must match the next character of the input stream.
- A conversion specification that directs the conversion of the next input field. see scanf_conversion for details.

See Also

printf , read , scanf , sscanf , mfscanf , scanf_conversion

Name

fscanfMat — Reads a Matrix from a text file.

```
M=fscanfMat(filename);
[M,text]=fscanfMat(filename);
```

Parameters

filename
: a character string giving the name of the file to be scanned.

M
: Output variable. A matrix of real numbers.

text
: Output variable. A string matrix.

Description

The `fscanfMat` function is used to read a scalar matrix from a text file. The first non-numeric lines of the file are returned in `text` if requested and all the remaining lines must have the same number of columns (column separator are assumed to be white spaces or tab characters). The number of columns of the matrix will follow the number of columns found in the file and the number of lines is fetched by detecting eof in the input file. This function can be used to read back numerical data saved with the `fprintfMat`.

Examples

```
fd=mopen(TMPDIR+'/Mat','w');
mfprintf(fd,'Some text.....\n');
mfprintf(fd,'Some text again\n');
a=rand(6,6);
for i=1:6 ,
 for j=1:6, mfprintf(fd,'%5.2f ',a(i,j));end;
 mfprintf(fd,'\n');
end
mclose(fd);
a1=fscanfMat(TMPDIR+'/Mat')
```

See Also

mclose , meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetstr , mopen , mprintf , mput , mputstr , mscanf , mseek , mtell , mdelete

Name

fullfile — Build a full filename from parts

```
f = fullfile(varargin)
```

Parameters

varargin
: all directories and filename used to build the full filename (at least one directory and filename)

f
: full filename

Description

`f = fullfile(varargin)` builds a full filename taking care of platform on which it is run and handling the cases when the directories begin or end with a directory separator.

Examples

```
f=fullfile("/home/","\scilab","macros","\util","fullfile.sci")
f=fullfile("C:","\scilab","macros","\util","fullfile.sci")
```

See Also

pathconvert , fileparts

Authors

V.C.

Name

fullpath — Creates an full path name for the specified relative path name.

```
res = fullpath(relative_path)
```

Parameters

res
: a string

relative_path
: a string

Description

Creates an full path name for the specified relative path name.

On linux 'relative_path' needs to exist.

Examples

```
mkdir(TMPDIR+'/niv1');
mkdir(TMPDIR+'/niv1/niv2');
mputl(' ',TMPDIR+'/niv1/test.txt');
cd(TMPDIR+'/niv1/niv2');
fullpath('../test.txt')
```

Authors

A.C

Name

getdrives — Get the drive letters of all mounted filesystems on the computer.

```
drives = getdrives()
```

Parameters

drives
: a matrix of strings

Description

Get the drive letters of all mounted filesystems on the computer.

returns the roots of all mounted filesystems on the computer as a matrix of strings.

For Linux this list consists solely of the root directory, / .

Examples

```
getdrives()
```

Authors

A.C

Name

getlongpathname — get long path name (Only for Windows)

```
longpath=getlongpathname(shortpath)
[longpath,bOK]=getlongpathname(shortpath)
```

Parameters

shortpath
: A character string the short path

longpath
: A character string the long path

bOK
: A boolean %T if path has been converted else %F

Description

The getlongpathname primitive converts the specified path to its long form. It no long path is found, this primitive returns the specified name.

Examples

```
[longpath,bOK]=getlongpathname(SCI)
```

See Also

getshortpathname

Authors

Allan CORNET

Name

getshortpathname — get short path name (Only for Windows)

```
shortpath=getshortgpathname(longpath)
[shortpath,bOK]=getshortpathname(longpath)
```

Parameters

longpath
: A character string the long path

shortpath
: A character string the short path

bOK
: A boolean %T if path has been converted else %F

Description

The getshortpathname primitive converts the specified path to its short form.

Examples

```
[shortpath,bOK]=getshortpathname(SCI)
```

See Also

getlongpathname

Authors

Allan CORNET

Name

isdir — checks if argument is a directory path

```
r=isdir(path)
```

Parameters

path
: a character string, the file pathname

r
: a boolean, true if `path` is a path to a directory.

Description

`r=isdir(path)` checks if `path` is a path to a directory.

Reference

This function is based on the C function stat. The `SCI` and ~ shortcuts for Scilab directory and home directory are handled.

Examples

```
isdir(TMPDIR)
isdir SCI/etc/scilab.start
```

See Also

fileinfo

Authors

S. Steer INRIA

Name

listfiles — list files

```
files= listfiles(paths [,flag,flagexpand])
```

Parameters

paths
: a string matrix giving a set of pathnames (eventually ended by a pattern built with *)

flag,flagexpand
: boolean optional parameters. (default value %t).

files
: a string matrix.

Description

`listfiles` can be used to list the files which match the patterns given by one of the paths entries. Patterns are given to the unix `ls` or to the windows `dir` commands in order to get information on files. Thus in order to write portable Scilab script valid wildcard patterns for both os are to be given. Note that Pathname conversion is performed and for example `SCI/core/macros/*.sci` is a valid pattern for both unix and windows.

if `flag` is true the pathnames given by `paths` are converted according to the MSDOS value (See `pathconvert`). Moreover, if `flagexpand` is true leading strings like `HOME`, `SCI` or ~ are expanded using environment variables.

Examples

```
files=listfiles(['SCI/modules/core/macros/*.sci';'SCI/modules/core/macros/*.bin
```

See Also

findfiles , basename , pathconvert

Name

listvarinfile — list the contents of a saved data file

```
listvarinfile(filename)
[nams,typs,dims,vols]=listvarinfile(filename)
```

Parameters

filename
: character string, the pathname of the file to be inspected

nams
: character array, names of the variables saved in the file

dims
: list, dimensions of the variables saved in the file

typs
: numeric array, types of the variables saved in the file

vols
: numeric array, size in bytes of the variables saved in the file

Description

- This utility function lists "a la whos" the variables contained in a Scilab data file produced by save.

  Remark: hypermatrices are reported as plain mlists; rationals and state-spaces are reported as plain tlists; graphic handles are not recognized.

Examples

```
a=eye(2,2); b=int16(ones(a)); c=rand(2,3,3);
save("vals.dat",a,b,c)
listvarinfile("vals.dat")
```

See Also

whos , save , load , save_format , type

Authors

Serge Steer
: 31 Jan 2001; reediting by Enrico Segre

Name

ls — show files

```
ls path options
files=ls( [path] )
```

Parameters

path
: a string matrix giving a directory pathname (eventually ended by a pattern built with `*`). Default value is `.`

files
: a string column vector. By default it contains a column formatted output. if one of the option is `'-1'`. `files` contains an entry for each files

Description

`ls` can be used to list the files which match the patterns given by the path argument. Patterns are given to the unix `ls` or to the windows `dir` commands in order to get information on files. Thus in order to write portable Scilab script valid wildcard patterns for both os are to be given. Note that Pathname conversion is performed and for example `SCI/modules/core/macros/*.sci` is a valid pattern for both unix and windows.

If you want to get a vector of all files matching a pattern use preferabily the listfiles or the dirfunction.

Please note that starting from the version 5.0 of Scilab, the second input argument has been removed (a sequence of strings which can be added under Unix systems: the Unix ls command options). This option has been removed mainly for security and portability reasons.

Examples

```
ls
ls SCI/modules/core/macros/*.sci
x=ls('SCI/modules/core/macros/*.sci')
```

See Also

listfiles , findfiles , dir , fileinfo

Name

maxfiles — sets the limit for the number of files a scilab is allowed to have open simultaneously.

```
r= maxfiles(newnumbermax)
```

Parameters

newnumbermax
: a integer the new value

r
: effective new value.

Description

sets the limit for the number of files a scilab is allowed to have open simultaneously.

Minimum : 20

Maximum : 100

Default : 20

Examples

```
r=maxfiles(50);
```

See Also

mopen

Name

mclearerr — reset binary file access errors

```
mclearerr([fd])
```

Parameters

fd

scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

Description

The function `clearerr` is used to resets the error indicator and EOF indicator to zero.

See Also

merror , mclose , mopen , mput , mget , mgetstr , mputstr , meof , mseek , mtell

Name

mclose — close an opened file

```
err=mclose([fd])
mclose('all')
```

Parameters

fd
: scalar. The `fd` parameter returned by the function `mopen` is used as a file descriptor (it's a positive integer).

err
: a scalar. Error indicator : vector

Description

`mclose` must be used to close a file opened by `mopen`. If fd is omitted `mclose` closes the last opend file.

`mclose('all')` closes all files opened by `file('open',..)` or `mopen`. Be careful with this use of `mclose` because when it is used inside a Scilab script file, it also closes the script and Scilab will not execute commands written after `mclose('all')`.

See Also

meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetl , mgetstr , mopen , mprintf , mput , mputl , mputstr , mscanf , mseek , mtell , file , mdelete

Name

mdelete — Delete file(s)

```
mdelete(filename)
```

Parameters

filename
: a character string. The pathname of the file(s) to delete.

Description

`mdelete` may be used to delete a file or a set of files if `filename` contains meta-charaters.

Note that `mdelete` does not ask for confirmation when you enter the delete command. To avoid accidentally losing files, make sure that you have accurately specified the items you want deleted.

See Also

mopen , mclose , meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetstr , mopen , mprintf , mput , mputstr , mscanf , mseek , mtell

Name

meof — check if end of file has been reached

```
err=meof(fd)
```

Parameters

fd
: scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

err
: scalar. Error indicator

Description

The function `meof` will return a non null value if end of file has been reached in a previous call to `mget` or `mgetstr`. The function `clearerr` is used to reset the error flag and EOF flag to zero.

See Also

mclose , meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetstr , mopen , mprintf , mput , mputstr , mscanf , mseek , mtell , mdelete

Name

merror — tests the file access errors indicator

```
merror([fd])
```

Parameters

fd

scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

Description

The function `merror` is used to tests the file access errors indicator. returning non-zero if it is set. The error indicator can only be reset by the `mclearerr` function.

See Also

mclearerr , mclose , mopen , mput , mget , mgetstr , mputstr , meof , mseek , mtell

Name

mscanf — reads input from the standard input (interface to the C scanf function)
mfscanf — reads input from the stream pointer stream (interface to the C fscanf function)
msscanf — reads its input from the character string (interface to the C sscanf function)

```
[n,v_1,...v_n]=mfscanf([niter,]fd,format)
L=mfscanf([niter,] fd,format)

[n,v_1,...v_n]=mscanf([niter,] format)
L=mscanf([niter,]format)

[n,v_1,...v_m]=msscanf([niter,]str,format)
L=msscanf([niter,] str,format)
```

Parameters

format
: a Scilab string describing the format to use to write the remaining operands. The format operand follows, as close as possible, the C printf format operand syntax as described in scanf_conversion.

fd
: :The fd parameter returned by the function `mopen` is used as a file descriptor (it's a positive integer). The value -1 refers to the last opened file.

str
: a Scilab string or string vector.

niter
: an integer, the number of times the format as to be used.

n
: an integer, the number of data read or -1 if EOL has been encountered before any datum has been read.

v_i
: Each function reads characters, interprets them according to a format, and stores the results in its output arguments. If more than n output arguments are provided, the last ones `v_n+1,...v_m` are set to empty matrices.

L
: if all data are homogeneous they are stored in a unique vector which is returned, otherwise subsequences of same data type are stored in matrices and an `mlist` (with type `cblock`) containing all the built matrices is returned.

Description

The `mfscanf` function reads characters from the stream `fd`.

The `mscanf` function reads characters from Scilab window.

The `msscanf` function reads characters from the Scilab string `str`.

The `niter` optional argument specifies how many time the format has to used. One iteration produces one line in the output matrix. If `niter==-1` the function iterates up to the end of file. The `niter` default value is 1.

comments about precision :

mfscanf is based on C function fscanf. If you use '%f', '%g', '%e' as format your datas will be cast to float and returned in a scilab variable.

This scilab variable is a double then you can have some precision errors. In this case, it is better to use '%lg' format.

Examples

```
//-----------------------------------------------------------
//--    Simple use                                         --
//-----------------------------------------------------------
s='1 1.3'  //a string
[n,a,b]=msscanf(s,"%i %e")
L=msscanf(s,"%i %e")

//-----------------------------------------------------------
//--    Formats samples                                    --
//-----------------------------------------------------------

msscanf(" 12\n",'%c%c%c%c') //scan characters

msscanf('0xabc','%x') //scan with hexadecimal format

msscanf('012345abczoo','%[0-9abc]%s')  //[] notation

// reading float and double
msscanf('4345.988','%g')-4345.988 // scan as float
msscanf('4345.988','%lg')-4345.988 // scan as double

//-----------------------------------------------------------
//--    scanning multi-line data files                     --
//-----------------------------------------------------------
//create a file with data
u=mopen(TMPDIR+'/foo','w');
t=(0:0.1:%pi)';mfprintf(u,"%6.3f %6.3f\n",t,sin(t))
mclose(u);

u=mopen(TMPDIR+'/foo','r'); // open the file for reading
//read the file line by line
[n,a,b]=mfscanf(u,'%e %e') //first line using mutiple LHS syntax
l=mfscanf(u,'%e %e')       //second one using single LHS syntax
//use niter to read 5 more lines
l=mfscanf(5,u,'%e %e')

//use niter=-1 to read up to the end of file
l=mfscanf(-1,u,'%e %e')

mclose(u); //close the file

//-----------------------------------------------------------
//--    scanning multi-line  strings vectors               --
//-----------------------------------------------------------
//use niter to scan a string vector
[n,Names,Ages]=msscanf(-1,["Alain 19";"Pierre 15";"Tom 12"],'%s %d')
D=msscanf(-1,["Alain 19";"Pierre 15";"Tom 12"],'%s %d')
typeof(D)
Names=D(:,1) //strings
```

```
Age=D(:,2)   //numerical values
```

See Also

mclose, meof, mfprintf, fprintfMat, mfscanf, fscanfMat, mget, mgetstr, mopen, mprintf, mput, mputstr, mscanf, mseek, mtell, mdelete, scanf_conversion

Name

mget — reads byte or word in a given binary format and convert to double
mgeti — reads byte or word in a given binary format return an int type

```
x=mget([n,type,fd])
x=mgeti([n,type,fd])
```

Parameters

n
: a positive scalar: The number of items to be read.

fd
: a scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

type
: a string. Give the binary format used to write all the entries of x.

x
: a vector of floating point or integer type numbers

Description

The `mget` function reads data in the input specified by the stream parameter `fd` and returns a vector of floating point data. The `mgeti` function reads data in the input specified by the stream parameter `fd` and returns a vector of integer data.

Data is read at the position at which the file pointer is currently pointing and advances the indicator appropriately.

The `type` parameter is a conversion specifier which may be set to any of the following flag characters (with default value "l"):

Note , On Windows, default behavior is to skip byte 13 (0x0D). mopen should be called with the `b` option, e.g. fd1=mopen(file1,'rb') , so that all bytes will be read without exception.

Data type:

d
: double

f
: float

l
: long

i
: int

s
: short

c
: character

Optional flag:

u..
: unsigned (in combination with one of the above types)

..l
: little endian (in combination with one of the above types)

..b
: big endian (in combination with one of the above types)

Bytes read are automatically swapped if necessary (by checking little=endian status).

This default swapping behavior can be suprressed by adding a flag in the mopen function.

Formats "l", "d", and "f" are only valid with the mget function.

Examples

```
file1 = 'test1.bin';
file2 = 'test2.bin';
fd1=mopen(file1,'wb');
fd2=mopen(file2,'wb');
mput(1996,'ull',fd1);
mput(1996,'ull',fd2);
mclose(fd1);
mclose(fd2);

fd1=mopen(file1,'rb');
if 1996<>mget(1,'ull',fd1) ;write(%io(2),'Bug');end;
fd2=mopen(file2,'rb');
if 1996<>mget(1,'ull',fd2) ;write(%io(2),'Bug');end;
mclose(fd1);
mclose(fd2);
```

See Also

mclose, meof, mfprintf, fprintfMat, mfscanf, fscanfMat, mgetl, mgetstr, mopen, mprintf, mput, mputl, mputstr, mscanf, mseek, mtell, mdelete

Name

mgetl — read lines from an ascii file

```
txt=mgetl(file_desc [,m])
```

Parameters

file_desc
: :a character string giving the file name or a logical unit returned by mopen

m
: an integer scalar. number of lines to read. Default value is -1.

txt
: a column vector of string

Description

`mgetl` function allows to read a lines from an ascii file.

If `m` is omitted or is `-1` all lines till end of file occurs are read.

If `m` is given `mgetl` tries to read exactly m lines. This option is useful to sequentialy read part of a file. In this case if an end of file (EOF) occurs before `m` lines are read the read lines are returned (it is possible to check if EOF had occured using the meof function) issued.

`mgetl` allows to read files coming from Unix, Windows, or Mac operating systems.

Examples

```
mgetl('SCI/etc/scilab.start',5)

mgetl SCI/modules/elementary_functions/macros/erf.sci

fd=mopen('SCI/etc/scilab.start','r')
mgetl(fd,10)
mclose(fd)
```

See Also

mputl , mclose , mfscanf , mget , mput , mgetstr , mopen , read

Authors

S. Steer

Name

mgetstr — read a character string

```
str=mgetstr(n [,fd] )
```

Parameters

n
: :a positive scalar: The number of character to read.

fd
: scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

str
: a character string

Description

mgetstr function allows to read a character string in a binary file. If EOF is reached before read completion only the properly read values will be returned.

See Also

mclose , meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetstr , mopen , mprintf , mput , mputstr , mscanf , mseek , mtell , mdelete

Name

mkdir — Make new directory

```
mkdir('dirname')
mkdir('parentdir','newdir')
status=mkdir( ... )
[status,msg]=mkdir( ... )
```

Description

mkdir('dirname') creates the directory dirname in the current directory, if dirname represents a relative path. Otherwise, dirname represents an absolute path and mkdir attempts to create the absolute directory dirname

mkdir('parentdir','dirname') creates the directory dirname in the existing directory parentdir, where parentdir is an absolute or relative pathname.

[status,message] = mkdir(...,'dirname') creates the directory dirname in the existing directory parentdir, returning the status, a message. Here, status is 1 for success, 2 if it already exists, -2 if it is a filename and 0 otherwise.

Examples

```
// Absolute pathname
mkdir(TMPDIR+"/mkdir_example_1")
status_2 = mkdir(TMPDIR+"/mkdir_example_2")
[status_3,msg_3] = mkdir(TMPDIR+"/mkdir_example_3")

// Absolute pathname (parentdir + dirname)
[status_4,msg_4] = mkdir(TMPDIR,"mkdir_example_4")

// Relative pathname
cd TMPDIR;
[status_5,msg_5] = mkdir("mkdir_example_5")
[status_6,msg_6] = mkdir("mkdir_example_5/mkdir_example_6")
```

See Also

cd, dir, rmdir

Authors

A.C

Name

mopen — open a file

```
[fd,err]=mopen(file [, mode, swap ])
```

Parameters

file
: a character string. The pathname of the file to open.

mode
: a character string that controls whether the file is opened for reading (r), writing (w), or appending (a) and whether the file is opened for updating (+). The mode can also include a b parameter to indicate a binary file.

swap
: a scalar. If `swap` is present and `swap=0` then automatic bytes swap is disabled.

err
: a scalar. Error indicator

fd
: scalar. The `fd` parameter returned by the function `mopen` is used as a file descriptor (it's a positive integer).

Description

`mopen` may be used to open a file in a way compatible with the C `fopen` procedure. Without swap argument the file is supposed to be coded in "little endian IEEE format" and data are swaped if necessary to match the IEEE format of the processor.

The mode parameter controls the access allowed to the stream. The parameter can have one of the following values. In this list of values, the b character indicates a binary file

r
: Opens the file for reading.

rb
: Opens a binary file for reading.

rt
: Opens a text file for reading.

w
: Creates a new file for writing, or opens and truncates a file to zero length.

wb
: Creates a new binary file for writing, or opens and truncates a file to zero length.

wt
: Creates a text binary file for writing, or opens and truncates a file to zero length.

a or ab
: Appends (opens a file for writing at the end of the file, or creates a file for writing).

r+ or r+b
: Opens a file for update (reading and writing).

w+ or w+b
: Truncates to zero length or creates a file for update.

a+ or a+b
: :Appends (opens a file for update, writing at the end of the file, or creates a file for writing).

When you open a file for update, you can perform both input and output operations on the resulting stream. However, an output operation cannot be directly followed by an input operation without a file-positioning operation (mseek() function). Also, an input operation cannot be directly followed by an output operation without an intervening file positioning operation, unless the input operation encounters the end of the file.

When you open a file for append (that is, when the mode parameter is a or a+), it is impossible to overwrite information already in the file. You can use the fseek() function to reposition the file pointer to any position in the file, but when output is written to the file, the current file pointer is ignored. All output is written at the end of the file and the file pointer is repositioned to the end of the output.

To open files in a way compatible with Fortran like functions use function `file`.

See Also

mclose, meof, mfprintf, fprintfMat, mfscanf, fscanfMat, mget, mgetl, mgetstr, mopen, mprintf, mput, mputl, mputstr, mscanf, mseek, mtell, mdelete

Name

mfprintf — converts, formats, and writes data to a file
mprintf — converts, formats, and writes data to the main scilab window
msprintf — converts, formats, and writes data in a string

```
mfprintf(fd,format,a1,...,an);
mprintf(format,a1,...,an);
str=msprintf(format,a1,...,an);
```

Parameters

fd
: scalar, file descriptor given by `mopen` (it's a positive integer).

  if fd equals 0 redirection in stderr.

  if fd equals 6 redirection in stdout.

  OBSOLETE :The value `-1` refers to the default file (i.e the last opened file).

format
: a Scilab string describing the format to use to write the remaining operands. The format operand follows, as close as possible, the C printf format operand syntax.

str
: a character string, string to be scanned.

a1,...,an
: Specifies the data to be converted and printed according to the format parameter.

Description

The `mprintf`, `mfprintf`, and `msprintf` functions are interface for C-coded version of `printf`, `fprintf` and `sprintf` functions.

The `mprintf` function writes formatted operands to the standard Scilab output (i.e the Scilab window). The argument operands are formatted under control of the format operand.

The `mfprintf` function writes formatted operands to the file specified by the file desciptor `fd`. The argument operands are formatted under control of the format operand.

The `msprintf` writes formatted operands in its returned value (a Scilab string). The argument operands are formatted under control of the format operand. Note that, in this case, the escape sequences (`"\n, \t,.."`) are treated as a normal sequence of characters.

All these functions may be used to output column vectors of numbers and string vectors without an explicit loop on the elements. In that case these functions iterates on the rows. The shortest vector gives the number of time the format has to be iterated.

An homogeneous sequence of identical type parameters can be replaced by a matrix

Examples

```
mprintf('At iteration %i, Result is:\nalpha=%f',33,0.535)

msprintf('%5.3f %5.3f',123,0.732)
msprintf('%5.3f\n%5.3f',123,0.732)
```

```
A=rand(5,2);
// vectorized forms: the format directive needs
// two operand, each column of A is used as an operand.
// and the mprintf function is applied on each row of A
mprintf('%5.3f\t%5.3f\n',A)

colors=['red';'green';'blue';'pink';'black'];
RGB=[1 0 0;0 1 0;0 0 1;1 0.75 0.75;0 0 0];
mprintf('%d\t%s\t%f\t%f\t%f\n',(1:5)',colors,RGB)

fd = mopen(TMPDIR+'/text.txt','wt');
mfprintf(fd,'hello %s %d.\n','world',1);
mfprintf(fd,'hello %s %d.\n','scilab',2);
mclose(fd);
scipad(TMPDIR+'/text.txt')
```

See Also

mclose, meof, mfprintf, fprintfMat, mfscanf, fscanfMat, mget, mgetstr, mopen, mprintf, mput, mputstr, mscanf, mseek, mtell, mdelete, printf_conversion

Name

mput — writes byte or word in a given binary format

```
mput(x [,type,fd])
```

Parameters

x
: a vector of floating point or integer type numbers

fd
: scalar. The `fd` parameter returned by the function. Default value is -1 which stands for the last (`mopen`) opened file.

type
: a string. Give the binary format used to write all the entries of x.

Description

The `mput` function writes data to the output specified by the stream parameter `fd`. Data is written at the position at which the file pointer is currently pointing and advances the indicator appropriately.

The `type` parameter is a conversion specifier which may be set to any of the following flag characters (with default value "l"):

"l","i","s","ul","ui","us","d","f","c","uc"
: for writing respectively a long, an int, a short, an unsigned long, an unsigned int, an unsigned short, a double, a float, a char and an unsigned char. The bytes which are wrote are automatically swapped if necessary (by checking little-endian status) in order to produce machine independent binary files (in little-endian mode). This default swapping mode can be suppressed by adding a flag in the `mopen` function.

"..l" or "..b"
: It is also possible to write in little-endian or big-endian mode by adding a 'l' or 'b' character at the end of a type specification. For example "db" will write a double in big-endian mode.

Examples

```
  filen = 'test.bin';
  mopen(filen,'wb');
  mput(1996,'l');mput(1996,'i');mput(1996,'s');mput(98,'c');
// force little-endian
  mput(1996,'ll');mput(1996,'il');mput(1996,'sl');mput(98,'cl');
// force big-endian
  mput(1996,'lb');mput(1996,'ib');mput(1996,'sb');mput(98,'cb');
//
  mclose();
  mopen(filen,'rb');
  if 1996<>mget(1,'l') then pause,end
  if 1996<>mget(1,'i') then pause,end
  if 1996<>mget(1,'s') then pause,end
  if   98<>mget(1,'c') then pause,end
  // force little-endian
  if 1996<>mget(1,'ll') then pause,end
```

```
if 1996<>mget(1,'il') then pause,end
if 1996<>mget(1,'sl') then pause,end
if   98<>mget(1,'cl') then pause,end
// force big-endian
if 1996<>mget(1,'lb') then pause,end
if 1996<>mget(1,'ib') then pause,end
if 1996<>mget(1,'sb') then pause,end
if   98<>mget(1,'cb') then pause,end
//
mclose();
```

See Also

mclose , meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetl , mgetstr , mopen , mprintf , mputl , mputstr , mscanf , mseek , mtell , mdelete

Name

mputl — writes strings in an ascii file

```
r = mputl(txt ,file_desc)
```

Parameters

r
: returns %t or %f to check if function has correctly written on the file.

file_desc
: A character string giving the name of the file or a logical unit returned by mopen.

  OBSOLETE : If omitted, lines are written in the last file opened by mopen.

txt
: a vector of strings.

Description

`mputl` function allows to write a vector of strings as a sequence of lines in an ascii file.

Examples

```
fd = mopen(TMPDIR+'/text_mputl.txt','wt');
mputl('Hello World',fd);
mclose(fd);

fd = mopen(TMPDIR+'/text_mputl.txt','rt');
disp(mgetl(fd));
mclose(fd);
```

See Also

mget, mgetl, mclose, mfprintf, mput, mputstr, mopen, write

Authors

S. Steer

Allan CORNET

Name

mputstr — write a character string in a file

```
mputstr(str [, fd]);
```

Parameters

fd
: scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

str
: a character string

Description

mputstr function allows to write a character string in a binary file.

See Also

mclose , meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetstr , mopen , mprintf , mput , mputstr , mscanf , mseek , mtell , mdelete

Name

mseek — set current position in binary file.

```
mseek(n [,fd, flag])
```

Parameters

n
: :a positive scalar: The offset from origin in number of bytes.

fd
: scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

flag
: a string. specifies the origin. Default value 'set'.

Description

The function `mseek()` sets the position of the next input or output operation on the stream `fd`. The new position is at the signed distance given by `n` bytes from the beginning, from the current position, or from the end of the file, according to the `flag` value which can be `'set'`, `'cur'` or `'end'`.

`mseek()` allows the file position indicator to be set beyond the end of the existing data in the file. If data is later written at this point, subsequent reads of data in the gap will return zero until data is actually written into the gap. `mseek()`, by itself, does not extend the size of the file.

Examples

```
file3='test3.bin'
fd1= mopen(file3,'wb');
for i=1:10, mput(i,'d'); end
mseek(0);
mput(678,'d');
mseek(0,fd1,'end');
mput(932,'d');
mclose(fd1)
fd1= mopen(file3,'rb');
res=mget(11,'d')
res1=[1:11]; res1(1)=678;res1($)=932;
if res1<>res ;write(%io(2),'Bug');end;
mseek(0,fd1,'set');
// trying to read more than stored data
res1=mget(100,'d',fd1);
if res1<>res ;write(%io(2),'Bug');end;
meof(fd1)
mclearerr(fd1)
mclose(fd1);
```

See Also

mclose , meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetstr , mopen , mprintf , mput , mputstr , mscanf , mseek , mtell , mdelete

Name

mtell — binary file management

```
mtell([fd])
```

Parameters

fd

scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

Description

The function `mtell()` returns the offset of the current byte relative to the beginning of the file associated with the named stream `fd`.

See Also

mclose , meof , mfprintf , fprintfMat , mfscanf , fscanfMat , mget , mgetstr , mopen , mprintf , mput , mputstr , mscanf , mseek , mtell , mdelete

Name

pathconvert — pathnames convertion between posix and windows.

```
paths=pathconvert(paths [,flagtrail [,flagexpand [,type]]])
```

Parameters

paths
: a string matrix giving a set of pathnames

flagtrail,flagexpand
: boolean optional parameters (default value depends on the MSDOS variable).

type
: a string `'u'` or `'w'`.

Description

`pathconvert` can be used to convert a set of pathnames (given by a string matrix `paths`) from windows native filename to posix-style pathnames and back. The target style is given by the optional string `type` which can be `'u'` for Unix or `'w'` for Windows. The default style is set according to the value of `MSDOS`. If `MSDOS` is true (resp. false) then default type is `'w'` (resp. `'u'`).

Windows pathnames starting with `name:` are converted to pathnames starting with `/cygdrive/name/` using the cygwin convention.

`flagtrail` is an optional boolean parameter. When its value is true a trailing separator (`'\'` or `'/'`) is added at the end of the path.

`flagexpand` is an optional boolean parameter. When its value is true leading strings like `HOME`, `SCI` or ~ are expanded using environment variables.

Examples

```
pathconvert('SCI/modules/fileio\macros/foo.sci',%f,%f,'u')
pathconvert('SCI/modules/fileio\macros/foo.sci',%f,%f,'w')
pathconvert('SCI/modules/fileio/macros/foo.sci',%f,%t,'w')
pathconvert('HOME/modules/fileio/macros/foo.sci',%t,%t,'w')
pathconvert('c:/tmp',%f,%t,'u')
pathconvert('/cygdrive/c/tmp',%f,%f,'w')
```

See Also

basename , listfiles

Name

pathsep — returns path separator for current platform

```
s = pathsep()
```

Parameters

s
: a string

Description

returns path separator. (':' on Linux or ';' on Windows)

Examples

```
pathsep()
```

Authors

A.C

Name

removedir — Remove a directory

```
removedir('dirname')
[status] = removedir('dirname','s')
```

Description

removedir('dirname') removes the directory dirname from the current directory. If the directory is not empty, files and subdirectories are removed. If dirname is not in the current directory, specify the relative path to the current directory or the full path for dirname.

[status] = removedir('dirname') removes the directory dirname and its contents from the current directory, returning the status. Here, status is %T for success and is %F for error.

removedir is used by rmdir.

Examples

```
createdir(SCIHOME+'/Directory_test')
removedir(SCIHOME+'/Directory_test')
```

See Also

mkdir , rmdir

Authors

A.C

Name

rmdir — Remove a directory

```
rmdir('dirname')
rmdir('dirname','s')
[status,message] = rmdir('dirname','s')
```

Description

rmdir('dirname') removes the directory dirname from the current directory. If the directory is not empty, you must use the s argument. If dirname is not in the current directory, specify the relative path to the current directory or the full path for dirname.

rmdir('dirname','s') removes the directory dirname and its contents from the current directory.

[status, message] = rmdir('dirname','s') removes the directory dirname and its contents from the current directory, returning the status, and a message. Here, status is 1 for success and is 0 for error.

Examples

```
mkdir(SCI,'Directory')
rmdir(SCI+'/Directory')
```

See Also

cd , dir , mkdir

Authors

A.C

Name

save_format — format of files produced by "save"

Description

Variables are saved by Scilab with the save function in the following format:

each variable record is appended consecutively to the file. The variable record begins with 6 long integer holding the variable name in encoded format (see the Remarks section below),

After that comes the variable type (long integer), then, depending on it, for:

Floating matrices (type 1)
row_size m (a long integer),

column_size n (a long integer),

real/complex flag it (a long integer in {0,1}),

data (n*m*(it+1) doubles)

Polynomials (type 2) and Size implicit indices (type 129)
row_size m (a long integer),

column_size n (a long integer),

real/complex flag it (long integer in {0,1}),

formal variable name (16 bytes),

index_table (m*n+1 long integers);

data ((N-1)*(it+1) doubles) , where N is the value of the last entry of the index_table

Booleans (type 4)
row_size m (a long integer),

column_size n (a long integer);

data (n*m long integers)

Floating sparse matrices (type 5)
row_size m (a long integer),

column_size n (a long integer),

real/complex_flag it (a long integer in {0,1}),

total_number_of_non_zero_elements nel (a long integer),

number_of_non_zero_elements_per_row (m long integers),

column_index_non_zero_elements (nel long integers),

non_zero_values (nel*(it+1) doubles)

Boolean sparse matrices (type 6)
row_size m (a long integer),

column_size n (a long integer),

unused it (a long integer),

total_number_of_non_zero_elements nel (a long integer),

number_of_non_zero_elements_per_row (m long integers),

column_index_non_zero_elements (nel long integers)

Matlab sparse matrix (type 7)
row_size m (a long integer),

column_size n (a long integer),

real/complex_flag it (a long integer in {0,1}),

total_number_of_non_zero_elements nel (a long integer),

number_of_non_zero_elements_per_column (n long integers),

row_index_non_zero_elements (nel long integers),

non_zero_values (nel*(it+1) doubles)

Integer matrices (type 8)
row_size m (a long integer),

column_size n (a long integer),

integer_type (a long integer): 1,2,4, or 11,12,14 for signed and unsigned 1,2,4 bytes integers;

data (n*m bytes for integer_type 1 or 11, n*m short integers for integer_type 2 or 12, n*m long integers for integer_type 4 or 14)

handles (type 9)
version (4 bytes)

row_size m (a byte),

column_size n (a byte),

data (m*n serialization_records)

A serialization_record is a flat representation of the C data structure associated with the corresponding graphic object. Each graphic object is defined by a (recursive) set of properties (see the get) function).

The saved serialization_record of a graphic object is structured as follow
type_length n (a byte)

type (n bytes, the ascii codes of the type name)

property_values record (variable length)

Strings (type 10)
row_size m (a long integer),

column_size n (a long integer),

index_table (n*m+1 long integers);

data (N long integers, the Scilab encoding of the characters (see code2str), where N is the value of the last entry of the index_table

Uncompiled functions (type 11)
nout (long integer),

lhs_names (6*nout long integers, see the Remarks section below),

nin (long integer),

rhs_names (6*nin long integers, see the Remarks section below);

code_length N (a long integer),

code (N long integers)

Compiled functions (type 13)
nout (long integer),

lhs_names (6*nout long integers, see the Remarks section below),

nin (long integer),

rhs_names (6*nin long integers, see the Remarks section below),

pseudo_code_length N (a long integer),

pseudo_code (N long integers)

Libraries (type 14)
path_length np (a long integer),

path_name (np long integers: the path character codes sequence, (see code2str)),

number of names nn (long integer),

names (6*nn long integers, see the Remarks section below);

Lists (type 15), tlists (type 16), mlists (type 17)
number of fields n (a long integer),

index (n+1 long integers);

variables_sequence (n variables, each one written according to its format)

Pointers (type 128)
Not handled

Function pointers (type 130)
function_ptr (a long integer,(see funptr))

function_name_code (6 long integers,see the Remarks section below);

Remarks

Numbers (long interger, short integers, double) are stored using the little endian convention.

The variable names are stored as a sequence of 6 long integers, with a specific encoding. see the cvname.f file for details.

See Also

save, load, listvarinfile, type, typeof

Authors

compiled by Enrico Segre

Name

scanf — Converts formatted input on standard input

```
[v_1,...v_n]=scanf (format);
```

Parameters

format
:Specifies the format conversion.

Description

The scanf functions get character data on standard input (%io(1)), interpret it according to a format, and returns the converted results.

The format parameter contains conversion specifications used to interpret the input.

The format parameter can contain white-space characters (blanks, tabs, newline, or formfeed) that, except in the following two cases, read the input up to the next nonwhite-space character. Unless there is a match in the control string, trailing white space (including a newline character) is not read.

- Any character except % (percent sign), which must match the next character of the input stream.
- A conversion specification that directs the conversion of the next input field. see scanf_conversion for details.

See Also

printf , read , fscanf , sscanf , scanf_conversion

Name

scanf_conversion — scanf, sscanf, fscanf conversion specifications

Description

Each conversion specification in the format parameter contains the following elements:

+
The character % (percent sign)

+
The optional assignment suppression character *

+
An optional numeric maximum field width

+
A conversion code

The conversion specification has the following syntax:

`[*][width][size]convcode`.

The results from the conversion are placed in v_i arguments unless you specify assignment suppression with * (asterisk). Assignment suppression provides a way to describe an input field that is to be skipped. The input field is a string of nonwhite-space characters. It extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion code indicates how to interpret the input field. You should not specify the v_i parameter for a suppressed field. You can use the following conversion codes:

%
:Accepts a single % (percent sign) input at this point; no assignment is done.

d, i
:Accepts a decimal integer;

u
:Accepts an unsigned decimal integer;

o
:Accepts an octal integer;

x
:Accepts a hexadecimal integer;

e,f,g
:Accepts a floating-point number. The next field is converted accordingly and stored through the corresponding parameter, which should be a pointer to a float. The input format for floating-point numbers is a string of digits, with the following optional characteristics:

+
It can be a signed value.

+
It can be an exponential value, containing a decimal point followed by an exponent field, which consists of an E or an e followed by an (optionally signed) integer.

+
It can be one of the special values INF, NaN,

s
:Accepts a string of characters.

c
:character value is expected. The normal skip over white space is suppressed.

%lg
: get value as a double.

See Also

scanf , scanf , fscanf

Functions

Nom

add_profiling — Adds profiling instructions to a function.

```
add_profiling(funname)
```

Parameters

funname
: A character string, the name of the function

Description

`add_profiling(funname)` Adds profiling instructions to the function named `funname`. Then when this function is run the number of calls, the time spent is stored for each function line.

Examples

```
function x=foo(a,n)
 x=0;
 for i=1:n
   if x<10 then
     x=x+a
   else
     x=x+1
   end
 end
 x=x^2+1
endfunction

add_profiling("foo")
foo(0.1,100) //run the function
profile(foo) //extract profile information
```

See Also

profile, plotprofile, remove_profiling, reset_profiling

Authors

Serge Steer, INRIA

Used Functions

This function uses the Scilab functions `bytecode` and `walkbytecode`

Nom

bytecode — given a function returns the "bytecode" of a function in a Scilab array and conversely.

```
x = bytecode(f)
f= bytecode(X)
```

Parameters

f
: A scilab function.

x
: an int32 row vector

Description

`x = bytecode(f)` returns the "bytecode" of the function `f` in the Scilab integer array `x`.

`f = bytecode(x)` returns in `f` the function associated with the "bytecode" given in the Scilab integer array `x`. Warning the validity of `x` is not checked.

Remark

The bytecode of Scilab function will evolve drastically in the future, So the use of this function should be restricted to the profiling instruction handling.

Examples

```
function a=foo(),a=sin(3),endfunction
bytecode(foo)
```

See Also

add_profiling , bytecodewalk , macr2lst , macr2tree

Authors

Serge Steer, INRIA

Nom

bytecodewalk — walk in function bytecode applying transformation.

```
c1 = bytecodewalk(code,query,job)
```

Parameters

code
: int32 vector: input byte code array

query
: integer, the opcode to look for

job
: the operation to perform, for the requested opcode

c1
: int32 vector: output byte code array

Description

walk in function bytecode applying transformation.

See Also

bytecode

Authors

Serge Steer INRIA

Name

fun2string — generates ascii definition of a scilab function

```
txt=fun2string(fun,name)
```

Parameters

fun
: a function type variable

name
: a character string, the generated function name

txt
: a column vector of strings, the text giving the scilab instructions

Description

Given a loaded Scilab function pseudo-code fun2string allows to re-generate the code. The generated code is indented and beautified.

The mechanism is similar, but simpler than the mfile2sci one. It may be adapted for syntax translations.

Examples

```
txt=fun2string(asinh,'foo');
write(%io(2),txt,'(a)')
```

See Also

getf , edit , macrovar

Name

function — opens a function definition
endfunction — closes a function definition

Description

```
function <lhs_arguments>=<function_name><rhs_arguments>
<statements>
endfunction
```

Where

<function_name>
stands for the name of the function

<rhs_arguments>
stands for the input argument list. It may be

- a comma separated sequence of variable names enclosed in parenthesis, like `(x1,...,xm)`. Last variable name can be the key word `varargin` (see varargin)
- the sequence `()` or nothing, if the function has no input argument.

<lhs_arguments>
stands for the output argument list. It may be

- a comma separated sequence of variable names enclosed in brackets, like `[y1,...,yn]`. Last variable name can be the key word `varargout` (see varargout)
- the sequence `[]` ,if the function has no input argument. In this case the syntax may also be: `function <function_name><rhs_arguments>`

<statements>
stands for a set of scilab instructions (statements) This syntax may be used to define function (see functions) inline or in a script file (see exec). For compatibility with old Scilab versions, functions defined in a script file containing only function definitions can be "loaded" into Scilab using the `getf` function.

The `function <lhs_arguments>=<function_name><rhs_arguments>` sequence cannot be split over several lines. This sequence can be followed by statements in the same line if a comma or a semicolon is added at its end.

function definitions can be nested

Examples

```
//inline definition (see function)
function [x,y]=myfct(a,b)
x=a+b
y=a-b
endfunction
```

```
[x,y]=myfct(3,2)

//a one line function definition
function y=sq(x),y=x^2,endfunction

sq(3)

//nested functions definition
function y=foo(x)
a=sin(x)
function y=sq(x), y=x^2,endfunction
y=sq(a)+1
endfunction

foo(%pi/3)


// definition in an script file (see exec)
exec SCI/modules/elementary_functions/macros/asinh.sci;
```

See Also

functions , exec , getf

Name

functions — Scilab procedures and Scilab objects

Description

Functions are Scilab procedures ("macro", "function" and "procedure" have the save meaning).

Function definition

Usually, they are defined in files with an editor and loaded into Scilab using the exec function or through a library (see lib or genlib). But They can also be defined on-line (see deff or function. A function is defined by two components:

- a "syntax definition" part as follows:

```
function [y1,...,yn]=foo(x1,...,xm)
function [y1,...,yn,varargout]=foo(x1,...,xm,varargin)
```

- a sequence of scilab instructions.

The "syntax definition" line gives the "full" calling syntax of this function. The `yi` are output variables calculated as functions of input variables `xi` and variables existing in Scilab when the function is executed.

Calling function

- Usually function calling syntax is `[y1,...,yn]=foo(x1,...,xm)`. Shorter input or output argument list than definition ones may be used. In such cases, only the first variables from the left are used or set.

  The argn function may be used to get the actual number of calling arguments.

- It is possible to define function with indeterminate number of input or output maximum number of arguments. This can be done using the varargin and varargout keywords. See the given links for details.

- It is also possible to use "named argument" to specify input arguments: suppose function `fun1` defined as `function  y1=fun1(x1,x2,x3)` then it can be called with a syntax like `y=fun1(x1=33,x3=[1 2 3])` within `fun1` x2 will be undefined,

  it can also be called with a syntax like `y=fun1(x1=33,y='foo')`. In such a case the `y` variable will be available in the context of the function `fun1`. Note that the maximum number of argument must be less or equal to the number of formal input argument used in the function syntax part.

  It is possible to check for defined variables with the exists function.

- When a function has no left hand side argument and is called only with character string arguments, the calling syntax may be simplified:

```
fun('a','toto','a string')
```

  is equivalent to:

```
fun a toto 'a string'
```

Miscellaneous

Functions are Scilab objects (with type numbers 13 or 11). They and can be manipulated (built, saved, loaded, passed as arguments,..) as other variable types.

Collections of functions can be collected in libraries. Functions which begin with % sign (e.g. `%foo`) are often used to overload (see overloading) operations or functions for new data type.

Examples

```
//inline definition (see function)
function [x,y]=myfct(a,b)
  x=a+b
  y=a-b
endfunction

[x,y]=myfct(3,2)

//inline definition (see deff)
deff('[x,y]=myfct(a,b)',['x=a+b';
                         'y=a-b'])
// definition in an ascii file (see exec)
exec SCI/modules/elementary_functions/macros/asinh.sci;
```

See Also

function , deff , getf , comp , lib , getd , genlib , exists , varargin , varargout

Name

genlib — build library from functions in given directory

```
genlib(lib_name [[,dir_name, [ Force [,verb [,Names]]]])
genlib(lib_name [,path=dir_name] [,verbose=verb] [,force=Force] [,names=Names])
```

Parameters

lib_name:
: Scilab string. The variable name of the library to (re)create.

dir_name:
: Scilab string. The name of the directory to look for `.sci`-files.

Force
: boolean value (default value is `%f`). Set it to `%t` to force the sci-files recompilation.

verb
: boolean values (default value is `%f`). Set it to `%t` to get information.

Names
: a vector of strings, the names of function to include in the library. By default all the sci-files are taken into account

Description

For each `.sci` file in `dir_name` (or only those specified by the `Names` argument), `genlib` executes a `getf` and saves the functions to the corresponding `.bin` file. The `.sci` file must not contain anything but Scilab functions. If a `.bin` file is newer than the associated `.sci` file, `genlib` does not translate and save the file.

This default behaviour can be changed if `force` is given and set to `%t`. In this latter case the recompilation is always performed for each `.sci` file.

When all `.sci` files have been processed, `genlib` creates a library variable named `lib_name` and saves it in the file `lib` in `dir_name`. If the Scilab variable `lib_name` is not protected (see predef) this variable is updated.

If `verbose` is et to `%t` information are displayed during the build process.

If `dir_name` argument is not given and if `lib_name` Scilab variable exists and it is a library dir_name is taken equal to the `lib_name` library path (update mode).

Restrictions

Scilab tacitly assumes that file `foo.sci` defines at least a function named `foo`. If subsidiary functions are included, they are made known to Scilab only after the function `foo` had been referenced.

See Also

getd , getf , save , lib

Name

get_function_path — get source file path of a library function

```
path=get_function_path(fun_name)
```

Parameters

fun_name
: a string, the name of the function

path
: a string, the absolute pathname of the function source file (.sci) or [].

Description

Given the name of a function get_function_path returns the absolute pathname of the function source file if the function is defined in a Scilab library (see lib) or [] if name does not match any library function.

Examples

```
get_function_path('median')
```

See Also

lib , string

Name

getd — getting all functions defined in a directory

```
getd(path)
```

Parameters

path
: Scilab string. The directory pathname

Description

loads all `.sci` files (containing Scilab functions) defined in the `path` directory.

Examples

```
getd('SCI/modules/cacsd/macros')
```

See Also

getf , lib , getcwd , pwd , chdir

Name

head_comments — display scilab function header comments

```
head_comments(name)
head_comments(name,%paths)
```

Parameters

name
: character string, the function name

%paths
: character string vector, paths where to look for the asscoated sci-file

Description

`comments(name)` displays the function header comments (like the Matlab help). The comments are read from the associated sci-file. If `name` if a function in a library the sci-file path is those given by the library path (see `lib`). If `name` if a function which is not in a library, a file with name `name.sci` is searched in the directories given by the variable `%paths`

Warning, most of the scilab predefined functions have no header comments.

Examples

```
head_comments sinc
```

See Also

help

Authors

Serge Steer, INRIA

Name

library — library datatype description

Description

A library is a data type with type number 14. It contains a path-name and a set of names. It allows automatic loading of variables using the following algorithm:

Suppose the Scilab user references the variable named `foo`. Scilab first looks if `foo` is the name of a primitive or of an already defined variable. If not, it looks for `foo` sequentially (the newest first) in all defined library .

Suppose `foo` belongs to the set of names of the library `xlib` then Scilab tries to load the file <xlib-path-name>/foo.bin. <xlib-path-name>/foo.bin must have been created using the `save` function

Library are often used for collection of functions, but they can also be used for any collection of scilab variables

If a function is defined in more than one library, the default search algorithm loads thode contained in the newest. It possible to force the use of a specific library using dot notation:

`xlib.foo` loads the variable `foo` contained in `xlib`. if `foo` is a function and `xlib.foo(args)` executes the functions

Examples

```
// elemlib is a predefined library
elementary_functionlib //displays the contents of the library
A=rand(3,3);
cosm(A) //loads cosm and executes it
whos -name cosm // now cosm is a variable
elementary_functionlib.sinm //loads sinm from the library
elementary_functionlib.cosm(A) //reloads cosm and executes it
```

See Also

lib , string , load , save

Name

listfunctions — properties of all functions in the workspace

```
[flist,compiled,profilable,called] = listfunctions([scope])
```

Parameters

scope
: string, "local" (default) or "global"

flist
: string array, names of all the function variables in the specified namespace

compiled
: boolean array, true if the corresponding element of flist is of type=13

profilable
: boolean array, true if the corresponding element of flist is of type=13, and additionally profiling information is found in the pseudocode of the function

called
: uint32 array, number of times the corresponding element of flist has been already called (nonzero only for profilable functions)

Description

- This function checks all the variables in the workspace (given by who) and collects those of type 11 or 13; for the latter, lst=macr2lst(fun) is called, in order to check for the magic profiling entry at the end of the first codeline, i.e. lst(5)(1)=="25".

Examples

```
recompilefunction("asinh","p")
[flist,compiled,profilable,called] = listfunctions();
flist(profilable)
```

See Also

function , getf , deff , comp , fun2string , profile , recompilefunction

Authors

Enrico Segre

Bibliography

http://wiki.scilab.org/Scilab_function_variables%3A_representation%2C_manipulation

Name

macro — Scilab procedure and Scilab object

Description

Macros are Scilab procedures ("macro", "function" and "procedure" have the save meaning). Usually, they are defined in files with an editor and loaded into Scilab by `getf` or through a library.

They can also be defined on-line (see `deff`). A file which contains a macro must begin as follows:

```
function [y1,...,yn]=foo(x1,...,xm)
```

The `yi` are output variables calculated as functions of input variables and variables existing in Scilab when the macro is executed. A macro can be compiled for faster execution. Collections of macros can be collected in libraries. Macros which begin with `%` sign (e.g. `%foo`) and whose arguments are lists are used to perform specific operations: for example, `z=%rmr(x,y)` is equivalent to `z=x*y` when `x` and `z` are rationals (i.e. `x=list('r',n,d,[])` with `n` and `d` polynomials).

See Also

deff , getf , comp , lib

Name

macrovar — variables of function

```
vars=macrovar(function)
```

Parameters

vars
: list `list(in,out,globals,called,locals)`

function
: name of a function

Description

Returns in a list the set of variables used by a function. `vars` is a list made of five column vectors of character strings

`in` input variables (`vars(1)`)

`out` output variables (`vars(2)`)

`globals` global variables (`vars(3)`)

`called` names of functions called (`vars(4)`)

`locals` local variables (`vars(5)`)

Examples

```
deff('y=f(x1,x2)','loc=1;y=a*x1+x2-loc')
vars=macrovar(f)
```

See Also

string , macr2lst

Name

plotprofile — extracts and displays execution profiles of a Scilab function

```
plotprofile(fun)
```

Parameters

fun
: a Scilab compiled function, or a function name (string), or an array of function names

Description

To use `plotprofile`, the Scilab function must have been prepared for profiling (see getf).

When such a function is executed, the system counts how many times each line is executed and how much cpu time is spent executing each line. This data is stored within the function data structure. The function `plotprofile` in an interactive command which displays this results in a graphic window. When a line is clicked, the source of the function is displayed with the selected line highlighted.

NOTE: you have to click on the "Exit" item in the graphics windows to exit from "plotprofile".

The function code is regenerated with `fun2string` and dumped into a temporary file.

Examples

```
//define a function and prepare it for profiling
deff('x=foo(n)',['if n==0 then'
                 '  x=[]'
                 'else'
                 '  x=0'
                 '  for k=1:n'
                 '    s=svd(rand(n+10,n+10))'
                 '    x=x+s(1)'
                 '  end'
                 'end'],'p')
//call the function
foo(30)
//get execution profiles
plotprofile(foo) // click on Exit to exit
```

See Also

profile , showprofile , fun2string

Name

profile — extract execution profiles of a Scilab function

```
c=profile(fun)
```

Parameters

fun
: a Scilab function

c
: a nx3 matrix containig the execution profiles

Description

To use `profile` the Scilab function must have been prepared for profiling (see getf).

For such function, When such a function is executed the systems counts how many time each line is executed and how may cpu time is spend for each line execution. These data are stored within the function data structure. The profile function allows to extract these data and return them in the two first columns of `c`. The `c` third column gives a measure of interpetor effort for one execution of corresponding line. Ith line of `c` corresponds to Ith line of the function (included first)

Note that, due to the precision of the processor clock (typically one micro second), some executed lignes may appear with 0 cpu time even if total cpu time really spend in their execution is large.

Examples

```
//define function and prepare it for profiling
deff('x=foo(n)',['if n==0 then'
                 '  x=[]'
                 'else'
                 '  x=0'
                 '  for k=1:n'
                 '    s=svd(rand(n+10,n+10))'
                 '    x=x+s(1)'
                 '  end'
                 'end'],'p')
//call the function
foo(10)
//get execution profiles
profile(foo)
//call the function
foo(20)
profile(foo) //execution profiles are cumulated
```

See Also

getf , deff , plotprofile , showprofile

Name

recompilefunction — recompiles a scilab function, changing its type

```
recompilefunction(funname [,kind [,force]])
```

Parameters

funname
: string, name of the function to recompile

kind
: string: **"n"** (noncompiled, type 11), **"c"** (compiled, type 13) or **"p"** (compiled, type 13, with provision for profiling). Default "c".

force
: boolean. If false, the function is recomplied only if its kind changes; if true, it is recompiled even if it keeps the same kind (notably useful to recompile a "p" function, to reset the profiling statistics).

Description

- This function reverse-compiles a function variable via fun2string, and recompiles it to the desired kind with deff.

Examples

```
recompilefunction("asinh","p")
for i=1:100; asinh(rand(100,100)); end
showprofile(asinh)
```

See Also

function , getf , deff , comp , fun2string , profile

Authors

Enrico Segre

Bibliography

http://wiki.scilab.org/Scilab_function_variables%3A_representation%2C_manipulation

Nom

remove_profiling — Removes profiling instructions toout of a function.

```
remove_profiling(funname)
```

Parameters

funname
: A character string, the name of the function

Description

`remove_profiling(funname)` Removes profiling instructions (if any) out of the function named `funname`.

Examples

```
function x=foo(a,n)
 x=0;
 for i=1:n
   if x<10 then
     x=x+a
   else
     x=x+1
   end
 end
 x=x^2+1
endfunction

add_profiling("foo")
foo(0.1,100) //run the function
profile(foo) //extract profile information
remove_profiling("foo")
```

See Also

profile, plotprofile, remove_profiling, reset_profiling

Authors

Serge Steer, INRIA

Used Functions

This function uses the Scilab functions `bytecode` and `walkbytecode`

Nom

reset_profiling — Resets profiling counters of a function.

```
reset_profiling(funname)
```

Parameters

funname
: A character string, the name of the function

Description

`reset_profiling(funname)` Resets profiling counters (if any) of the function named `funname`.

Examples

```
function x=foo(a,n)
 x=0;
 for i=1:n
   if x<10 then
     x=x+a
   else
     x=x+1
   end
 end
 x=x^2+1
endfunction

add_profiling("foo")
foo(0.1,100) //run the function
profile(foo) //extract profile information
reset_profiling("foo")
profile(foo) //extract profile information
```

See Also

profile, plotprofile, add_profiling, reset_profiling, remove_profiling

Authors

Serge Steer, INRIA

Used Functions

This function uses the Scilab functions `bytecode` and `walkbytecode`

Name

showprofile — extracts and displays execution profiles of a Scilab function

```
showprofile(fun)
```

Parameters

fun
: a Scilab function

Description

To use `showprofile` the Scilab function must have been prepared for profiling (see getf).

For such function, When such a function is executed the systems counts how many time each line is executed and how may cpu time is spend for each line execution. These data are stored within the function data structure. The `showprofile` function outputs profiling results (see `profile`) with text of the function lines.

Function text is rebuild with `fun2string`.

Examples

```
//define function and prepare it for profiling
deff('x=foo(n)',['if n==0 then'
                 '  x=[]'
                 'else'
                 '  x=0'
                 '  for k=1:n'
                 '    s=svd(rand(n+10,n+10))'
                 '    x=x+s(1)'
                 '  end'
                 'end'],'p')
//call the function
foo(30)
//get execution profiles
showprofile(foo)
```

See Also

profile , plotprofile , fun2string

Name

varargin — variable numbers of arguments in an input argument list

Description

A function whose last input argument is `varargin` can be called with more input arguments than indicated in the input argument list. The calling arguments passed form `varargin` keyword onwards may then be retrieved within the function in a list named `varargin`.

Suppose that `varargin` keyword is the `n` th argument of the formal input argument list, then if the function is called with less than `n-1` input arguments the `varargin` list is not defined, if the function is called with `n-1` arguments then `varargin` list is an empty list.

`y = function ex(varargin)` may be called with any number of input arguments. Within function `ex` input arguments may be retrieved in `varargin(i)` ,`i=1:length(varargin)`

If it is not the last input argument of a function, `varargin` is a normal input argument name with no special meaning.

The total number of actual input arguments is given by argn(2).

Remark

Named argument syntax like `foo(...,key=value)` is incompatible with the use of varargin. The reason is that the names (i.e. keys) associated with values are not stored in the varargin list. Consider for instance:

function foo(varargin),disp([varargin(1),varargin(2)]),endfunction

foo(a=1,b=2)

Scilab answers: 1. 2.

foo(b=1,a=2)

Scilab answers: 1. 2.

Result is the same, but the arguments were inverted.

Examples

```
deff('exampl(a,varargin)',['[lhs,rhs]=argn(0)'
                           'if rhs>=1 then disp(varargin),end'])
exampl(1)
exampl()
exampl(1,2,3)
l=list('a',%s,%t);
exampl(1,l(2:3))
```

See Also

function, varargout, list

Name

varargout — variable numbers of arguments in an output argument list

Description

A function whose output argument list contains `varargout` must be called with more output arguments than indicated in the output argument list. The calling arguments passed form `varargout` keyword onwards are extracted out of the `varargout` list defined in the function.

`varargout= function ex()` may be called with any number of output arguments. Within function `ex` output arguments may be stored in in `varargout(i)`.

`[X1,...Xn,varargout]= function ex()` may also be used. In this case the `Xi` variables must be assigned in the function as well as `varargout(i)`.

The actual total number of output argument is given argn(1)

Remark

The `varargout` variable must be created within the function and assigned to a list. If varargout is the only formal output variable the list must contain at least one entry.

Examples

```
function varargout=exampl()
   varargout=list(1,2,3,4)
endfunction

x=exampl()
[x,y]=exampl()
[x,y,z]=exampl()


function [a,b,varargout]=exampl1()
   a='first'
   b='second'
   varargout=list(1,2,3,4)
endfunction
exampl1()
[a,b]=exampl1()
[a,b,c]=exampl1()
```

See Also

function, varargin, list

GUI

Name

about — show "about scilab" dialog box

```
about()
```

Description

show "about scilab" dialog box.

Examples

```
about()
```

Authors

Allan CORNET

Name

addmenu — interactive button or menu definition

```
addmenu(button [,submenus] [,action])
addmenu(gwin,button [,submenus] [,action])
```

Parameters

button
: a character string. The button name. An & can be placed before the character in the name to be used for keyboard shortcut; this character will be underlined on the GUI. Under MacOSX, a sub-menu with the same name is automaticaly added (no button can be added to the menu bar).

submenus
: a vector of character string. The sub_menus items names

action
: a list with 2 elements action=list(flag,proc_name)

  flag
  : an integer (default value is 0)

  flag==0
  : the action is defined by a scilab instruction

  flag==1
  : the action is defined by a C or Fortran procedure

  flag==2
  : the action is defined by a scilab function

  proc_name
  : a character string which gives the name of scilab variable containing the instruction or the name of procedure to call.

gwin
: integer. The number of graphic window where the button is required to be installed

Description

The function allows the user to add new buttons or menus in the main window or graphics windows command panels.

If
: `action` argument is not given the action associated with a button must be defined by a scilab instruction given by the character string variable which name is

  + `button` for a main window command

  + `button_gwin` for a graphic window command

If
: `action` argument is set to 0 `proc_name` should be the name of a Scilab string vector. Actions associated with the kth sub_menu must be defined by scilab instructions stored in the kth element of the character string variable.

If
: `action` argument is set to 1 `proc_name` designes a C or Fortran procedure, this procedure may be interfaced in Fortran subroutine default/fbutn.f or dynamically linked with scilab using the `link` function. The C calling sequence is: `(char* button_name, int* gwin,int *k)`

If

`action` argument is set to 2 `proc_name` designes a Scilab function. This function calling sequence should be:

+ `proc_name(k)` for a main window command

+ `proc_name(k,gwin)` for a graphic window command or a main window command

Examples

```
addmenu('foo')
foo='disp(''hello'')'

addmenu('Hello',['Franck';'Peter'])
Hello=['disp(''hello Franck'')';'disp(''hello Peter'')']

addmenu(0,'Hello',['Franck';'Peter'])
Hello_0=['disp(''hello Franck'')';'disp(''hello Peter'')']

addmenu('Bye',list(0,'French_Bye'))
French_Bye='disp(''Au revoir'')'

//C defined Callback
// creating Callback code
code=[ '#include ""machine.h""'
 'void foo(char *name,int *win,int *entry)'
 '{'
 '  if (*win==-1) '
 '    sciprint(""menu %s(%i) in Scilab window selected\r\n"",name,*entry+1);'
 '  else'
 '    sciprint(""menu %s(%i) in window %i selected\r\n"",name,*entry+1,*win);'
 '}'];
//creating foo.c file
dir=getcwd(); chdir(TMPDIR)
mputl(code,TMPDIR+'/foo.c');
//reating Makefile
ilib_for_link('foo','foo.o',[],'c');
exec('loader.sce');
chdir(dir);
//add menu
addmenu('foo',['a','b','c'],list(1,'foo'))
```

See Also

setmenu , unsetmenu , delmenu

Name

clipboard — Copy and paste strings to and from the system clipboard.

```
clipboard("copy",data)
str=clipboard("paste")
clipboard("do","paste")
clipboard("do","copy")
clipboard("do","empty")
clipboard(winnum,"EMF")
clipboard(winnum,"DIB")
```

Parameters

data
: Scilab variable or data to set as the clipboard contents.

str
: The clipboard contents returned as a Scilab character string.

winnum
: Number of the graphic window to set as the clipboard contents.

Description

`clipboard("copy",data)` sets the clipboard contents to data. If data is not a character array, the clipboard uses `sci2exp` to convert it to a string.

`str = clipboard("paste")` returns the current contents of the clipboard as a string or as an empty string (""), if the current clipboard contents cannot be converted to a string.

`clipboard("do","paste")`, `clipboard("do","copy")`, `clipboard("do","empty")` performs a paste, copy or empty clipboard.

`clipboard(winnum,"EMF")` copy a graphic window identified by his window's number in the clipboard to EMF format.

`clipboard(winnum,"DIB")` copy a graphic window identified by his window's number in the clipboard to DIB format.

Note that `clipboard` function works only when Scilab used in window mode.

Authors

A.C.

Name

close — close a figure

Parameters

h

integer Handle of the window to close.

Description

This routine close a tksci figure (toplevel window). If a handle is given, the figure corresponding to this handle is closed. Otherwise, the current (active) figure is closed.

Examples

```
h=figure();
// creates  figure number 1.
uicontrol( h, 'style','text', ...
 'string','scilab is great', ...
 'position',[50 70 100 100], ...
 'fontsize',15);
// put a clever text in figure 1
figure();
// create figure 2
uicontrol( 'style','text', ...
 'string','Really great', 'position',[50 70 100 100], 'fontsize',15);
// put a text in figure 2
close();
// close the current graphic window (ie fig. 2)
close(h);
// close figure 1
```

See Also

figure , gcf

Authors

Bertrand Guiheneuf

Name

delmenu — interactive button or menu deletion

```
delmenu(button)
delmenu(gwin,button)
```

Parameters

button
: a character string. The button name. On Windows operating systems (not X_window), an & should be placed before the character in the name used for keyboard shortcut; this character is underlined on the GUI.

gwin
: integer. The number of graphic window where the button is required to be installed

Description

The function allows the user to delete buttons or menus create by `addmenu` in the main or graphics windows command panels. Predefined buttons on Scilab graphic windows can also be deleted.

If possible, it is better to delete first the latest created button for a given window to avoid gaps in command panels.

Examples

```
addmenu('foo')
delmenu('foo')
```

See Also

setmenu , unsetmenu , addmenu

Name

exportUI — Call the file export graphical interface

```
exportUI(figId)
exportUI(fig)
```

Parameters

figId
: integer, Id of the figure to export.

fig
: Figure handle, handle of the figure to export.

Description

`exportUI` routine call the graphical interface dedicated in exporting a graphic window into an image file.

See Also

xs2jpg , xs2eps , xs2png , xs2svg , xs2pdf

Authors

Jean-Baptiste Silvy

Name

figure — create a figure

```
f = figure(num);
f = figure("PropertyName1", Propertyvalue1, ..., ..., "PropertyNameN", Prop
```

Description

This routine creates a figure. If an ID is given, the figure corresponding to this ID is created. Otherwise, the window is created with the first free ID, that is the lowest integer not already used by a window.

Parameters

num
: ID of the window to create. If not specified, the first free ID is used.

PropertyName{1, ..., N}
: character string name of a property to set. One of the property names listed below.

PropertyValue{1, ..., N}
: scilab object value to give to the corresponding property.

f
: handle of the newly created window.

Properties

BackgroundColor
: [1,3] real vector or string Background color of the figure. A color is specified as Red, Green and Blue values. Those values are real in [0,1]. The color can be given as a real vector, ie [R,G,B] or a string where each value is separated by a "|", ie "R|G|B"

Figure_name
: character string, allows to set the title of the figure.

ForegroundColor
: [1,3] real vector or string Foreground color of the figure. A color is specified as Red, Green and Blue values. Those values are real in [0,1]. The color can be given as a real vector, ie [R,G,B] or a string where each value is separated by a "|", ie "R|G|B"

Position
: allows to control the geometrical aspect of the control. It is a [1,4] real vector [x y width height] where the letters stand for the x location of the left bottom corner, the y location of the left bottom corner, the width and the height of the uicontrol. One can also set this property by giving a string where the fields are separated by a "|", ie "x|y|width|height".

Tag
: string this property is generally used to identify the figure. It allows to give it a "name". Mainly used in conjontion with findobj().

Userdata
: this can be used to associate some Scilab objects to a fugure.

Examples

```
// Create figure having figure_id==3
h=figure(3);
// Add a text uicontrol in figure 3
uicontrol(h, "style", "text", ...
             "string", "This is a figure", ...
             "position", [50 70 100 100], ...
             "fontsize",15);


// Create figure having figure_id==1
figure();
// Add a text uicontrol in figure 1
uicontrol("style", "text", ...
          "string", "Another figure", ...
          "position", [50 70 100 100], ...
          "fontsize", 15);

// Close current figure (ie figure 1)
close();
// close figure 3
close(h);
```

See Also

close , gcf

Authors

Bertrand Guiheneuf

V.C.

Name

findobj — find an object with specified property

```
h =  findobj(propertyName, propertyValue)
```

Parameters

propertyName
: string character Name of the property to test (case unsensitive).

propertyValue
: string character specify the value the tested propoerty should be equal to (case sensitive).

h
: handle of the found object.

Description

This routine is currently used to find objects knowing their 'tag' property. It returns handle of the first found object which property *propertyName* is equal to *propertyValue*. If such an object does not exist, the function returns an empty matrix.

Examples

```
// Create a figure
h=figure();
// Put a text in the figure
uicontrol(h, "style","text", ...
 "string","This is a figure", ...
 "position",[50 70 100 100], ...
 "fontsize",15, ...
 "tag","Alabel");
// Find the object which "tag" value is "Alabel"
lab=findobj("tag","Alabel");
disp("The text of the label is """+lab.string+""""");
// Close the figure
close();
```

See Also

uicontrol , uimenu , set , get

Authors

Bertrand Guiheneuf

V.C.

Name

gcbo — Handle of the object whose callback is executing.

```
gcbo
```

Description

`gcbo` is a Scilab variable automatically created each time a callback in executed. This variable is initialised using getcallbackobject.

`gcbo` does not exists in Scilab environment if no callback is currently executed.

You can use `gcbo` in callback functions particularly if you write a single callback function for multiple objects, it helps you to know which object received a user action.

See Also

getcallbackobject

Authors

Vincent COUVERT

Name

getcallbackobject — Return the handle of the object whose callback is executing.

```
h = getcallbackobject()
```

Parameters

h
: Handle: the handle of the object whose callback is executing.

Description

`getcallbackobject` is used to automatically create Scilab variable called gcbo each time a callback in executed.

`getcallbackobject` returns [] if no callback is currently executed.

See Also

gcbo

Authors

Vincent COUVERT

Name

getinstalledlookandfeels — returns a string matrix with all Look and Feels.

```
lnf=getinstalledlookandfeels()
```

Parameters

lnf
: a string matrix.

Description

returns a string matrix with all Look and Feels that you can use.

Examples

```
getinstalledlookandfeels()
```

See Also

setlookandfeel , getlookandfeel

Authors

Allan CORNET

Name

getlookandfeel — gets the current default look and feel.

```
lnf=getlookandfeel()
```

Parameters

lnf
: a string with current look and feel.

bok
: a boolean.

Description

Gets the current default look and feel.

Examples

```
currentlnf = getlookandfeel();

// Look and feel CDE/Motif
setlookandfeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel")

sleep(3000);

// Look and feel métal
setlookandfeel("javax.swing.plaf.metal.MetalLookAndFeel")

sleep(3000);

setlookandfeel(currentlnf)
```

See Also

getinstalledlookandfeels , setlookandfeel

Authors

Allan CORNET

Name

getvalue — xwindow dialog for data acquisition

```
[ok,x1,..,x14]=getvalue(desc,labels,typ,ini)
```

Parameters

desc
: column vector of strings, dialog general comment

labels
: n column vector of strings, `labels(i)` is the label of the ith required value

typ
: `list(typ_1,dim_1,..,typ_n,dim_n)`

typ_i
: defines the type of the ith value, may have the following values:

"mat"
: for constant matrix

"col"
: for constant column vector

"row"
: for constant row vector

"vec"
: for constant vector

"str"
: for string

"lis"
: for list

dim_i
: defines the size of the ith value it must be a integer or a 2-vector of integer, -1 stands for undefined dimension

ini
: n column vector of strings, `ini(i)` gives the suggested response for the ith required value

ok
: boolean ,%t if ok button pressed, %f if cancel button pressed

xi
: contains the ith value if ok=%t. If left hand side has one more `xi` than required values the last `xi` contains the vector of answered strings.

Description

This function encapsulate `x_mdialog` function with error checking, evaluation of numerical response, ...

Remarks

All valid expressions can be used as answers; for matrices and vectors `getvalues` automatically adds [ ] around the given answer before numeric evaluation.

Examples

```
labels=["magnitude";"frequency";"phase    "];
[ok,mag,freq,ph]=getvalue("define sine signal",labels,...
     list("vec",1,"vec",1,"vec",1),["0.85";"10^2";"%pi/3"])
```

See Also

x_mdialog , x_matrix , x_dialog

Authors

S. Steer ; ;

Name

messagebox — Open a message box.

```
[btn] = messagebox(msg)
[btn] = messagebox(msg, msgboxtitle)
[btn] = messagebox(msg, msgboxtitle, msgboxicon)
[btn] = messagebox(msg, msgboxtitle, msgboxicon)
[btn] = messagebox(msg, msgboxtitle, msgboxicon, buttons)
[btn] = messagebox(msg, msgboxtitle, msgboxicon, buttons, ismodal)
```

Parameters

msg
: Matrix of strings: the message box displays each entry of this matrix (one entry per line).

msgboxtitle
: String: the title of the message box (default value is "Scilab Message").

msgboxicon
: String: the name of the icon to be displayed in the message box, its possible values are:

  - "error"
  - "hourglass"
  - "info"
  - "passwd"
  - "question"
  - "warning"
  - "scilab": default icon

buttons
: 1xn vector of strings: the names of the buttons to be displayed in the message box. By default, only one button is displayed with label "OK".

modal
: String: "modal" to create a modal dialog, any other string to create a non-modal dialog. Please note that "modal" can replace any of the other input arguments except msg (See examples).

btn
: Scalar: number of the button that the user pressed (1 is the leftmost button) for a modal dialog, 0 else.

Description

Creates a dialog window to display a message waiting or not for a user action. This function is used by x_message.

Examples

```
// Simple example
messagebox("Single line message")

// Multi line message with title
messagebox(["Multi-line" "message"], "User defined title")

// Icon specified by th euser
messagebox("An error message", "Error", "error")

// Buttons labels + "modal" replaces title
messagebox("Have you seen this beautiful message", "modal", "info", ["Yes" "No"

// "modal" given as fifth input argument
messagebox("An error message", "Error", "error", ["Continue" "Stop"], "modal")
```

See Also

x_message

Authors

Vincent COUVERT

Name

printfigure — Opens a printing dialog and prints a figure.

```
printfigure(figid)
status = printfigure(figid)
```

Parameters

figid
: Real: the id of the figure to be printed.

status
: Boolean: *%T* if the printing succeeds, *%F* otherwise.

Description

This function opens a dialog to select a printer, printing options... and then prints the figure.

Examples

```
plot2d();
printfigure(get(gcf(), "figure_id"));
```

See Also

toprint , printsetupbox

Authors

V.C.

Name

printsetupbox — Display print dialog box.

```
printsetupbox()
status=printsetupbox()
```

Parameters

status
: Boolean: *%T* if the user clicked on the OK button, *%F* otherwise.

Description

Displays the built-in printing dialogbox and configure the printer.

See Also

toprint , printfigure

Authors

A.C

Name

progressionbar — Draw a progression bar

```
winId=progressionbar(mes)
progressionbar(winId[,mes])
```

Parameters

mes
: string, message to display.

winId
: integer greater than 0, window identificator.

Description

`progressionbar(mes)` create a new progression bar, return window identificator.

`progressionbar(winId[,mes])` update the progression bar identificated as winId.

Examples

```
winId=progressionbar('Do something');
realtimeinit(0.3);
for j=0:0.1:1,
    realtime(3*j);
    progressionbar(winId);
end
winclose(winId);
```

Authors

Jaime Urzua

Name

root_properties — description of the root object properties.

Description

The root object is a virtual object used to get the computer screen properties. Use `get` function with *0* as first argument to access its properties.

Root properties

screensize_px:
: The screen size in pixels.

screensize_pt:
: The screen size in points.

screensize_mm:
: The screen size in millimeters.

screensize_cm:
: The screen size in centimeters.

screensize_in:
: The screen size in inches.

screensize_norm:
: The normalized screen size.

screendepth:
: The number of bits used to encode colors.

Examples

```
get(0, "screensize_px")
get(0, "screendepth")
```

See Also

get

Author

Vincent COUVERT

Name

setlookandfeel — sets the current default look and feel.

```
bok=setlookandfeel()
bok=setlookandfeel(lnf)
```

Parameters

lnf
: a string with a look and feel.

bok
: a boolean.

Description

Sets the current default Look and Feel.

setlookandfeel() without parameter set system default look and feel.

Examples

```
currentlnf = getlookandfeel();

// Look and feel Windows Classic
setlookandfeel("com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel")

// Look and feel Windows
setlookandfeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel")

sleep(3000);

// Look and feel CDE/Motif
setlookandfeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel")

sleep(3000);

// Look and feel GTK+
setlookandfeel("com.sun.java.swing.plaf.gtk.GTKLookAndFeel")

sleep(3000);

// Look and feel métal
setlookandfeel("javax.swing.plaf.metal.MetalLookAndFeel")

sleep(3000);

// Look and feel Macintosh
setlookandfeel("it.unitn.ing.swing.plaf.macos.MacOSLookAndFeel")

// System default look and feel

setlookandfeel()

sleep(3000);
```

```
// restore previous look and feel
setlookandfeel(currentlnf)
```

See Also

getinstalledlookandfeels , getlookandfeel

Authors

Allan CORNET

Name

setmenu — interactive button or menu activation

```
setmenu(button [,nsub])
setmenu(gwin,button [,nsub])
```

Parameters

button
: a character string. The button name

gwin
: integer. The number of graphic window where the button is installed

nsub
: integer. The number of submenu to de-activate (if any). If button has no sub-menu, `nsub` is ignored

Description

The function allows the user to make active buttons or menus created by `addmenu` in the main or graphics windows command panels.

Examples

```
addmenu('foo')   //New button made in main scilab window
unsetmenu('foo')   //button foo cannot be activated (grey string)
setmenu('foo')     //button foo can be activated (black string)
```

See Also

delmenu , unsetmenu , addmenu

Name

toolbar — show or hide a toolbar

```
state1=toolbar(winnum,state2)
state1=toolbar(winnum)
```

Parameters

state1
: returns toolbar's state 'on' or 'off'

winum
: window's number (-1: Scilab console window)

state2
: 'on' or 'off' set toolbar's state

Description

show or hide a toolbar.

Examples

```
toolbar(-1,'off')
state=toolbar(-1,'on')

plot3d();
h=gcf();
toolbar(h.figure_id,'off')
```

Authors

Allan CORNET
Vincent COUVERT

Name

toprint — Send text or figure to the printer.

```
toprint(filename)
toprint(linestoprint,pageheader)
toprint(figid)
toprint(figid,output)
status = toprint(filename)
status = toprint(linestoprint,pageheader)
status = toprint(figid)
status = toprint(figid,output)
```

Parameters

filename
: String: path of the text file to be printed.

linestoprint
: String matrix: text to be printed, each entry is a line in printed pages.

pageheader
: String: header of printed pages.

figid
: Real: the id of the figure to be printed.

output
: String: printing output type, must be *"pos"* for PostScript or *"gdi"* for Bitmap format (*"gdi"* by default).

status
: Boolean: *%T* if the printing succeeds, *%F* otherwise.

Description

Prints a text file, Scilab character strings or figure.

Examples

```
toprint(SCI+"/etc/scilab.start");
toprint(['Test','toprint primitive'],'Scilab page header');
scf(4);
plot();
toprint(4);
toprint(4,"pos");
```

See Also

printfigure , printsetupbox

Authors

A.C.

V.C.

Name

uicontrol — create a Graphic User Interface object

```
h = uicontrol(PropertyName,PropertyValue,...)
h = uicontrol(parent,PropertyName,PropertyValue,...)
h = uicontrol(uich)
```

Description

This routine creates an object in a figure.

If the handle of the figure is given (as the first parameter), the uicontrol is created in this figure. If no handle is given, the uicontrol is created in the current figure (which may be obtained with a call to `gcf()`). If there is no current figure, then one is created before the creation of the uicontrol.

Then when the control is created, the properties given as parameters are set with the corresponding values. It is equivalent to create the uicontrol, and then set its properties with the `set()` command. Nevertheless, it generally more efficient to set the properties in the call to `uicontrol()`. This is particularly true concerning the "Style" property. Indeed, the default value for this property is "Pushbutton". So if you do not set it at creation time, a button will be created, and will be transformed to another uicontrol when you call the `set(h, "Style", ... )` instruction. Scilab and all the graphic objects communicate through the property mechanism. Thus, to create adapted uicontrol, one has to know the use of the property fields.

`h = uicontrol(PropertyName, PropertyValue,...)` creates an uicontrol and assigns the specified properties and values to it. It assigns the default values to any properties you do not specify. The default uicontrol style is a "Pushbutton". The default parent is the current figure. See the Properties section for information about these and other properties.

`h = uicontrol(parent, PropertyName, PropertyValue,...)` creates a uicontrol in the object specified by the handle, parent. If you also specify a different value for the Parent property, the value of the Parent property takes precedence. parent is the handle of a figure.

`h = uicontrol(uich)` gives focus to the uicontrol specified by uich.

Properties

BackgroundColor
: [1,3] real vector or string

  Background color of the uicontrol. A color is specified as Red, Green and Blue values. Those values are real in [0,1]. The color can be given as a real vector, ie [R,G,B] or a string where each value is separated by a "|", ie "R|G|B".

Callback
: String

  Instruction evaluated by the Scilab interpreter when an uicontrol is activated. (for example when you click on a button).

Enable
: {on} | off

  Enable or disable the uicontrol. If this property is set to `"on"` (default), the uicontrol is operational, but if this property is set to `"off"`, the uicontrol will not respond to the mouse actions and will be grayed out.

FontAngle
{normal} | italic | oblique

For a control containing some text, this property sets the slant of the font.

FontSize
Scalar

For a control containing some text, this property sets the size of the font in FontUnits.

FontUnits
{points} | pixels | normalized

For a control containing some text, this property sets the units with which the FontSize is specified.

FontWeight
light | {normal} | demi | bold

For a control containing some text, this property sets the weight of the used font.

FontName
String

Used to choose the name of the font selected to display the text of the control.

ForegroundColor
[1,3] real vector or string

Foreground color of the uicontrol. A color is specified as Red, Green and Blue values. Those values are real in [0,1]. The color can be given as a real vector, ie [R,G,B] or a string where each value is separated by a "|", ie "R|G|B".

HorizontAlalignment
left | {center} | right

Set text horizontal alignment in the uicontrol. This property has only effect with Text, Edit and Check Boxes.

ListboxTop
Scalar

For a ListBox, this property tells which item of the list appears on the first line of the visible area of the list.

Max
Scalar

Specifies the largest value the "Value" property can be set to. It has however different meaning on each uicontrol:

- CheckBoxes: Max is the value the "Value" property take when control is checked.
- Sliders: Maximum value of the slider.
- ListBoxes: if (Max-Min)>1 the list allows multiple selection, Otherwise not.

Min
Scalar

Specifies the lowest value the "Value" property can be set to. It has however different meaning on each uicontrol:

- CheckBoxes: Min is the value the "Value" property take when control is unchecked.
- Sliders: Minimum value of the slider.
- ListBoxes: if (Max-Min)>1 the list allows multiple selection, Otherwise not.

Parent
Handle

Handle of the uicontrol parent. Changing this property allows to move a control from a figure to another.

Path
This property is no more supported.

Position
[1,4] real vector or string.

This property is used to set or get the geometrical configuration of a control. It is a vector [x y w h] where the letters stand for the x location of the left bottom corner, the y location of the left bottom corner, the width and the height of the uicontrol or a character string where each value is separated by a "|", ie "x|y|w|h". The units are determined by the "Units" property.

The width and height values determine the orientation of sliders. If width is greater than height, then the slider is oriented horizontally, otherwise the slider is oriented vertically.

Relief
flat | groove | raised | ridge | solid | sunken

Appearance of the border of the uicontrol:

- PushButtons: the default value for "Relief" property is "raised".
- Edits: the default value for "Relief" property is "sunken".
- Other styles: the default value for "Relief" property is "flat".

SliderStep
[1,2] real vector

[small big], the small step represents the movement achieved when clicking on the slider trough or tapping on the keyboard arrows (when the slider has focus); the big step is the amount moved when using Ctrl-keyboard-arrows. If the big step is omitted, it is defaulted to 1/10 of the scale.

String
String.

This property represents the text appearing in a uicontrol (Except for Frame and Slider styles). For ListBoxes and PopupMenus, the value can be a vector of string or a string where the items are separated by a "|". For Text uicontrols, this string can contain HTML code to format the text.

Style
{pushbutton} | radiobutton | checkbox | edit | text | slider | frame | listbox | popupmenu

Style of the uicontrol. Here is a short description of each one:

- Pushbutton: a rectangular button generally used to run a callback.
- Radiobutton: a button with to states. RadioButtons are intended to be mutually exclusive (Your code must implement mutually exclusive behavior).
- Checkbox: a button with to states (Used for multiple independent choices).

- Edit: an editable string zone.
- Text: a text control (generally static).
- Slider: a scale control, that is a scrollbar use to set values between in range with the mouse.
- Frame: a control representing a zone used to group related controls.
- Listbox: a control representing a list of items that can be scrolled. The items can be selected with the mouse.
- Popupmenu: a button which make a menu appear when clicked.

Tag
String

This property is generally used to identify the control. It allows to give it a "name". Mainly used in conjunction with `findobj()`.

Units
{points} | pixels | normalized

Set the units used to specify the "Position" property.

Userdata
Scilab data

This can be used to associate some Scilab objects (string,string matrix, matrix mxn) to an uicontrol.

Value
Scalar or vector

Value of the uicontrol. The exact meaning depends on the style of the uicontrol:

- CheckBoxes, Radio buttons: value is set to Max (see above) when on and Min when off.
- ListBoxes, PopupMenus: value is a vector of indexes corresponding to the indexes of the selected entries in the list. 1 is the first item of the list.
- Sliders: value indicated by the slider bar.

Verticalalignment
top | {middle} | bottom

Set text vertical alignment in the uicontrol. This property has only effect with Text and CheckBoxes styles.

Visible
{on} | off

Set the visibility of the uicontrol. If this property is set to `"on"` (default), the uicontrol is visible, but if this property is set to `"off"`, the uicontrol will not appear in its parent figure.

Examples

```
f=figure();
// create a figure
h=uicontrol(f,'style','listbox', ...
 'position', [10 10 150 160]);
```

```
// create a listbox
set(h, 'string', "item 1|item 2|item3");
// fill the list
set(h, 'value', [1 3]);
// select item 1 and 3 in the list
close(f);
// close the figure
```

See Also

figure, set, get, uimenu

Authors

Bertrand Guiheneuf

Vincent Couvert

Name

uigetcolor — Opens a dialog for selecting a color.

```
uigetcolor()
RGB = uigetcolor([title])
RGB = uigetcolor([title,] defaultRGB)
RGB = uigetcolor([title,] defaultRed, defaultGreen, defaultBlue)
[R, G, B] = uigetcolor([title])
[R, G, B] = uigetcolor([title,] defaultRGB)
[R, G, B] = uigetcolor([title,] defaultRed, defaultGreen, defaultBlue)
```

Parameters

title
: String: Optional argument, the title to display in the dialog. Default value is "Color Chooser".

defaultRGB
: 1x3 vector: the default values for Red, Green and Blue values given as a vector [red, green, blue].

defaultRed
: Scalar: the default value for red.

defaultGreen
: Scalar: the default value for green.

defaultBlue
: Scalar: the default value for blue.

RGB
: 1x3 vector: the values for Red, Green and Blue values given as a vector [red, green, blue] or [] if the user cancels.

R
: Scalar: the value for red or [] if the user cancels.

G
: Scalar: the value for green or [] if the user cancels.

B
: Scalar: the value for blue or [] if the user cancels.

Description

Creates a dialog window for selecting a color. All (default and returned) values must be in the interval [0 255].

Examples

```
uigetcolor()
[R, G, B] = uigetcolor([255 128 0])
RBG = uigetcolor(0, 128, 255)
RBG = uigetcolor("My color chooser", 0, 128, 255)
```

See Also

getcolor

Authors

Vincent COUVERT

Name

uigetdir — dialog for selecting a directory

```
directory = uigetdir()
directory = uigetdir(start_path [,title])
```

Parameters

start_path
: a character string which gives the initial directory used for search. By default uigetdir uses current working directory.

title
: the title for the uigetdir window.

directory
: is the user selected directory if user answers "Ok" or the " " string if user cancels.

Description

Creates a dialog window for selecting a directory

Examples

```
uigetdir()
uigetdir("SCI/modules/")
uigetdir("SCI/modules/", "Choose a directory")
```

See Also

uigetfile , tk_getdir

Name

uigetfile — dialog window to get a file(s) name(s), path and filter index

```
[FileName[,PathName[,FilterIndex]]]=uigetfile([file_mask[,dir[,boxTitle[,multip
PathFileName=uigetfile([file_mask[,dir[,boxTitle[,multiple]]]])
```

Input parameters

file_mask
: a string matrix which gives the file masks to use for file selection. `file_mask` is written with Unix convention. The default value is '*'.

  we can also add descriptions for masks, for example ["*.x*","X files";"*.bin","BIN files"].

dir
: a character string which gives the initial directory used for file search. By default uigetfile uses the previously selected directory.

boxTitle
: a character string which gives the title of the uigetfile window. By default uigetfile's title is 'uigetfile'.

multipleSelection
: a boolean which allows to load only one file if it is at '%f' (false) or multiple files if it is at '%t" (true). By default uigetfile's multiple file selection is not enable.

Output parameters

FileName
: matrix of string which give the user selected file(s) (path + file(s) name(s)) if user answers "Ok" or the " " string if user answers "Cancel".

PathName
: is the user selected file(s) path if user answers "Ok" or the " " string if user answers "Cancel".

FilterIndex
: is the user selected filter index on the list box if user answers "Ok" or '0' string if user answers "Cancel"

Description

Creates a dialog window for file(s) selection

Examples

```
uigetfile(["*.bin";"*.sce";"*.cos*"])
uigetfile(["*.sci";"*.bin"],"SCI/modules/gui/macros/")
uigetfile(["*.sc*";"*.bin"],"SCI/modules/gui/macros/")
uigetfile(["*.x*","X files";"*.bin","BIN files"],"SCI/modules/gui/macros/")
uigetfile(["*.sce";"*.bin"],"SCI/modules/gui/macros/", "Choose a file name"", %
uigetfile(["*.sce";"*.bin"],"SCI/modules/gui/macros/", "Choose a file name"", %
```

See Also

uigetdir , x_dialog , file , read , write , exec , getf

Name

uigetfont — Opens a dialog for selecting a font.

```
uigetfont()

[fontname [,fontsize [,bold [,italic]]]] = uigetfont([defaultfontname [,default

[fontname ,fontsize ,bold ,italic] = uigetfont(defaultfontname ,defaultfontsize
```

Parameters

defaultfontname
: String: the default font name to select in the dialog.

defaultfontsize
: Scalar: the default font size to select in the dialog.

defaultbold
: Boolean: the default bold attribute in the dialog (%T for bold font, %F otherwise).

defaultitalic
: Boolean: the default italic attribute in the dialog (%T for bold font, %F otherwise).

fontname
: The selected font name ("" if the user cancels).

fontsize
: The selected font size ([] if the user cancels).

bold
: %T if bold attribute has been selected, %F otherwise ([] if the user cancels).

italic
: %T if italic attribute has been selected, %F otherwise ([] if the user cancels).

Description

Creates a dialog window for selecting a font.

Examples

```
uigetfont()
uigetfont("arial")
uigetfont("arial", 24)
uigetfont("arial", 24, %T)
uigetfont("arial", 24, %T, %F)
```

See Also

getfont

Authors

Vincent COUVERT

Name

uimenu — Create a menu or a submenu in a figure

```
h=uimenu([prop1,val1] [,prop2, val2] ...)
h=uimenu(parent,[prop1, val1] [,prop2, val2] ...)
```

Parameters

parent
: integer Handle of menu's parent

prop{1, 2 ...}
: string character name of a property to set up

val{1, 2 ...}
: scilab object value to affect to the corresponding property

h
: integer handle of the corresponding menu

Description

This allows to create menus in a figure. If `parent` is a figure, then the menu item will be added to the menu bar of the figure. If `parent` is a menu item , then the new item will be added to the parent item, allowing to create cascaded submenu. To create a customized menu, you can use the properties listed below:

Properties

Callback
: String

  Instruction evaluated by the Scilab interpreter when the menu is activated. Under MacOSX, the callback will not be executed for a "button menu" (a menu without children), you must specify at least a child.

Enable
: {on} | off

  Enable or disable the menu. If this property is set to `"on"` (default), the menu is operational, but if this property is set to `"off"`, the menu will not respond to the mouse actions and will be grayed out.

ForegroundColor
: [1,3] real vector or string

  Foreground color of the uimenu (font color). A color is specified as Red, Green and Blue values. Those values are real in [0,1]. The color can be given as a real vector, ie [R,G,B] or a string where each value is separated by a "|", ie "R|G|B".

Label
: String.

  This property represents the text appearing in the menu.

Tag
: String

This property is generally used to identify the menu. It allows to give it a "name". Mainly used in conjunction with `findobj()`.

Visible
{on} | off

Set the visibility of the uimenu. If this property is set to `"on"` (default), the uimenu is visible, but if this property is set to `"off"`, the uimenu will not appear in its parent figure.

Examples

```
f=figure('position', [10 10 300 200]);
// create a figure
m=uimenu(f,'label', 'windows');
// create an item on the menu bar
m1=uimenu(m,'label', 'operations');
m2=uimenu(m,'label', 'quit scilab', 'callback', "exit");
//create two items in the menu "windows"
m11=uimenu(m1,'label', 'new window', 'callback',"xselect()");
m12=uimenu(m1,'label', 'clear  window', 'callback',"xbasc()");
// create a submenu to the item "operations"
close(f);
// close the figure
```

See Also

figure, uicontrol, set, get

Authors

Bertrand Guiheneuf

Name

unsetmenu — interactive button or menu or submenu de-activation

```
unsetmenu(button,[nsub])
unsetmenu(gwin,button,[nsub])
```

Parameters

button
: a character string. The button name

gwin
: integer. The number of graphic window where the button is installed

nsub
: integer. The number of submenu to de-activate (if any). If button has no sub-menu, `nsub` is ignored

Description

The function allows the user to desactivate buttons or menus created by `addmenu` in the main or graphics windows command panels.

Examples

```
//addmenu('foo')
//unsetmenu('foo')
//unsetmenu('File',2)
```

See Also

delmenu , setmenu , addmenu

Name

usecanvas — Get/Set the main component used for Scilab graphics.

```
[canvasused] = usecanvas([usecanvasfordisplay]);
```

Parameters

canvasused
Boolean:

- %T if a "GLCanvas" is used for graphics display (Mixing uicontrols and graphics **not available**).
- %F if a "GLJPanel" is used for graphics display (Mixing uicontrols and graphics available).

usecanvasfordisplay
Boolean:

- %T to use a "GLCanvas" for graphics display (Mixing uicontrols and graphics **not available**).
- %F to use a "GLJPanel" for graphics display (Mixing uicontrols and graphics available).

Description

Scilab uses a "GLJPanel" (a Swing OpenGL component) to display graphics (plot3d, plot, ...). This component uses some high level OpenGL primitives which are not correctly supported on some platforms (depending on the operating system, video cards, drivers ...)

"GLCanvas" (AWT + OpenGL) is an alternative component provided by the Java Framework. Scilab can use it to render graphics. **However, using this component disables some capabilities such as mixing plots and uicontrols (see demo GUI/UIcontrol2). That is why it is not the default behavior.**

In some particular cases, the use of the "GLCanvas" component is forced when Scilab starts (a warning message is displayed when a graphics function is used for the first time), here is a list of these cases:

Operating System	**Video Card**	**Details**
64-bits Windows	All	When Scilab is used in a remote session.
Linux	NVIDIA Card	With free drivers.
ATI Card	With free drivers or ATI-drivers with version < 8.52.3 (Installer version < 8.8 / OpenGL version < 2.1.7873).	
INTEL Card	With Direct Rendering activated.	

You can also dynamically activate this component through Scilab using usecanvas:

- usecanvas(%T) will use "GLCanvas" for plot rendering.
- usecanvas(%F) will use "GLJPanel" for plot rendering. If your configuration is known as a one having problems with "GLJPanel" (See table above), a warning message will be displayed.

If you believe your configuration is able to use the "GLJPanel" and Scilab automatically forces the use of "GLCanvas", you can test your configuration by swithing to "GLJPanel" (usecanvas(%F))

and try to plot something (`plot3d()` for example). If Scilab graphics work, please inform us about it by sending an email to `scilab.support@scilab.org` and giving us your Operating System/ Video Card/Video Card driver version: this will help use to improve future versions of Scilab.

Technical Aspects

Since version 5.0, Scilab is doing an advanced use of JOGL (the Java Binding for the OpenGL), which is using the Java2D OpenGL Pipeline. For performance reasons, we use the Java2D OpenGL Pipeline. From a more technical aspect, it uses the internal buffer of the graphic cards called `pbuffer`.

Problems may occur when the driver of the graphic card does not support properly this approach. As far as we know, there is no free driver under Linux handling this feature. In the proprietary world, the situation is as follows:

- `NVIDIA`: Nvidia provides the appropriate proprietary drivers. Scilab's graphics work without any problem with most NVIDIA drivers.
- `ATI`: From the driver version 8.8, most ATI graphics supports the pbuffer under Linux.
- `Intel`: This is the big drawback of using the pbuffer. There is currently no support of pbuffer by any official Intel drivers under Linux.

There is a workaround for Linux to tackle this issue, but a solution is to use a software accelerated driver. To do it, in `/etc/X11/xorg.conf`, look for the *Section "Device"* and change the option *Driver* to *vesa*:

```
Section "Device"
        Identifier      "Your Graphic card"
        Driver  "vesa"
[...]
EndSection
```

Unfortunately, this solution makes Scilab pretty slow.

Under Windows, video cards manufacturers update regularly and `pbuffers` are managed. Please download recent drivers at:

- For `ATI cards`: http://ati.amd.com/support/driver.html
- For `Intel cards`: http://www.intel.com/support/graphics/
- For `Matrox cards`: http://www.matrox.com/graphics/en/support/drivers/
- For `NVIDIA cards`: http://www.nvidia.com/content/drivers/drivers.asp
- For `S3 cards`: http://www.s3graphics.com/en/resources/drivers/index.jsp
- For `SiS cards`: http://www.sis.com/download/
- For `VIA cards`: http://www.viaarena.com/default.aspx?PageID=2

Some troubles can also occur when using Windows 2000 (video drivers are no more updated and no more supported).

In the cases where pBuffer create a problem, waiting for a working `pbuffer` is not a solution indeed: *The OpenGL community is moving away from pbuffers and toward the frame buffer object extension,*

which is a more portable and higher-performance solution for offscreen rendering than pbuffers. [https://jogl.dev.java.net/issues/show_bug.cgi?id=163]. The JOGL team is working to fix this issue.

For more information about this problem, please refer to:

- `JoGL` bug database: Bug #366 [https://jogl.dev.java.net/issues/show_bug.cgi?id=366]
- `Scilab` bug database: Bug #3525 [http://bugzilla.scilab.org/show_bug.cgi?id=3525]
- `Debian` bug database: Bug #501799 [http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=501799]
- `Freedesktop` bug database: Bug #17603 [https://bugs.freedesktop.org/show_bug.cgi?id=17603]

Examples

```
// Example using GLJPanel (Mixing uicontrols and graphics is available)
usecanvas(%F);
plot2d();
uicontrol("String", "Close the window", "Position", [10 10 100, 25], "Callback"
messagebox("You can see the button on the figure.", "Usecanvas example", "info"

// Example using GLCanvas (Mixing uicontrols and graphics is not available, uic
usecanvas(%T);
plot2d();
uicontrol("String", "Close the window", "Position", [10 10 100, 25], "Callback"
messagebox("You can''t see any button on the figure.", "Usecanvas example", "in
```

Authors

Vincent COUVERT

Name

waitbar — Draw a waitbar

```
winId=waitbar(x)
winId=waitbar(x,mes)
winId=waitbar(mes)
waitbar(x,winId)
waitbar(mes,winId)
waitbar(x,mes,winId)
```

Parameters

x
: real, fraction to display.

mes
: string, message to display.

winId
: integer greater than 0, window identificator.

Description

`waitbar(x)` create a new waitbar displaying a fraction x, return window identificator.

`waitbar(x,mes)` create a new waitbar displaying a fraction x and message mes, return window identificator.

`waitbar(mes)` create a new waitbar displaying a fraction 0 and message mes, return window identificator.

`waitbar(x,mes)` create a new waitbar displaying a fraction 0 and message mes, return window identificator.

`waitbar(x,winId)`, `waitbar(mes,winId)` and `waitbar(x,mes,winId)` update waitbar with window identificator winId.

Examples

```
winId=waitbar('This is an example');
realtimeinit(0.3);
for j=0:0.1:1,
    realtime(3*j);
    waitbar(j,winId);
end
winclose(winId);
```

Authors

Jaime Urzua

Name

x_choices — interactive Xwindow choices through toggle buttons

```
rep=x_choices(title,items)
```

Parameters

title
: vector of strings, title for the popup window.

items
: a list of items `items=list(item1,...,itemn)`, where each `item` is also a list of the following type : `item=list('label',default_choice,choices)`. `default_choice` is an integer which gives the default toggle on entry and `choices` is a row vector of strings which gives the possible choices.

rep
: an integer vector which gives for each item the number of the selected toggle. If user exits dialog with "cancel" button `rep` is set to `[]`.

Description

Select items through toggle lists and return in `rep` the selected items

Type `x_choices()` to see an example.

Examples

```
l1=list('choice 1',1,['toggle c1','toggle c2','toggle c3']);
l2=list('choice 2',2,['toggle d1','toggle d2','toggle d3']);
l3=list('choice 3',3,['toggle e1','toggle e2']);
rep=x_choices('Toggle Menu',list(l1,l2,l3));
```

Name

x_choose — interactive window choice (modal dialog)

```
[num]=x_choose(items,title [,button])
```

Parameters

items
: column vector of string, items to choose

title
: column vector of string, comment for the dialog

button
: string, text to appear in the button. Default value is 'Cancel'

num
: integer, choosen item number or 0 if dialog resumed with "Cancel" button

Description

Returns in `num` the number of the chosen item.

WARNING: this dialog was not modal before Scilab 5.0, please use x_choose_modeless for ascendant compatibility.

Examples

```
n=x_choose(['item1';'item2';'item3'],['that is a comment';'for the dialog'])
n=x_choose(['item1';'item2';'item3'],['that is a comment'],'Return')
```

See Also

x_choose_modeless , x_choices , x_mdialog , getvalue , unix_g

Name

x_choose_modeless — interactive window choice (not modal dialog)

```
[num]=x_choose_modeless(items,title [,button])
```

Parameters

items
: column vector of string, items to choose

title
: column vector of string, comment for the dialog

button
: string, text to appear in the button. Default value is 'Cancel'

num
: integer, choosen item number or 0 if dialog resumed with "Cancel" button

Description

Returns in `num` the number of the chosen item.

Examples

```
n=x_choose_modeless(['item1';'item2';'item3'],['that is a comment';'for the dia
n=x_choose_modeless(['item1';'item2';'item3'],['that is a comment'],'Return')
```

See Also

x_choose , x_choices , x_mdialog , getvalue , unix_g

Name

x_dialog — Xwindow dialog

```
result=x_dialog(labels,valueini)
```

Parameters

labels
: column vector of strings, comment for dialog

valueini
: n column vector of strings, initial value suggested

result
: response : n column vector of strings if returned with "Ok" button or [] if returned with "Cancel" button

Description

Creates an X-window multi-lines dialog

Examples

```
gain=evstr(x_dialog('value of gain ?','0.235'))
x_dialog(['Method';'enter sampling period'],'1')
m=evstr(x_dialog('enter a  3x3 matrix ',['[0 0 0';'0 0 0';'0 0 0]']))
```

See Also

x_mdialog , x_matrix , evstr , execstr

Name

x_matrix — Xwindow editing of matrix

```
[result]=x_matrix(label,matrix-init)
```

Parameters

label
: character string (name of matrix)

matrix-init
: real matrix

Description

For reading or editing a matrix .

Examples

```
//m=evstr(x_matrix('enter a  3x3 matrix ',rand(3,3)))
```

See Also

x_mdialog , x_dialog

Name

x_mdialog — Xwindow dialog

```
result=x_mdialog(title,labels,default_inputs_vector)
result=x_mdialog(title,labelsv,labelsh,default_input_matrix)
```

Parameters

title
: column vector of strings, dialog general comment

labels
: n column vector of strings, `labels(i)` is the label of the ith required value

default_input_vector
: n column vector of strings, `default_input_vector(i)` is the initial value of the ith required value

labelsv
: n vector of strings, `labelsv(i)` is the label of the ith line of the required matrix

labelsh
: m vector of strings, `labelsh(j)` is the label of the jth column of the required matrix

default_input_matrix
: n x m matrix of strings, `default_input_matrix(i,j)` is the initial value of the (i,j) element of then required matrix

result
: n x m matrix of string if returned with "Ok" button or [] if returned with "Cancel" button

Description

X-window vector/matrix interactive input function

Examples

```
txt=['magnitude';'frequency';'phase    '];
sig=x_mdialog('enter sine signal',txt,['1';'10';'0'])
mag=evstr(sig(1))
frq=evstr(sig(2))
ph=evstr(sig(3))

rep=x_mdialog(['System Simulation';'with PI regulator'],...
                     ['P gain';'I gain '],[' ';' '])
```

See Also

editvar , x_dialog , x_choose , x_message , getvalue , evstr , execstr , editvar

Name

x_message — X window message

```
[num]=x_message(strings [,buttons])
```

Parameters

strings
: vector of characters strings to be displayed

buttons
: character string or 2 vector of character strings which specifies button(s) name(s). Default value is "Ok"

num
: number of button clicked (if 2 buttons are specified)

Description

for displaying a message (diagnostic,...) and waiting for an answer (button click). The function returns only after a click on a button.

Examples

```
gain=0.235;x_message('value of gain is :'+string(gain))
x_message(['singular matrix';'use least squares'])

r=x_message(['Your problem is ill conditioned';
            'continue ?'],['Yes','No'])
```

See Also

x_dialog , x_mdialog , x_message_modeless

Name

x_message_modeless — X window modeless message

```
x_message_modeless(strings)
```

Parameters

strings
: vector of characters strings to be displayed

Description

for displaying a message (information, user-guide ...). The function returns immediately. The message window is killed when "Ok" button is clicked.

Examples

```
x_message_modeless(['This is a modeless message'
                    'Scilab may continue computation'
                    ' '
                    'Click on ""Ok"" to close the message'])
x_message_modeless('Now two message windows are opened')
```

See Also

x_dialog , x_mdialog , x_message

Name

xgetfile — dialog to get a file path

```
path=xgetfile([title='string'])
path=xgetfile(file_mask,[title='string'])
path=xgetfile(file_mask,dir,[title='string'])
path=xgetfile(file_mask,dir,'string')
```

Parameters

file_mask
: a character string which gives the file mask to use for file selection. `file_mask` is written with Unix convention. the default value is '*'.

dir
: a character string which gives the initial directory used for file search. by default xgetfile uses the previously selected directory.

path
: is the user selected file path if user answers "Ok" or the " " string if user answers "Cancel"

title='string'
: :Optional arguments which gives the title for the xgetfile window.

Description

Creates a dialog window for file selection

Examples

```
xgetfile()
xgetfile(title="Choose a file name")
xgetfile("*.sci")
xgetfile("*.sci", title="Choose a file name")
xgetfile("*.sci", "SCI/modules/gui/macros/")
xgetfile("*.sci", "SCI/modules/gui/macros/", title="Choose a file name")
xgetfile("*.sci", "SCI/modules/gui/macros/", "Choose a file name")
```

See Also

uigetdir , x_dialog , file , read , write , exec , getf

Genetic Algorithms

Name

coding_ga_binary — A function which performs conversion between binary and continuous representation

```
pop_out = coding_ga_binary(pop_in,direction,param)
```

Parameters

pop_in
: a list which contains all the individuals in the current population.

direction
: 'code' or 'decode'. If direction == 'code' then we perform a continuous to binary encoding. Else, we convert from binary to continuous.

param
: a parameter list.

  - 'binary_length': the number of bits by variables. If binary_length = 8 and the variable X is of dimension 2 then the binary code will be 16 bits length.
  - 'minboun': a vector of minimum bounds for the variable X.
  - 'maxbound': a vector of maximum bounds for the variable X.

pop_out
: the population coded to binary or decoded to continuous values.

Description

- This function allows to code or decode a population of individuals from (resp. to) continuous variables to (resp. from) binary.

See Also

optim_ga , mutation_ga_binary , crossover_ga_binary

Authors

Yann COLLETTE
: ycollet@freesurf.fr

Name

coding_ga_identity — A "no-operation" conversion function

```
pop_out = coding_ga_identity(pop_in,direction,param)
```

Parameters

pop_in
: the population to be converted.

direction
: 'code' or 'decode'. This value has no influence of the state of pop_in.

param
: a parameter list. For this function, there are no useful parameters set.

pop_out
: a population identical to pop_in.

Description

- This function is a do-nothing function. It is essentially useful to implement an evolutionnary algorithm. In an evolutionnary algorithm, we work directly on the variable and not on a binary code.

See Also

mutation_func_default , crossover_func_default , init_func_default , optim_ga

Authors

Yann COLLETTE
: ycollet@freesurf.fr

Name

crossover_ga_binary — A crossover function for binary code

```
[Crossed_Indiv1,Crossed_Indiv2] = crossover_ga_binary(Indiv1,Indiv2,param)
```

Parameters

Indiv1
: the first individual (here a binary code) to be crossed-over.

Indiv2
: the second individual to be crossed-over.

param
: a list of parameters.

  - 'binary_length': the length of the binary code.
  - 'multi_cross': a boolean. If %T then we allow several cuts in the binary code.
  - 'multi_cross_nb': the number of cuts in the binary code. Only used when multi_cross is set to %T.

Crossed_Indiv1
: The first individual obtained by the cross-over function.

Crossed_Indiv2
: The second individual obtained by the cross-over function.

Description

- This function implements a classical binary cross-over.

See Also

crossover_ga_binary , crossover_ga_default , mutation_ga_binary , optim_ga

Authors

Yann COLLETTE
: ycollet@freesurf.fr

Name

crossover_ga_default — A crossover function for continuous variable functions

```
[Crossed_Indiv1,Crossed_Indiv2] = crossover_ga_default(Indiv1,Indiv2,param)
```

Parameters

Indiv1
: The first individual to be crossed-over.

Indiv2
: The second individual to be crossed-over.

param
: a list of parameters.

  - 'beta': the range of the random generator. A random value will be sampled between -beta and 1+beta. This sampled value will be used to perform a convex combination between Indiv1 and Indiv2.
  - 'minbound': a vector of minimum bounds for the variable X.
  - 'maxbound': a vector of maximum bounds for the variable X.

Crossed_Indiv1
: The first individual resulting from the crossover.

Crossed_Indiv2
: The second individual resulting from the crossover.

Description

crossover_ga_default is a crossover function for functions with continuous variables. This crossover function is an extension of a convexe combination. The crossed individuals are computed with the following equations :

```
mix = (1 + 2*Beta)*rand(1,1) - Beta;
Crossed_Indiv1 =     mix*Indiv1 + (1-mix)*Indiv2;
Crossed_Indiv2 = (1-mix)*Indiv1 +     mix*Indiv2;
```

The Beta parameter should be set to a positive value. If Beta is set to 0, the resulting crossover is a simple convexe combination between the two parents. That may lead to a too fast convergence of the genetic algorithm and may decrease the diversity of the individuals of the population. If Beta is chosen strictly positive, that may allow children to explore the domain beyond the domain explored by their parents.

See Also

crossover_ga_binary , mutation_ga_default , init_ga_default , optim_ga

References

Michalewicz, Zbigniew Genetic Algorithms + Data Structures = Evolution Programs

Authors

Yann COLLETTE
ycollet@freesurf.fr

Name

init_ga_default — A function a initialize a population

```
Pop_init = init_ga_default(popsize,param)
```

Parameters

popsize
: the number of individuals to generate.

param
: a list of parameters.

- 'dimension': the size of the vector X.
- 'minbound': a vector of minimum bounds for the variable X.
- 'maxbound': a vector of maximum bounds for the variable X.

Pop_init
: a list which contains the initial population of individuals.

Description

- This function generate an initial population containing pop_size individuals.

See Also

crossover_ga_default , mutation_ga_default , mutation_ga_binary , optim_ga

Authors

Yann COLLETTE
: ycollet@freesurf.fr

Name

mutation_ga_binary — A function which performs binary mutation

```
Mut_Indiv = mutation_ga_binary(Indiv,param)
```

Parameters

Indiv
: the individual on which we will perform the mutation.

param
: a list of parameters.

  - 'binary_length': the size of the binary code.
  - 'multi_mut': a boolean. If %T, several random bits will be flipped.
  - 'multi_mut_nd': the number of bits to be flipped. Works only when multi_mut is set to %T.

Mut_Indiv
: The mutated individual.

Description

- This function performs a classical multi-bits binary mutation.

See Also

mutation_ga_default , crossover_ga_binary , init_func_default , optim_ga

Authors

Yann COLLETTE
: ycollet@freesurf.fr

Name

mutation_ga_default — A continuous variable mutation function

```
Mut_Indiv = mutation_ga_default(Indiv,param)
```

Parameters

Indiv
: The individual to be mutated.

param
: a list of parameters.

- 'delta': a random perturbation will be sampled via an uniform distribution between -delta and + delta.
- 'minbound': a vector of minimum bound for the variable X.
- 'maxbound': a vector of maximum bound for the variable X.

Mut_Indiv
: The resulting mutated individual.

Description

- This function performs the classical continuous variable mutation function.

See Also

mutation_ga_binary , crossover_ga_default , init_ga_default , optim_ga

Authors

Yann COLLETTE
: ycollet@freesurf.fr

Name

optim_ga — A flexible genetic algorithm

```
[pop_opt,fobj_pop_opt,pop_init,fobj_pop_init] = optim_ga(ga_f,pop_size,nb_genera
```

Parameters

ga_f
: the function to be optimized. The prototype if y = f(x) or y = list(f,p1,p2,...).

pop_size
: the size of the population of individuals (default value: 100).

nb_generation
: the number of generations (equivalent to the number of iterations in classical optimization) to be computed (default value: 10).

p_mut
: the mutation probability (default value: 0.1).

p_cross
: the crossover probability (default value: 0.7).

Log
: if %T, we will display to information message during the run of the genetic algorithm.

param
: a list of parameters.

  - 'codage_func': the function which will perform the coding and decoding of individuals (default function: codage_identity).
  - 'init_func': the function which will perform the initialization of the population (default function: init_ga_default).
  - 'crossover_func': the function which will perform the crossover between two individuals (default function: crossover_ga_default).
  - 'mutation_func': the function which will perform the mutation of one individual (default function: mutation_ga_default).
  - 'selection_func': the function whcih will perform the selection of individuals at the end of a generation (default function: selection_ga_elitist).
  - 'nb_couples': the number of couples which will be selected so as to perform the crossover and mutation (default value: 100).
  - 'pressure': the value the efficiency of the worst individual (default value: 0.05).

pop_opt
: the population of optimal individuals.

fobj_pop_opt
: the set of objective function values associated to pop_opt (optional).

pop_init
: the initial population of individuals (optional).

fobj_pop_init
: the set of objective function values associated to pop_init (optional).

Description

- This function implements the classical genetic algorithm.

Examples

```
deff('y=f(x)','y = sum(x.^2)');

PopSize     = 100;
Proba_cross = 0.7;
Proba_mut   = 0.1;
NbGen       = 10;
NbCouples   = 110;
Log         = %T;
nb_disp     = 10; // Nb point to display from the optimal population
pressure    = 0.05;

ga_params = init_param();
// Parameters to adapt to the shape of the optimization problem
ga_params = add_param(ga_params,'minbound',[-2; -2]);
ga_params = add_param(ga_params,'maxbound',[2; 2]);
ga_params = add_param(ga_params,'dimension',2);
ga_params = add_param(ga_params,'beta',0);
ga_params = add_param(ga_params,'delta',0.1);
// Parameters to fine tune the Genetic algorithm. All these parameters are
// If you need to adapt the GA to a special problem, you
ga_params = add_param(ga_params,'init_func',init_ga_default);
ga_params = add_param(ga_params,'crossover_func',crossover_ga_default);
ga_params = add_param(ga_params,'mutation_func',mutation_ga_default);
ga_params = add_param(ga_params,'codage_func',codage_ga_identity);
ga_params = add_param(ga_params,'selection_func',selection_ga_elitist);
//ga_params = add_param(ga_params,'selection_func',selection_ga_random);
ga_params = add_param(ga_params,'nb_couples',NbCouples);
ga_params = add_param(ga_params,'pressure',pressure);

Min = get_param(ga_params,'minbound');
Max = get_param(ga_params,'maxbound');
x0  = (Max - Min) .* rand(size(Min,1),size(Min,2)) + Min;

[pop_opt, fobj_pop_opt, pop_init, fobj_pop_init] = optim_ga(f, PopSize, NbG
```

See Also

optim_moga , optim_nsga , optim_nsga2

References

Michalewicz, Zbigniew Genetic Algorithms + Data Structures = Evolution Programs

Authors

Yann COLLETTE
ycollet@freesurf.fr

Name

optim_moga — multi-objective genetic algorithm

```
[pop_opt,fobj_pop_opt,pop_init,fobj_pop_init] = optim_moga(ga_f,pop_size,nb_gen
```

Parameters

ga_f
: the function to be optimized. The header of the function is the following :

```
y = f(x)
```

or

```
y = list(f,p1,p2,...)
```

pop_size
: the size of the population of individuals (default value: 100).

nb_generation
: the number of generations (equivalent to the number of iterations in classical optimization) to be computed (default value: 10).

p_mut
: the mutation probability (default value: 0.1).

p_cross
: the crossover probability (default value: 0.7).

Log
: if %T, we will display to information message during the run of the genetic algorithm.

param
: a list of parameters.

- 'codage_func': the function which will perform the coding and decoding of individuals (default function: codage_identity).
- 'init_func': the function which will perform the initialization of the population (default function: init_ga_default).
- 'crossover_func': the function which will perform the crossover between two individuals (default function: crossover_ga_default).
- 'mutation_func': the function which will perform the mutation of one individual (default function: mutation_ga_default).
- 'selection_func': the function whcih will perform the selection of individuals at the end of a generation (default function: selection_ga_elitist).
- 'nb_couples': the number of couples which will be selected so as to perform the crossover and mutation (default value: 100).
- 'pressure': the value the efficiency of the worst individual (default value: 0.05).

pop_opt
: the population of optimal individuals.

fobj_pop_opt
: the set of multi-objective function values associated to pop_opt (optional).

pop_init
: the initial population of individuals (optional).

fobj_pop_init
: the set of multi-objective function values associated to pop_init (optional).

Description

- This function implements the classical "Multi-Objective Genetic Algorithm". For a demonstration: see SCI/modules/genetic_algorithms/examples/MOGAdemo.sce.

See Also

optim_ga , optim_nsga , optim_nsga2

Authors

Yann COLLETTE
: ycollet@freesurf.fr

Name

optim_nsga — A multi-objective Niched Sharing Genetic Algorithm

```
[pop_opt,fobj_pop_opt,pop_init,fobj_pop_init] = optim_nsga(ga_f,pop_size,nb_gen
```

Parameters

ga_f
: the function to be optimized. The prototype if y = f(x) or y = list(f,p1,p2,...).

pop_size
: the size of the population of individuals (default value: 100).

nb_generation
: the number of generations (equivalent to the number of iterations in classical optimization) to be computed (default value: 10).

p_mut
: the mutation probability (default value: 0.1).

p_cross
: the crossover probability (default value: 0.7).

Log
: if %T, we will display to information message during the run of the genetic algorithm.

param
: a list of parameters.

  - 'codage_func': the function which will perform the coding and decoding of individuals (default function: codage_identity).
  - 'init_func': the function which will perform the initialization of the population (default function: init_ga_default).
  - 'crossover_func': the function which will perform the crossover between two individuals (default function: crossover_ga_default).
  - 'mutation_func': the function which will perform the mutation of one individual (default function: mutation_ga_default).
  - 'selection_func': the function whcih will perform the selection of individuals at the end of a generation (default function: selection_ga_elitist).
  - 'nb_couples': the number of couples which will be selected so as to perform the crossover and mutation (default value: 100).
  - 'pressure': the value the efficiency of the worst individual (default value: 0.05).

sigma
: the radius of the sharing area.

pow
: the power coefficient of the penalty formula.

pop_opt
: the population of optimal individuals.

fobj_pop_opt
: the set of objective function values associated to pop_opt (optional).

pop_init
: the initial population of individuals (optional).

fobj_pop_init
: the set of objective function values associated to pop_init (optional).

Description

- This function implements the classical "Niched Sharing Genetic Algorithm". For a demonstration, see SCI/modules/genetic_algorithms/examples/NSGAdemo.sce.

See Also

optim_moga , optim_ga , optim_nsga2

Authors

Yann COLLETTE
: ycollet@freesurf.fr

Name

optim_nsga2 — A multi-objective Niched Sharing Genetic Algorithm version 2

```
[pop_opt,fobj_pop_opt,pop_init,fobj_pop_init] = optim_nsga2(ga_f,pop_size,nb_ge
```

Parameters

ga_f
: the function to be optimized. The prototype if y = f(x) or y = list(f,p1,p2,...).

pop_size
: the size of the population of individuals (default value: 100).

nb_generation
: the number of generations (equivalent to the number of iterations in classical optimization) to be computed (default value: 10).

p_mut
: the mutation probability (default value: 0.1).

p_cross
: the crossover probability (default value: 0.7).

Log
: if %T, we will display to information message during the run of the genetic algorithm.

param
: a list of parameters.

  - 'codage_func': the function which will perform the coding and decoding of individuals (default function: codage_identity).
  - 'init_func': the function which will perform the initialization of the population (default function: init_ga_default).
  - 'crossover_func': the function which will perform the crossover between two individuals (default function: crossover_ga_default).
  - 'mutation_func': the function which will perform the mutation of one individual (default function: mutation_ga_default).
  - 'selection_func': the function whcih will perform the selection of individuals at the end of a generation (default function: selection_ga_elitist).
  - 'nb_couples': the number of couples which will be selected so as to perform the crossover and mutation (default value: 100).
  - 'pressure': the value the efficiency of the worst individual (default value: 0.05).

pop_opt
: the population of optimal individuals.

fobj_pop_opt
: the set of objective function values associated to pop_opt (optional).

pop_init
: the initial population of individuals (optional).

fobj_pop_init
: the set of objective function values associated to pop_init (optional).

Description

- This function implements the classical "Niched Sharing Genetic Algorithm". For a demonstration, see SCI/modules/genetic_algorithms/examples/NSGA2demo.sce.

See Also

optim_moga , optim_ga , optim_nsga

Authors

Yann COLLETTE
ycollet@freesurf.fr

Name

pareto_filter — A function which extracts non dominated solution from a set

```
[F_out,X_out,Ind_out] = pareto_filter(F_in,X_in)
```

Parameters

F_in
: the set of multi-objective function values from which we want to extract the non dominated solutions.

X_in
: the associated values in the parameters space.

F_out
: the set of non dominated multi-objective function values.

X_out
: the associated values in the parameters space.

Ind_out
: the set of indexes of the non dominated individuals selected from the set X_in.

Description

- This function applies a Pareto filter to extract non dominated solutions from a set of values.

See Also

optim_moga , optim_nsga , optim_nsga2

Authors

Yann COLLETTE
ycollet@freesurf.Fr

Name

selection_ga_elitist — An 'elitist' selection function

```
[Pop_out,FObj_Pop_out,Efficiency,MO_Total_FObj_out] = selection_ga_elitist(Pop_
```

Parameters

Pop_in
: The initial population of individuals.

Indiv1
: a first set of childs generated via crossover + mutation.

Indiv2
: a second set of childs generated via crossover + mutation.

FObj_Pop_in
: a vector of objective function values associated to each individuals of Pop_in.

FObj_Indiv1
: a vector of objective function values associated to each individuals of Indiv1.

FObj_Indiv2
: a vector of objective function values associated to each individuals of Indiv2.

MO_Total_FObj_in
: a matrix of multi-objective function values associated to each individuals of Pop_in.

MO_FObj_Indiv1
: a matrix of multi-objective function values associated to each individuals of Indiv1.

MO_FObj_Indiv2
: a matrix of multi-objective function values associated to each individuals of Indiv2.

param
: a list of parameters. - 'pressure': the selection pressure coefficient. Each individuals with 0 efficiency will have an efficiency value equal to 'pressure'.

Pop_out
: all the selected individuals in a population of size pop_size.

FObj_Pop_out
: all the objective function values associated to each individuals of Pop_out.

Efficiency
: all the efficiency values associated to each individuals of Pop_out.

MO_Total_FObj_out
: all the multi-objective function values associated to each individuals of Pop_out.

Description

- This function performs the elitist selection function. We select the best individuals in the set of parents and childs individuals.

See Also

selection_ga_random , mutation_ga_default , crossover_ga_default , init_ga_default , optim_ga

Authors

Yann COLLETTE
ycollet@freesurf.fr

Name

selection_ga_random — A function which performs a random selection of individuals

```
[Pop_out,FObj_Pop_out,Efficiency,MO_Total_FObj_out] = selection_ga_random(Pop_i
```

Parameters

Pop_in
: The initial population of individuals.

Indiv1
: a first set of childs generated via crossover + mutation.

Indiv2
: a second set of childs generated via crossover + mutation.

FObj_Pop_in
: a vector of objective function values associated to each individuals of Pop_in.

FObj_Indiv1
: a vector of objective function values associated to each individuals of Indiv1.

FObj_Indiv2
: a vector of objective function values associated to each individuals of Indiv2.

MO_Total_FObj_in
: a matrix of multi-objective function values associated to each individuals of Pop_in.

MO_FObj_Indiv1
: a matrix of multi-objective function values associated to each individuals of Indiv1.

MO_FObj_Indiv2
: a matrix of multi-objective function values associated to each individuals of Indiv2.

param
: a list of parameters.

  - 'pressure': the selection pressure coefficient. Each individuals with 0 efficiency will have an efficiency value equal to 'pressure'.

Pop_out
: all the selected individuals in a population of size pop_size.

FObj_Pop_out
: all the objective function values associated to each individuals of Pop_out.

Efficiency
: all the efficiency values associated to each individuals of Pop_out.

MO_Total_FObj_out
: all the multi-objective function values associated to each individuals of Pop_out.

Description

- This function performs the random selection function. We select pop_size individuals in the set of parents and childs individuals at random.

See Also

selection_ga_elitist , mutation_ga_default , crossover_ga_default , init_ga_default , optim_ga

Authors

Yann COLLETTE
ycollet@freesurf.fr

Graphics : exporting and printing

Name

driver — select a graphics driver

```
driver(driver_name)
current_driver=driver()
```

Parameters

driver_name
: string, driver to be selected.

Description

This function is used to select a graphics driver, or with no arguments to get the current graphics driver name. Most of the time, a user can ignore this function and change the driver by calling high level functions such as `xbasc`. The selected driver can be one of the followings:

"X11"
: output to the screen of the computer.

"Pos"
: output into Postscript format.

"Rec"
: output to the screen of the computer. Same as `X11`.

"Fig"
: output into XFig format.

"GIF"
: output into Gif format.

"PPM"
: output into PPM format.

Remark

To convert "GIF" or "PPM" files to other image format or for building animation one can use the "convert" program for ImageMagic (http://www.imagemagick.org/)

For example if one has generated a sequence of Gif files named `img*.gif` it is possible to build an animated Gif file (named anim.gif) by

```
convert -delay 10  img*.gif anim.gif
```

See Also

xbasc

Authors

J.Ph.C.

Name

xend — close a graphics session

```
xend()
```

Description

`xend` is used to close a graphics session. Under the Postscript, Xfig or Gif drivers `xend` closes the file which was opened by `xinit`.

Examples

```
driver("Pos")
xinit("foo.ps")
plot2d()
xend()
driver("X11")
```

See Also

xinit

Authors

J.Ph.C.

Name

xinit — Initialization of a graphics driver

```
xinit(FileName)
xinit()
```

Parameters

FileName
: string: name of the export file.

Description

For the Postscript, Xfig, Gif or PPM driver, `FileName` must be specified. It is the name of the file where all the graphics operations are recorded.

For screen drivers (X11 or Rec), `xinit` should be called without any argument and opens an empty graphic window.

Examples

```
driver("Pos")
xinit("foo.ps")
plot2d()
xend()
driver("X11")
```

See Also

driver , xend , scf

Authors

J.Ph.C.

Jean-Baptiste Silvy

Name

xs2bmp — send graphics to a file in BMP syntax

```
xs2bmp(win_num,filen)
```

Parameters

win_num
: integer scalar.

filen
: string, file name.

Description

`xs2bmp` sends the recorded graphics of the window `win_num` in the file `filen` in BMP format.

Examples

```
scf(0)
plot2d()
//BMP export
xs2bmp(0,'foo.bmp');
```

See Also

xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

Authors

A.C

Name

xs2emf — send graphics to a file in EMF syntax (Only for Windows)

```
xs2emf(win_num,filen [,orientation])
```

Parameters

win_num
: integer scalar.

filen
: string, file name.

orientation
: optional character, with possible values 'p' (portrait) or 'l' (landscape). The default value is 'p'.

Description

`xs2emf` sends the recorded graphics of the window `win_num` in the file `filen` in EMF format.

For format EMF we create an EPS file which will be convert into EMF format by pstoedit.

Examples

```
if MSDOS then
  scf(0);
  plot2d();
  //EMF export
  xs2emf(0,'foo.emf');
end
```

See Also

xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig

Authors

A.C

Name

xs2eps — save graphics to a Postscript file.

```
xs2eps(win_num,filen [,orientation])
```

Parameters

win_num
: integer scalar or vector .

filen
: string, file name.

orientation
: optional character, with possible values 'p' (portrait) or 'l' (landscape). The default value is 'p'.

Description

`xs2eps` saves the recorded graphics of the window `win_num` to file `filen` in Postscript syntax. Note that `filen` must not have extension.

`xs2eps` produces a complete encapsulated Postscrip file.

Examples

```
scf(0);
plot2d();
//EPS export
filename='foo.eps';
xs2eps(0,filename);
```

See Also

set_posfig_dim, toprint, printfigure, xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

Name

xs2fig — send graphics to a file in FIG syntax

```
xs2fig(win_num, filen [,orientation])
```

Parameters

win_num
: integer scalar.

filen
: string, file name.

orientation
: optional character, with possible values 'p' (portrait) or 'l' (landscape). The default value is 'p'.

Description

`xs2fig` sends the recorded graphics of the window `win_num` in the file `filen` in FIG format.

For format FIG we create an EPS file which will be convert into FIG format by pstoedit.

To export FIG files GPL Ghostscript (32bits) need to be installed.

Link to get GPL Ghostscript : http://www.ghostscript.com/awki

Examples

```
//simple example
scf(0);
plot2d();
xs2fig(0,'foo.fig');
```

See Also

xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2emf

Authors

S.K

Name

xs2gif — send graphics to a file in GIF syntax

```
xs2gif(win_num,filen)
```

Parameters

win_num
: integer scalar or vector .

filen
: string, file name.

Description

`xs2gif` sends the recorded graphics of the window `win_num` in the file `filen` in GIF format.

To convert a sequence of "GIF" files to an animated GIF file one can use the "convert" program for ImageMagic (http://www.imagemagick.org/)

For example if one has generated a sequence of Gif files named `img*.gif` it is possible to build an animated Gif file (named anim.gif) by

```
   convert -delay 10  img*.gif anim.gif
```

Examples

```
    scf(0)
plot2d()
//GIF export
xs2gif(0,'foo.gif');
```

See Also

xs2bmp, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

Name

xs2jpg — send graphics to a file in JPG syntax

```
xs2jpg(win_num,filen)
```

Parameters

win_num
: integer scalar.

filen
: string, file name.

Description

`xs2jpg` sends the recorded graphics of the window `win_num` in the file `filen` in JPG format.

Examples

```
scf(0);
plot2d();
//JPG export
xs2jpg(0,'foo.jpg');
```

See Also

xs2bmp, xs2gif, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

Authors

S.K

Name

xs2pdf — save graphics to a PDF file.

```
xs2pdf(win_num,filen [,orientation])
```

Parameters

win_num
: integer scalar .

filen
: string, file name.

orientation
: optional character, with possible values 'p' (portrait) or 'l' (landscape). The default value is 'p'.

Description

`xs2pdf` saves the recorded graphics of the window `win_num` to file `filen` in PDF syntax. Note that `filen` must not have extension.

Examples

```
scf(0);
plot2d();
//PDF export
filename='foo'; // ! no extension
xs2pdf(0,filename);
```

See Also

set_posfig_dim, toprint, printfigure, xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2svg, xs2ps, xs2fig, xs2emf

Name

xs2png — send graphics to a file in PNG syntax

```
xs2png(win_num,filen)
```

Parameters

win_num
: integer scalar.

filen
: string, file name.

Description

`xs2png` sends the recorded graphics of the window `win_num` in the file `filen` in PNG format.

Examples

```
scf(0)
plot2d()
//PNG export
xs2png(0,'foo.png');
```

See Also

xs2bmp, xs2gif, xs2jpg, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

Authors

S.K

Name

xs2ppm — send graphics to a file in PPM syntax

```
xs2ppm(win_num,filen)
```

Parameters

win_num
: integer scalar or vector .

filen
: string, file name.

Description

`xs2ppm` sends the recorded graphics of the window `win_num` in the file `filen` in PPM format.

Examples

```
scf(0)
plot2d()
//PPM export
filename='foo.ppm';
xs2ppm(0,filename);
```

See Also

xs2bmp, xs2gif, xs2jpg, xs2png, xs2eps, xs2pdf, xs2svg, xs2ps, xs2fig, xs2emf

Name

xs2ps — send graphics to a file in PS syntax

```
xs2ps(win_num,filen,[orientation])
```

Parameters

win_num
: integer scalar or vector .

filen
: string, file name.

orientation
: optional character, with possible values 'p' (portrait) or 'l' (landscape). The default value is 'p'.

Description

`xs2ps` saves the recorded graphics of the window `win_num` to file `filen` in Postscript syntax. The `filen` must not have suffix extension.

Note that the generated Postscript file cannot be direcly printed since it requires a header file. The function `xs2eps` can be used to directly produce an encapsulated Postscript file with an header.

Examples

```
scf(0);
plot2d();
// Postcript export
filename='foo.ps';
xs2ps(0,filename);
```

See Also

set_posfig_dim, toprint, printfigure, xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2svg, xs2fig, xs2emf

Name

xs2svg — save graphics to a SVG file.

```
xs2svg(win_num,filen [,orientation])
```

Parameters

win_num
: integer scalar or vector .

filen
: string, file name.

orientation
: optional character, with possible values 'p' (portrait) or 'l' (landscape). The default value is 'p'.

Description

`xs2svg` saves the recorded graphics of the window `win_num` to file `filen` in SVG syntax. Note that `filen` must not have extension.

Examples

```
scf(0)
plot2d()
//SVG export
filename='foo.svg'
xs2svg(0,filename);
```

See Also

set_posfig_dim, toprint, printfigure, xs2bmp, xs2gif, xs2jpg, xs2png, xs2ppm, xs2eps, xs2pdf, xs2ps, xs2fig, xs2emf

Graphics Library

Name

GlobalProperty — to customize the objects appearance (curves, surfaces...) in a plot or surf command.

Description

The GlobalProperty is an optional argument that can be used inside a plot or surf command. It allows a global customization of all the new plotted lines (respectivly surfaces). It has to be given as a couple {PropertyName, PropertyValue}. Several couples can be set at the same time in a plot or surf call.

PropertyName must be a string defining the property to set.The PropertyValue can be a real, integer or string (scalar or matrix) depending on the type of property used. For example, to specify a red (color) longdash-dot (line style) with diamond marker (marker), the sequence should be : 'Colo','red','LineSt','-.','Marker','diam'.

As you can see, a full complete spelling of each property name and value is not required but those arguments, specified in any order, must remain unambiguous. Furthermore, the string specification is not case sensitive. GlobalProperty is predominant on all LineSpec previously stated.

Here is a complete list of the `PropertyName` you can specify (using plot or surf) and their available associated `PropertyValue`. If not specified, those properties are available for both `Polyline` and `Fac3d` objects (created respectivly by plot or surf) and, as previously said, they are applied to the new created objects (lines or surfaces).

Sometimes, you may have two PropertyName correponding to one property : the first one is the equivalent default Matlab name, the second is the default name used by Scilab (i.e.: `Color` or `Foreground` for a line, see below).

CData or ColorData:
: a real matrix specifiying the color at every points defined by `Z` matrix. This property is linked to the object's `data.color` property (see surface_properties). Note that this property is available for surfaces only.

CDataMapping or ColorDataMapping:
: a string with value `'scaled'` or `'direct'`. If a `data.color` is set, each index color data specifies a single value for each vertex. `cdata_mapping` determines wether those indices are scaled to map linearly into the current colormap (`'scaled'` mode) or point directly into this colormap (`'direct'` mode). This property is usefull when `color_flag` equals 2,3 or 4. Note that this property exists only with `Fac3d` entities. Note also that `plot3d` has `'direct'` mode by default and `surf` has `'scaled'` mode by default.

Clipping:
: a string `"on"` or `"off"` defining the clipping mode (`"on"` by default). It is equivalent to the `clip_state` property. This field contains the `visible` property (see polyline_properties). Note that this property is not yet available for surface entities.

Color or Foreground:
: a string defining a known color (see color_list) or a `1x3` (or `3x1`) RGB vector defining a color number. Color number is given as a 3-uple `R`, `G`, `B` corresponding respectively to red, green and blue intensity between 0 and 1. This property is linked to the object's `foreground` property (see polyline_properties). `Warning :` Color is not available for surfaces objects. The `Foreground` property exists for surfaces objects but is linked to the Matlab `EdgeColor` property (see surface_properties).

EdgeColor or Foreground:
: a string defining a known color (see color_list) or a `1x3` (or `3x1`) RGB vector defining a color number. Color number is given as a 3-uple `R`, `G`, `B` corresponding respectively to red, green and blue intensity between 0 and 1. This property is linked to the surface `foreground` property (see surface_properties). `Warning :` For polyline objects, the `Foreground` property exists with a different meaning (see above) and `EdgeColor` does not exist at all.

FaceColor:
a string with value `'none'`, `'flat'` or `'interp'` specifying the way the facet's color are rendered. When `'none'` is selected, a mesh of the surface is drawn; if `'flat'` (deault mode) is set, the `Fac3d color.data` values determine one color per facet using the color of the first vertex of the facet. If the value is `'interp'`, an interpolated shading is done on the surface using `color.data` to determine a color at each vertex of each facet.

LineStyle:
This property value should be a string defining a line style. This property is linked to tje object's `line_style` property (see polyline_properties or surface_properties).

Specifier	Line Style
-	Solid line (default)
--	Dashed line
:	Dotted line
-.	Dash-dotted line
none	No line

Marker or MarkStyle:
A string defining the marker type. Note that if you specify a marker wihtout a line style, both line (with default solid mode enabled) and marker are drawn.This property is linked to the object's `mark_style` and `mark_mode` properties (see polyline_properties or surface_properties).

Specifier	Marker Type
+	Plus sign
o	Circle
*	Asterisk
.	Point
x	Cross
'square' or 's'	Square
'diamond' or 'd'	Diamond
^	Upward-pointing triangle
v	Downward-pointing triangle
>	Right-pointing triangle
<	Left-pointing triangle
'pentagram'	Five-pointed star (pentagram)
'none'	No marker (default)

MarkerEdgeColor or MarkForeground:
a string defining a known color (see color_list) or a `1x3` (or `3x1`) RGB vector defining a color number. Color number is given as a 3-uple `R`, `G`, `B` corresponding respectively to red, green and blue intensity between 0 and 1. This property is linked to the object's `mark_foreground` property (see polyline_properties or surface_properties).

MarkerFaceColor or MarkBackground:
a string defining a known color (see color_list) or a `1x3` (or `3x1`) RGB vector defining a color number. Color number is given as a 3-uple `R`, `G`, `B` corresponding respectively to red, green and blue intensity between 0 and 1. This property is linked to the object's `mark_background` property (see polyline_properties or surface_properties).

MarkerSize or MarkSize:
: a scalar defining the marker size in `point` unit. This property is linked to the object's `mark_size` property with `mark_size_unit` enabled to `"point"` (see polyline_properties or surface_properties).

Visible:
: a string `"on"` or `"off"` defining the visibility mode (`"on"` by default). This property is linked to the object's `visible` property (see polyline_properties or surface_properties).

X data:
: a real vector or matrix (re-)defining the given data for all the plotted lines or surfaces. Concerning dimensions, note that this new data must match all the previous specified `X` data : that is to say all those data matrices must be of the same dimensions. This property is linked to the object's `data.x` property (see polyline_properties or surface_properties).

Y data:
: a real vector or matrix (re-)defining the given data for all the plotted lines or surfaces. Concerning dimensions, note that this new data must match all the previous specified `Y` data : that is to say all those data matrices must be of the same dimensions. This property is linked to the object's `data.y` property (see polyline_properties or surface_properties).

Z data:
: when used with `plot`, a real vector or matrix adding a

  `Z` data for all the plotted lines ; with `surf`, a real matrix (re-)defining the given data for all the surfaces. Concerning dimensions, note that this new data must match all the previous specified `X` and `Y` data. This property is linked to the object's `data.z` property (see polyline_properties or surface_properties).

Examples

```
// --------------------
// With the plot command :
// --------------------
x=1:10; // Init.
plot(x,sin(x),'colo','red','linest','-.','marker','>','markeredg','cyan','marke
clf();

// combinations' order in {PropertyName,PropertyValue} does not matter
plot(x,sin(x),'marker','p','markerfac','cyan','markersiz',10)
clf();

// combination of LineSpec and GlobalProperty shows the GlobalProperty predomina
plot(x,x.*x,'*cya--','color','gr','linestyle','-','marker','sq','markersize',6,
clf();

//multiple plots with different LineSpecs and finally some global GlobalPropert
clf();
t=0:%pi/20:2*%pi;
plot(t,sin(t),'ro-.',t,cos(t),'cya+',t,abs(sin(t)),'--mo','markstyl','diam')

// --------------------
// With the surf command :
// --------------------


Z= [   0.0001    0.0013    0.0053   -0.0299   -0.1809   -0.2465   -0.1100   -0.
    0.0005    0.0089    0.0259   -0.3673   -1.8670   -2.4736   -1.0866   -0.160
    0.0004    0.0214    0.1739   -0.3147   -4.0919   -6.4101   -2.7589   -0.277
```

```
   -0.0088   -0.0871    0.0364    1.8559    1.4995   -2.2171   -0.2729    0.836
   -0.0308   -0.4313   -1.7334   -0.1148    3.0731    0.4444    2.6145    2.441
   -0.0336   -0.4990   -2.3552   -2.1722    0.8856   -0.0531    2.6416    2.406
   -0.0137   -0.1967   -0.8083    0.2289    3.3983    3.1955    2.4338    1.212
   -0.0014   -0.0017    0.3189    2.7414    7.1622    7.1361    3.1242    0.663
    0.0002    0.0104    0.1733    1.0852    2.6741    2.6725    1.1119    0.197
    0.0000    0.0012    0.0183    0.1099    0.2684    0.2683    0.1107    0.019

clf();
f=gcf();
f.figure_size = [610,724];
subplot(211)
surf(Z,'facecol','interp','ydat',101:110,'edgecol','mage')
subplot(212)
surf(Z,'edgeco','b','marker','d','markersiz',9,'markerfac','k','xdata',-50:-41)
```

See Also

LineSpec , plot , surf , clf , polyline_properties , surface_properties

Authors

F.Leray

Name

Graphics — graphics library overview

2d plotting

plot2d
: plot a curve

plot2d2
: plot a curve as step function

plot2d3
: plot a curve with vertical bars

plot2d4
: plot a curve with arrows

fplot2d
: plot a curve defined by a function

champ
: 2D vector field

champ1
: 2D vector field with colored arrows

fchamp
: direction field of a 2D first order ODE

contour2d
: level curves of a surface on a 2D plot

fcontour2d
: level curves of a surface defined by a function on a 2D plot

grayplot
: 2D plot of a surface using colors

fgrayplot
: 2D plot of a surface defined by a function using colors

Sgrayplot
: smooth 2D plot of a surface using colors

Sfgrayplot
: smooth 2D plot of a surface defined by a function using colors

xgrid
: add a grid on a 2D plot

errbar
: add vertical error bars on a 2D plot

histplot
: plot a histogram

Matplot
: 2D plot of a matrix using colors

3d plotting

plot3d
: plot a surface

plot3d1
: plot a surface with gray or color level

fplot3d
: plot a surface defined by a function

fplot3d1
: plot a surface defined by a function with gray or color level

param3d
: plot one curve

param3d1
: plots curves

contour
: level curves on a 3D surface

fcontour
: level curves on a 3D surface defined by a function

hist3d
: 3D representation of a histogram

genfac3d
: compute facets of a 3D surface

eval3dp
: compute facets of a 3D surface

geom3d
: projection from 3D on 2D after a 3D plot

Line and polygon plotting

xpoly
: draw a polyline or a polygon

xpolys
: draw a set of polylines or polygons

xrpoly
: draw a regular polygon

xsegs
: draw unconnected segments

xfpoly
: fill a polygon

xfpolys
: fill a set of polygons

Rectangle plotting

xrect
: draw a rectangle

xfrect
: fill a rectangle

xrects
: draw or fill a set of rectangles

Arc plotting

xarc
: draw a part of an ellipse

xarcs
: draw parts of a set of ellipses

xfarc
: fill a part of an ellipse

xfarcs
: fill parts of a set of ellipses

Arrow plotting

xarrows
: draw a set of arrows

Strings

xstring
: draw strings

xstringl
: compute a box which surrounds strings

xstringb
: draw strings into a box

xtitle
: add titles on a graphics window

titlepage
: add a title in the middle of a graphics window

xinfo
: draw an info string in the message subwindow

Frames and axes

xaxis
: draw an axis

graduate
: pretty axis graduations

plotframe
: plot a frame with scaling and grids

Coordinates transformations

isoview
: set scales for isometric plot (do not change the size of the window)

square
: set scales for isometric plot (change the size of the window)

scaling
: affine transformation of a set of points

rotate
: rotation of a set of points

xsetech
: set the sub-window of a graphics window for plotting

subplot
: divide a graphics window into a matrix of sub-windows

xgetech
: get the current graphics scale

xchange
: transform real to pixel coordinates

Colors

colormap
: using colormaps

getcolor
: dialog to select colors in the current colormap

addcolor
: add new colors to the current colormap

graycolormap
: linear gray colormap

hotcolormap
: red to yellow colormap

Graphics context

xset
: set values of the graphics context

xget
: get current values of the graphics context

xlfont
: load a font in the graphics context or query loaded font

getsymbol
: dialog to select a symbol and its size

Save and load

xsave
: save graphics into a file

xload
: load a saved graphics

xs2fig
send graphics to a file in Xfig syntax

xs2gif
send graphics to a file in Gif syntax

xs2ppm
send graphics to a file in PPM syntax

Graphics primitives

xbasc
clear a graphics window and erase the associated recorded graphics

xclear
clear a graphics window

driver
select a graphics driver

xinit
initialisation of a graphics driver

xend
close a graphics session

xbasr
redraw a graphics window

replot
redraw the current graphics window with new boundaries

xpause
suspend Scilab

xselect
raise the current graphics window

xdel
delete a graphics window

winsid
return the list of graphics windows

xname
change the name of the current graphics window

Mouse position

xclick
wait for a mouse click

locate
mouse selection of a set of points

xgetmouse
get the current position of the mouse

Interactive editor

edit_curv
interactive graphics curve editor

gr_menu
: simple interactives graphic editor

sd2sci
: gr_menu structure to scilab instruction convertor

Graphics functions for automatic control

bode
: Bode plot

gainplot
: magnitude plot

nyquist
: Nyquist plot

m_circle
: M-circle plot

chart
: Nichols chart

black
: Black's diagram

evans
: Evans root locus

sgrid
: s-plane grid lines

plzr
: pole-zero plot

zgrid
: zgrid plot

Name

LineSpec — to quickly customize the lines appearance in a plot

Description

The LineSpec is an optional argument that can be used inside a plot command to customize each new line aspect. It has to be given as a concatenated string containing information about color, line style or markers. It is very usefull to quickly specify such basic line properties.

To specify a red longdash-dot with diamond marker, the string can be 'r-.diam'. As you can see, a full complete spelling of each property value is not required but the string, which is a concatenation (in any order) of these three types of properties , must remain unambiguous. Furthermore, the string specification is not case sensitive.

Here is a complete list of the `LineSpec` types you can specify (using plot).

LineStyle:

a string defining the line style. This property is linked to the object's `line_style` property (see polyline_properties).

`Specifier`	`Line Style`
`-`	Solid line (default)
`--`	Dashed line
`:`	Dotted line
`-.`	Dash-dotted line

Color:

a string defining the line color. This property is linked to the object's `foreground` property (see polyline_properties).

`Specifier`	`Color`
`r`	Red
`g`	Green
`b`	Blue
`c`	Cyan
`m`	Magenta
`y`	Yellow
`k`	Black
`w`	White

A default color table is used to color plotted curves if you do not specify a color (neither with `LineSpec` nor with GlobalProperty). When drawing multiple lines, the plot command automatically cycles through this table. Here are the used colors:

`R`	`G`	`B`
0.	0.	1.
0.	0.5	0.
1.	0.	0.

0.	0.75	0.75
0.75	0.	0.75
0.75	0.75	0.
0.25	0.25	0.25

Marker type:

A string defining the marker type. note that if you specify a marker wihtout a line style, only the marker is drawn. This property is linked to the object's `mark_style` and `mark_mode` properties (see polyline_properties).

`Specifier`	`Marker Type`
`+`	Plus sign
`o`	Circle
`*`	Asterisk
`.`	Point
`x`	Cross
`'square'` or `'s'`	Square
`'diamond'` or `'d'`	Diamond
`^`	Upward-pointing triangle
`v`	Downward-pointing triangle
`>`	Right-pointing triangle
`<`	Left-pointing triangle
`'pentagram'`	Five-pointed star (pentagram)
`'none'`	No marker (default)

Examples

```
x=1:0.1:10; // Init.
plot(x,sin(x),'r.->') // plots a dash-dotted line with a right-pointing triangl
clf();

// If you specify a marker wihtout a line style, only the marker is drawn
plot(x,sin(x),'d') // plots a dash-dotted line with a right-pointing triangle c

x=1:10; // Init.
// combinations' order does not matter
plot(x,x.*x,'*cya--')

//multiple plots with different LineSpecs
clf();
t=0:%pi/20:2*%pi;
plot(t,sin(t),'ro-.',t,cos(t),'cya+',t,abs(sin(t)),'--mo')
```

See Also

GlobalProperty , plot , clf

Authors

F.Leray

Name

Matplot — 2D plot of a matrix using colors

```
Matplot(a,[strf,rect,nax])
Matplot(a,<opt_args>)
```

Parameters

a
: real matrix of size (n1,n2).

<opt_args>
: This represents a sequence of statements `key1=value1, key2=value2`,... where `key1`, `key2,...` can be one of the following:

  rect
  : sets the bounds of the plot. If this key is given and neither `frameflag` nor `strf` is given then the `y` character of `strf` is supposed to be `7`. See below for value.

  nax
  : sets the grids definition. If this key is given and neither `axesflag` nor `strf` is given then the `z` character of `strf` is supposed to be `1`. See below for value.

  frameflag
  : specifies how the frame of the plot is computed. The value is an integer ranging from 0 to 8. It corresponds to the `y` character of `strf`. See below.

  axesflag
  : specifies what kind of axes are drawn around the plot. The value is an integer ranging from 0 to 5. It corresponds to the `z` character of `strf`. See below.

strf
: is a string of length 3 `"xyz"`.

  default
  : The default is `"081"`.

  x
  : controls the display of captions.

    x=0
    : no caption.

    x=1
    : captions are displayed. They are given by the optional argument `leg`.

  y
  : controls the computation of the actual coordinate ranges from the minimal requested values. Actual ranges can be larger than minimal requirements.

    y=0
    : no computation, the plot use the previus (or default) scale

    y=1
    : from the rect arg

    y=2
    : from the min/max of the x, y datas

y=3
: built for an isometric scale from the rect arg

y=4
: built for an isometric plot from the min/max of the x, y datas

y=5
: enlarged for pretty axes from the rect arg

y=6
: enlarged for pretty axes from the min/max of the x, y datas

y=7
: like y=1 but the previus plot(s) are redrawn to use the new scale

y=8
: like y=2 but the previus plot(s) are redrawn to use the new scale

z
: controls the display of information on the frame around the plot. If axes are requested, the number of tics can be specified by the `nax` optional argument.

z=0
: nothing is drawn around the plot.

z=1
: axes are drawn, the y=axis is displayed on the left.

z=2
: the plot is surrounded by a box without tics.

z=3
: axes are drawn, the y=axis is displayed on the right.

z=4
: axes are drawn centred in the middle of the frame box.

z=5
: axes are drawn so as to cross at point `(0,0)`. If point `(0,0)` does not lie inside the frame, axes will not appear on the graph.

rect
: This argument is used when the second character y of argument `strf` is 1, 3 or 5. It is a row vector of size 4 and gives the dimension of the frame: `rect=[xmin,ymin,xmax,ymax]`.

nax
: This argument is used when the third character z of argument `strf` is 1. It is a row vector with four entries `[nx,Nx,ny,Ny]` where `nx` (`ny`) is the number of subgraduations on the x (y) axis and `Nx` (`Ny`) is the number of graduations on the x (y) axis.

Description

The entries of matrix `int(a)` are used as colormap entries in the current colormap. The color associated to `a(i,j)` is used do draw a small square of size 1 with center at location `(x=j,y=(n1-i+1))`. If a matrix entry is outside the colormap, the corresponding rectangle is not displayed.

Enter the command `Matplot()` to see a demo.

Examples

```
Matplot([1 2 3;4 5 6])
clf()
// draw the current colormap
Matplot((1:xget("lastpattern")))
```

See Also

colormap , plot2d , Matplot1 , Matplot_properties

Authors

J.Ph.C.

Name

Matplot1 — 2D plot of a matrix using colors

```
Matplot1(a,rect)
```

Parameters

a
: real matrix of size (n1,n2).

rect
: `[xmin,ymin,xmax,ymax]`

Description

The entries of matrix `int(a)` are used as colormap entries in the current colormap. `rect` specify a rectangle in the current scale and the matrix is drawn inside this rectangle. Each matrix entry will be rendered as a small rectangle filled with its associated color. If a matrix entry is outside the colormap, the corresponding rectangle is not displayed.

Examples

```
//--- first example
clf();
ax=gca();//get current axes handle
ax.data_bounds=[0,0;10,10];//set the data_bounds
ax.box='on'; //draw a box
a=5*ones(11,11); a(2:10,2:10)=4; a(5:7,5:7)=2;
// first matrix in rectangle [1,1,3,3]
Matplot1(a,[1,1,3,3])
a=ones(10,10); a= 3*tril(a)+ 2*a;
// second matrix in rectangle [5,6,7,8]
Matplot1(a,[5,6,7,8])

//--- second example  (animation)
n=100;


clf();
f=gcf();//get current figure handle
f.pixmap='on';//double buffer mode
ax=gca();//get current axes handle
ax.data_bounds=[0,0;10,10];//set the data_bounds
ax.box='on'; //draw a box
show_pixmap()
for k=-n:n,
  a=ones(n,n);
  a= 3*tril(a,k)+ 2*a;
  a= a + a';
  k1= 3*(k+100)/200;
  if k>-n then delete(gce()),end
  Matplot1(a,[k1,2,k1+7,9])
  show_pixmap() //send double buffer to screen
end
```

See Also

colormap , plot2d , Matplot , grayplot , Matplot_properties

Authors

J.Ph.C.

Name

Matplot_properties — description of the Matplot entities properties

Description

The Matplot entity is a leaf of the graphics entities hierarchy. It represents 2D plots of surface using colors and images (see `Matplot` and `Matplot1`).

parent:
: This property contains the handle of the parent. The parent of the Matplot entity should be of the type `"Axes"`.

children:
: This property contains a vector with the `children` of the handle. However, `Matplot` handles currently do not have any `children`.

visible:
: This field contains the `visible` property value for the entity . It should be `"on"` or `"off"` . By default, the plot is visible, the value's property is `"on"`. If `"off"` the plot is not drawn on the screen.

data:
: This field defines a [`m`x`n`] `color` data matrix using the current colormap. The color associated to `color(i,j)` is used do draw a small square of length 1 with center at location `(x=j,y=(m-i+1))`.

clip_state:
: This field contains the `clip_state` property value for the Matplot. It should be :

  - `"off"` this means that the Matplot is not clipped.
  - `"clipgrf"` this means that the Matplot is clipped outside the Axes box.
  - `"on"` this means that the Matplot is clipped outside the rectangle given by property clip_box.

clip_box:
: This field is to determinate the `clip_box` property. By Default its value should be an empty matrix if clip_state is "off". Other cases the vector `[x,y,w,h]` (upper-left point width height) defines the portions of the Matplot to display, however `clip_state` property value will be changed.

user_data:
: This field can be use to store any scilab variable in the Matplot data structure, and to retreive it.

Examples

```
Matplot((1:xget("lastpattern")))
e=gce(); // get current entity

e.data=e.data($:-1:1) // reverse order
```

See Also

set , get , delete , grayplot , Matplot , Matplot1 , graphics_entities , grayplot_properties

Authors

F.Leray

Name

Sfgrayplot — smooth 2D plot of a surface defined by a function using colors

```
Sfgrayplot(x,y,f,<opt_args>)
Sfgrayplot(x,y,f [,strf, rect, nax, zminmax, colminmax, mesh, colout])
```

Parameters

x,y
: real row vectors of size n1 and n2.

f
: scilab function (z=f(x,y))

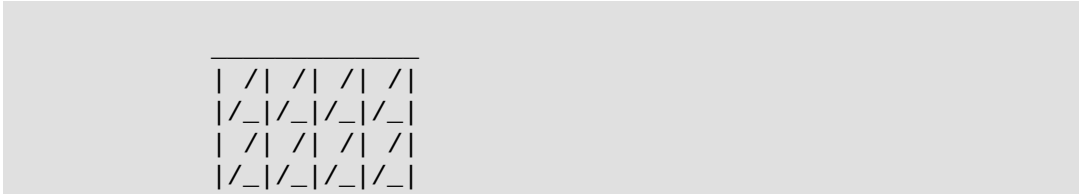
<opt_args>
: This represents a sequence of statements `key1=value1, key2=value2`,... where `key1`, `key2,...` can be one of the following: strf, rect, nax, zminmax, colminmax, mesh, colout (see plot2d for the 3 first and fec for the 4 last).

strf,rect,nax
: see plot2d.

zminmax, colminmax, mesh, colout
: see fec.

Description

`Sfgrayplot` is the same as `fgrayplot` but the plot is smoothed. The function `fec` is used for smoothing. The surface is plotted assuming that it is linear on a set of triangles built from the grid (here with n1=5, n2=3):



The function colorbar may be used to see the color scale (but you must know (or compute) the min and max values).

Instead of Sfgrayplot, you can use Sgrayplot and this may be a little faster.

Enter the command `Sfgrayplot()` to see a demo.

Examples

```
// example #1: plot 4 surfaces
function z=surf1(x,y), z=x*y, endfunction
function z=surf2(x,y), z=x^2-y^2, endfunction
function z=surf3(x,y), z=x^3+y^2, endfunction
function z=surf4(x,y), z=x^2+y^2, endfunction
xbasc()
xset("colormap",[jetcolormap(64);hotcolormap(64)])
x = linspace(-1,1,60);
y = linspace(-1,1,60);
drawlater() ;
```

```
subplot(2,2,1)
   colorbar(-1,1,[1,64])
   Sfgrayplot(x,y,surf1,strf="041",colminmax=[1,64])
   xtitle("f(x,y) = x*y")
subplot(2,2,2)
   colorbar(-1,1,[65,128])
   Sfgrayplot(x,y,surf2,strf="041",colminmax=[65,128])
   xtitle("f(x,y) = x^2-y^2")
subplot(2,2,3)
   colorbar(-1,2,[65,128])
   Sfgrayplot(x,y,surf3,strf="041",colminmax=[65,128])
   xtitle("f(x,y) = x^3+y^2")
subplot(2,2,4)
   colorbar(0,2,[1,64])
   Sfgrayplot(x,y,surf4,strf="041",colminmax=[1,64])
   xtitle("f(x,y) = x^2+y^2")
drawnow() ;
xselect()

// example #2: plot surf3 and add some contour lines
function z=surf3(x,y), z=x^3+y^2, endfunction
xbasc()
x = linspace(-1,1,60);
y = linspace(-1,1,60);
xset("colormap",hotcolormap(128))
drawlater() ;
colorbar(-1,2)
Sfgrayplot(x,y,surf3,strf="041")
fcontour2d(x,y,surf3,[-0.1, 0.025, 0.4],style=[1 1 1],strf="000")
xtitle("f(x,y) = x^3+y^2")
drawnow() ;
xselect()

// example #3: plot surf3 and use zminmax and colout optional arguments
//             to restrict the plot for -0.5<= z <= 1
function z=surf3(x,y), z=x^3+y^2, endfunction
xbasc()
x = linspace(-1,1,60);
y = linspace(-1,1,60);
xset("colormap",jetcolormap(128))
drawlater() ;
zminmax = [-0.5 1]; colors=[32 96];
colorbar(zminmax(1),zminmax(2),colors)
Sfgrayplot(x, y, surf3, strf="041", zminmax=zminmax, colout=[0 0], colminmax=co
fcontour2d(x,y,surf3,[-0.5, 1],style=[1 1 1],strf="000")
xtitle("f(x,y) = x^3+y^2, with parts under z = -0.5 and upper z = 1 removed")
drawnow() ;
xselect()
```

See Also

fec , fgrayplot , grayplot , Sgrayplot

Authors

J.Ph.C.

Name

Sgrayplot — smooth 2D plot of a surface using colors

```
Sgrayplot(x,y,z,<opt_args>)
Sgrayplot(x,y,z [,strf, rect, nax, zminmax, colminmax, mesh, colout])
```

Parameters

x,y
: real row vectors of size n1 and n2.

z
: real matrix of size (n1,n2). `z(i,j)` is the value of the surface at the point (x(i),y(j)).

<opt_args>
: This represents a sequence of statements `key1=value1,key2=value2`,... where `key1`, `key2,...` can be one of the following: strf, rect, nax, zminmax, colminmax, mesh, colout.

strf
: is a string of length 3 `"xyz"` (by default `strf= "081"`)

  x
  : controls the display of captions.

    x=0
    : no caption.

    x=1
    : captions are displayed. They are given by the optional argument `leg`.

  y
  : controls the computation of the actual coordinate ranges from the minimal requested values. Actual ranges can be larger than minimal requirements.

    y=0
    : no computation, the plot use the previus (or default) scale

    y=1
    : from the rect arg

    y=2
    : from the min/max of the x, y datas

    y=3
    : built for an isometric scale from the rect arg

    y=4
    : built for an isometric plot from the min/max of the x, y datas

    y=5
    : enlarged for pretty axes from the rect arg

    y=6
    : enlarged for pretty axes from the min/max of the x, y datas

    y=7
    : like y=1 but the previus plot(s) are redrawn to use the new scale

    y=8
    : like y=2 but the previus plot(s) are redrawn to use the new scale

z
: controls the display of information on the frame around the plot. If axes are requested, the number of tics can be specified by the `nax` optional argument.

  z=0
  : nothing is drawn around the plot.

  z=1
  : axes are drawn, the y=axis is displayed on the left.

  z=2
  : the plot is surrounded by a box without tics.

  z=3
  : axes are drawn, the y=axis is displayed on the right.

  z=4
  : axes are drawn centred in the middle of the frame box.

  z=5
  : axes are drawn so as to cross at point `(0,0)`. If point `(0,0)` does not lie inside the frame, axes will not appear on the graph.

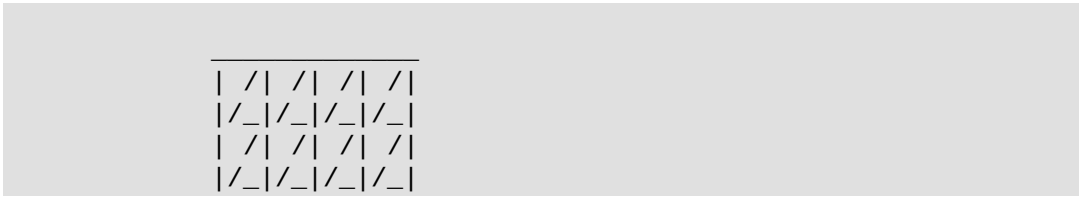
rect
: This argument is used when the second character y of argument `strf` is 1, 3 or 5. It is a row vector of size 4 and gives the dimension of the frame: `rect=[xmin,ymin,xmax,ymax]`.

nax
: This argument is used when the third character z of argument `strf` is 1. It is a row vector with four entries `[nx,Nx,ny,Ny]` where `nx` (`ny`) is the number of subgraduations on the x (y) axis and `Nx` (`Ny`) is the number of graduations on the x (y) axis.

zminmax, colminmax, mesh, colout
: See fec.

Description

`Sgrayplot` is the same as `grayplot` but the plot is smoothed. The function `fec` is used for smoothing. The surface is plotted assuming that it is linear on a set of triangles built from the grid (here with n1=5, n2=3):



The function colorbar may be used to see the color scale.

The parameter `zminmax` is useful for animation purpose (see an example after) and the parameter `colminmax` lets the user choose a part of the current colormap (see the fec help page).

Enter the command `Sgrayplot()` to see a demo.

Examples

```
// example #1
x=-10:10; y=-10:10;m =rand(21,21);
```

```
clf()
xset("colormap",hotcolormap(64))
Sgrayplot(x,y,m, strf="011", rect=[-20,-20,20,20])

// example #2
t=-%pi:0.1:%pi; m=sin(t)'*cos(t);
clf()
xset("colormap",jetcolormap(64))
colorbar(-1,1)
Sgrayplot(t,t,m, strf="041")

// example #3: an animation display cos(t)*sin(x)sin(y).
n = 30;
nt = 100;
x = linspace(0,2*%pi,n);
y = linspace(0,%pi,n/2);
z = sin(x')*sin(y);
t = linspace(0,4*%pi,nt);
xselect(); clf()
f=gcf();
f.color_map=jetcolormap(64);
f.pixmap='on';
colorbar(-1,1)
Sgrayplot(x,y,cos(t(1))*z, strf="042", zminmax=[-1,1])
c=gce(),e=c.children
xtitle("Kaa''s eyes")
for i = 1:nt
   e.data(:,3)=matrix(cos(t(i))*z,-1,1);
   show_pixmap()
end
f.pixmap='off';
```

See Also

fec , fgrayplot , grayplot , Sfgrayplot , colorbar

Authors

J.Ph.C.

Name

addcolor — add new colors to the current colormap

```
new=addcolor(c)
```

Parameters

new
: ids of the colors defined in c in the new color table.

c
: matrix with 3 columns, RGB color definition.

Description

`addcolor` adds new colors given in the `c` argument to the current colormap. `c` must be a matrix with 3 columns `[R G B]R` is red component, `G` is green component, `B` is blue component). Each entry in `c` must be a non negative number less or equal to 1.

The ids of the new colors are returned into `new`.

If a color defined in `c` is already present in the current colormap it is not added.

See Also

colormap

Name

alufunctions — pixel drawing functions

Description

src is the source ie the "value of the pixel" which we want to draw. dst is the destination ie "value of the pixel" which is already drawn.

0
: clear ie "0"

1
: and ie "src AND dst"

2
: and reverse ie "src AND NOT dst"

3
: copy ie "src"

4
: and inverted ie "(NOT src) AND dst"

5
: noop ie "dst"

6
: xor ie "src XOR dst"

7
: or ie "src OR dst"

8
: nor ie "(NOT src) AND (NOT dst)"

9
: equiv ie "(NOT src) XOR dst"

10
: invert ie "NOT dst"

11
: or reverse ie "src OR (NOT dst)"

12
: copy inverted ie "NOT src"

13
: or inverted ie "(NOT src) OR dst"

14
: nand ie "(NOT src) OR (NOT dst)"

15
: set ie "1"

Name

arc_properties — description of the Arc entity properties

Description

The Arc entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for ellipses and part of ellipses and the filled ones.

parent:
This field contains the handle of the parent. The parent of the arc entity should be of the type `"Axes"` or `"Compound"`.

children:
This property contains a vector with the `children` of the handle. However, `arc` handles currently do not have any `children`.

thickness:
This field contains the line `thickness` property. Its value should be positive integer.

line_style:
The `line_style` property value should be an integer in [1 6]. 1 stands for solid the other value stands for a selection of dashes.

line_mode:
This property allows to display or not the line representing the arc. The value must be `"on"` or `"off"`.

fill_mode:
If `fill_mode` property value is `"on"` , the arc is filled with the `background` color.

foreground:
This field contains the default `foreground` property value used to draw the outside of the arc. It should be a color index (relative to the current colormap).

background:
This field contains the color used to fill the arc. It should be a color index (relative to the current colormap).

data:
This property is to return the coordinates of the upper-left point, the width and the height of the inclosing rectangle as well as the boundary angles of the sector. It is the matrix in user coordinates `[xleft,yup,[zup],width,height,a1,a2]` where `a1` and `a2` are the sector boundary angles in degree.

Warning: in Scilab versions up to 4.1.2 `a1` and `a2` were given in degree/64.

visible:
This field contains the `visible` property value for the entity . It should be `"on"` or `"off"`. If `"on"` the arc is drawn, If `"off"` the arc is not displayed on the screen.

arc_drawing_method:
This field controls the kind of discretisation used to render the arc. Its value must be either `"nurbs"` or `"lines"`. If `"nurbs"` is selected then the arc is rendered using nurbs curves and surfaces. This results in the display of a perfect ellipse part whatever the view point is. If `"lines"` is selected then the arc is approximated with a constant number of lines. This reduce drawing time but some sharp edges may appear upon zooming. The use of `"lines"` value is discouraged and should only be used if a loss in framerate is noticed when using `"nurbs"` value.

clip_state:
This field contains the `clip_state` property value for the arc. Clip_state value should be :

- `"off"` this means that the arc is not clipped
- `"clipgrf"` this means that the arc is clipped outside the Axes box.
- `"on"` this means that the arc is clipped outside the arc given by property clip_box.

clip_box:
This field is to determinate the `clip_box` property. By Default its value should be an empty matrix if clip_state is "off". Other cases the vector `[x,y,w,h]` (upper-left point width height) defines the portions of the arc to display, however `clip_state` property value will be changed.

user_data:
This field can be use to store any scilab variable in the arc data structure, and to retreive it.

Examples

```
a=get("current_axes");//get the handle of the newly created axes
a.data_bounds=[-2,-2;2,2];

xarc(-1.5,1.5,3,3,0,360*64)

arc=get("hdl"); //get handle on current entity (here the arc entity)
arc.fill_mode="on";
arc.foreground=5;
arc.data(:,[3 6])=[2 270*64];
xfarc(-.5,1,.4,.6,0,360*64);
arc.visible="off";
```

See Also

set, get, delete, xarc, xarcs, xfarc, xfarcs, graphics_entities

Authors

Djalel ABDEMOUCHE

Jean-Baptiste SILVY

Name

autumncolormap — red through orange to yellow colormap

```
cmap=autumncolormap(n)
```

Parameters

n
: integer >= 3, the colormap size.

cmap
: matrix with 3 columns `[R,G,B]`.

Description

`autumncolormap` computes a colormap with `n` colors varying from red through orange to yellow.

Examples

```
f = scf();
plot3d1();
f.color_map = autumncolormap(32);
```

See Also

colormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

axes_properties — description of the axes entity properties

Description

The Axes entity is the second level of the graphics entities hierarchy. This entity defines the parameters allowing the change of coordinates and the axes drawing as well as the parameters' default values for the children creation.

Axes properties

parent:
: This field contains the handle of the parent figure.

children:
: FA vector containing the handles of all graphics objects children of the axes These graphics objects are of type `"Compound"`, `"Rectangle"`, `"Polyline"`, `"Segs"`, `"Arc"`, `"Grayplot"`,.. (see Compound_properties, rectangle_properties, champ_properties, axis_properties, polyline_properties, segs_properties, grayplot_properties, surface_properties, param3d_properties, fec_properties, text_properties, legend_properties)

visible:
: This field contains the `visible` property value for axes . Its value should be `"on"` or `"off"` . By default, axes is visible `"on"` in case all "visible" chidren are displayed on the screen, If `"off"` the axes and all its chidren are not drawn.

axes_visible:
: A `1x3` string vector. This property specifies whether each axis is drawn or not. Its value should be `"on"` or `"off"` for a global setting.To act on a single axis, the syntax is `axes_visible(N)` where `N` is `1,2` or `3` corresponding to the `x,y` or `z` axis. The scaling data and if required the grids are drawn if the value is `"on"`. Note that when creating a simple axes entity using the `gca()` (shorcut for `get"current_axes")`) or `gcf()` (shortcut for `get(current_figure)`) commands, the axes visiblibilty is set to `"off"`.

axes_reverse:
: A `1x3` string vector corresponding to the three axes (X,Y,Z). For each axes, the property specifies the direction of the incresing values. If `"off"`, the default direction is used. If `"on"`, the direction is reverse. It is also possible to use only one string, `"on"` or `"off"`, to set simultaneously the three data.

grid:
: The field value is a vector `[x-grid,y-grid,z-grid]` where `x-grid` controls a grid drawning for the x-axis and `y-grid, z-grid` respecting to the y-axis, z-axis. The default values is `-1` grids are not drawn, else the grids are drawn using the color given indexed by the grid value.

grid_position:
: This character string specifies the grid position compared with other graphic entities. Its value can be either `"foreground"` to draw the grid ahead other graphic entities or `"background"` to draw the grid behind.

x_location:
: Specify the location of the x-axis. The possible values are:

  - `"bottom"`. In this case the x axis is drawn at the bottom of the axes rectangle.
  - `"top"`. In this case the x axis is drawn at the top of the axes rectangle.
  - `"middle"` . In this case the x axis is drawn at the position nearest to the 0 y coordinates.

y_location:
Specify the location of the y-axis. The possible values are:

- `"left"`. In this case the y axis is drawn at the left of the axes rectangle.
- `"right"`. In this case the y axis is drawn at the right of the axes rectangle.
- `"middle"`. In this case the y axis is drawn at the position nearest to the 0 x coordinates.

title:
An object attached to the Axes entity and returning a graphic handle on a Label structure (see label_properties). This field defines a title with options on this label.

x_label:
An object attached to the Axes entity and returning a graphic handle on a Label structure (see label_properties). This field defines a label on x axis with options on this label.

y_label:
An object attached to the Axes entity and returning a graphic handle on a Label structure (see label_properties). This field defines a label on y axis with options on this label.

z_label:
An object attached to the Axes entity and returning a graphic handle on a Label structure (see label_properties). This field defines a label on z axis with options on this label.

auto_ticks:
A `1x3` string vector giving the auto_ticks status for each axis. This property specifies whether each axis is graduated using a computational algorithm or not (graduations are set by the user). Its value should be `"on"` or `"off"` for a global setting.To act on a single axis, the syntax is `auto_ticks(N)` where `N` is `1,2` or `3` corresponding to the `x,y` or `z` axis. Note that editing ticks (text and/or locations) via `x_ticks, y_ticks` or `z_ticks` automatically set `auto_ticks` to `"off"` for the corresponding axes.

x_ticks.locations:
A real vector containing the locations for the graduations on x axis. This property can be edited specifying a new real vector (of the same size). To specify greater or lesser graduations, man can act on the `x_ticks` `tlist` defining a corresponding `x_ticks.labels` string vector too.

y_ticks.locations:
A real vector containing the locations for the graduations on y axis. This property can be edited specifying a new real vector (of the same size). To specify greater or lesser graduations, man can act on the `y_ticks` `tlist` defining a corresponding `y_ticks.labels` string vector too.

z_ticks.locations:
A real vector containing the locations for the graduations on z axis. This property can be edited specifying a new real vector (of the same size). To specify greater or lesser graduations, man can act on the `z_ticks` `tlist` defining a corresponding `z_ticks.labels` string vector too.

x_ticks.labels:
A string vector containing the labels for the graduations on x axis. This property can be edited specifying a new string vector (of the same size). To specify greater or lesser graduations, man can act on the `x_ticks` `tlist` defining a corresponding `x_ticks.locations` real vector too.

y_ticks.labels:
A string vector containing the labels for the graduations on y axis. This property can be edited specifying a new string vector (of the same size). To specify greater or lesser graduations,

man can act on the `y_ticks` `tlist` defining a corresponding `y_ticks.locations` real vector too.

z_ticks.labels:
A string vector containing the labels for the graduations on z axis. This property can be edited specifying a new string vector (of the same size). To specify greater or lesser graduations, man can act on the `z_ticks` `tlist` defining a corresponding `z_ticks.locations` real vector too.

box:
This property specifies whether to enclose the axes in a box. Its value can be either `"off"`, `"hidden_axes"`, `"back_half"` or `"on".` If the property is `"off"`, the box is not draw. If the property is `"hidden_axes"`, only the back frame is drawn. If the property is `"back_half"`, the X, Y and Z axis are also drawn. If the property is `"on"` the whole box is drawn.

filled:
This property specifies whether the axes background should be drawn or not. Its value can be either `"off"` or `"on".` If the property is `"off"`, the background is not drawn, the axes box is transparent. If the property is `"on"` the background is drawn using the color specified by the `background` property.

sub_ticks:
This field sets the number of tics to draw between two main tics. The field value is the vector `[nx,ny]` where `nx` is the number of sub tics for the x-axis and `ny` respecting to the y-axis.

font_style:
Specifies the font used for displaying tics labels. This is a positive integer referecing one of the loaded fonts. Its value must be between 0, referecing the first font, and the number of loaded fonts minus one, referencing the last font. For more information see graphics_fonts.

font_size:
It is a scalar specifying the character size of tics labels. If `fractional_font` property is `"off"` only the integer part of the value is used. For more information see graphics_fonts.

font_color:
This property determines the color of the tics labels.

fractional_font:
This property specify whether ticks labels are displayed using fractional font sizes. Its value must be either `"on"` or `"off"`. If `"on"` the floating point value of `font_size` is used for display and the font is anti-aliased. If `"off"` only the integer part is used and the font is not smoothed.

isoview:
This property is used to have isometric scales on the x, y and z axes (for exemple to make the display of the curve sin(x) versus cos(x) be a circle not an ellipse). Its value should be `"on"` or `"off"`. If the value is `"on"`, the axes `data_bounds` automatically change according to the corresponding figure `figure_size` property values.

cube_scaling:
This property is used in `3d mode` to have a rescaling of the x, y and z axes. Indeed, it allows the `data` to fit into a 1x1x1 cube ; the goal is to better display 3d graphics in case axes scales are very different from one to another. Its value should be `"on"` or `"off"` (which is the default value). In most cases, it helps generating Matlab-like 3d view.

view:
This field is related to the graphics universe. It takes `"3d"` as value corresponding to the three-dimensional views. In the other case its value can be `"2d"` for initial 2d plotting (default

value). This flag also depends on the plots the user enters : a `plot3d` command, for example, will switch the `view` flag from `"2d"` to `"3d"`.

rotation_angles:
This field is the vector `[alpha,theta]`. These two values give the spherical coordinates of the observation points (in degree).

log_flags:
3 character string that sets the scale (linear or logarithmic) along the axes. Each character speficfies the scale for respectivgly the X, Y and Z axes. They should take a value between `'n'` for linear scale or `'l'` for logarithmic scale.

tight_limits:
If this property value is `"on"` axes adapt to fit exactly with the minima and maxima values of the `data_bounds`. If this field value is `"off"`, axes may enlarge boundaries such as to produce pretty tics labels.

data_bounds:
This field contains the boundary values for the x,y and z coordinates. It is the matrix `[xmin,ymin,zmin;xmax,ymax,zmax]` or `[xmin,ymin;xmax,ymax]`. Note that, to stricly have the specified data bounds, `tight_limits` must be set to `"on"` value (see above).

zoom_box:
This field contains the current zoom box if any coordinates are given. It is an empty matrix (no zoom) or the vector `[xmin,ymin,xmax,ymax,zmin,zmax]` (defines a smaller axes box).

margins:
A vector `[margin_left,margin_right,margin_top,margin_bottom]` specifying the margins portion for this axes. This vector is composed of numbers between [0 1] with default: [0.125 0.125 0.125 0.125] (these numbers are ratio relatives to the corresponding figure `figure_size` property values).

axes_bounds:
A vector `[x_left,y_up,width,height]` specifying the portion of figure used by this axes. Where `x_left`, `y_up`, `width` and `height` are numbers in [0 1] give respectively the position of the upper-left corner and the dimension of the axes (these numbers are ratio relative to the corresponding figure `figure_size` property values).

hidden_axis_color:
This property defined the color of the hidden axis. It takes an index relative to the current colormap.

user_data:
This field can be use to store any scilab variable in the axes data structure, and to retreive it.

Properties for high level functions
The `plot`, `plot2dx`, `grayplot` and `matplot` functions use the following properties to decide how to merge consecutive plots if this is not stated by the `frameflag` calling argument. The result of the merge is decided through these two following properties:

auto_clear:
If this property value is equal to `"on"`, a call to a high level graphic will re-ininitialize the current axes and erase all its children before preforming the drawing. If the value is `"off"` the drawings will be added to current axes according to `"auto_scale"` property.

auto_scale:
A property to update the axes data boundary. If value is `"on"`, a new plot will adapt the current axes properties to fit with previous and current plots. If its value is `"off"` the new plot will be drawn in the current axes data boundary.

Children's default values:

hiddencolor:
This property controls the hidden parts' color. It takes as value an index relative to the current colormap. In another case, if it is a negative value, the hidden parts take the same colors as the surface

line_mode:
This field contains the default `line_mode` property value for Segs Rectangle Legend Axis Plot3d Fac3d and Polyline objects. Its value should be `"on"` (default) or `"off"`.

line_style:
This field contains the default `line_style` property value for Segs, Arcs, Rectangle and Polyline objects. `line_style` selects the type of line to be used to draw lines . Its value should be an integer in [0 6]. 0 and 1 stand for solid, the other values stand for a selection of dashes (see getlinestyle).

thickness:
This field contains the default `thickness` property value for all objects using line drawing. Its value should be positive integer.

mark_mode:
This field contains the default `mark_mode` property value for Segs Rectangle Legend Axis Plot3d Fac3d and Polyline objects. Its value should be `"on"` or `"off"` (default).

mark_style:
This field contains the default `mark_style` property value for Segs Rectangle Legend Axis Plot3d Fac3d and Polyline objects. `mark_style` selects the type of mark to be displayed. Its value should be an integer in [0 9] which stands for: dot, plus, cross, star, filled diamond, diamond, triangle up, triangle down, trefle and circle.

mark_size_unit:
This field contains the default `mark_size_unit` property value for Segs Rectangle Legend Axis Plot3d Fac3d and Polyline objects. If `mark_size_unit` is set to `"point"`, then the `mark_size` value is directly given in points. When `mark_size_unit` is set to `"tabulated"`, `mark_size` is computed relative to the font size array: therefore, its value should be an integer in [0 5] whith stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt. Note that plot2d and pure scilab functions use `tabulated` mode as default ; when using plot function, the `point` mode is automatically enabled.

mark_size:
This field contains the default `mark_size` property value for Segs Rectangle Legend Axis Plot3d Fac3d and Polyline objects. `mark_size` selects the font size of mark to be displayed. Its value should be an integer in [0 5] whith stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt (see getmark).

mark_foreground:
This field contains the default `mark_foreground` property value for all objects created under this axes. Polyline, rectangle, legend, surface, segment and axis objects are using this property to specify a foreground (edge) color for their marks. Its value should be a color index (relative to the current color_map). Note that the default value is `-1` (default black) and, even if you change the `color_map`, this `-1` value will always point onto the default black color.

mark_background:
This property controls the default `mark_background` property value for all objects created under this axes. Polyline, rectangle, legend, surface, segment and axis objects are using this property to specify a background (face) color for their marks. It takes as value an index relative to the current colormap.Note that the default value is `-2` (default white) and, even if you change the `color_map`, this `-2` value will always point onto the default white color.

foreground:
: This field contains the default `foreground` property value for axes and all objects created under this axes. Its value should be a color index (relative to the current color_map). Note that the default value is `-1` (default black) and, even if you change the `color_map`, this `-1` value will always point onto the default black color.

background:
: This property controls the default `background` property value for axes and all objects created under this axes. It takes as value an index relative to the current colormap.Note that the default value is `-2` (default white) and, even if you change the `color_map`, this `-2` value will always point onto the default white color.

arc_drawing_mode:
: This property controls the default `arc_drawing_mode` property value for all created Arc objects under this Axes object. Its value should be either `"nurbs"` or `"lines"`.

clip_state:
: This field contains the default `clip_state` property value for all objects. Its value should be :

  - `"off"` this means that all objects created after that are not clipped (default value).
  - `"clipgrf"` this means that all objects created after that are clipped outside the Axes boundaries.
  - `"on"` this means that all objects created after that are clipped outside the rectangle given by property clip_box.

clip_box:
: This field contains the default `clip_box` property value for all objects. Its value should be an empty matrix if clip_state is `"off"`. Other case the clipping is given by the vector `[x,y,w,h]` (upper-left point width height).

Note on default values :

> All these listed properties and fields inherit from default values stored in an axes model. These default values can be seen and changed. To do so, use the `get("default_axes")` command : it returns a graphic handle on the axes model. Note that no graphic window is created by this command. The next created axes will inherit from this model (see "Example on axes model" below).

Examples

```
lines(0) // disables vertical paging
a=get("current_axes")//get the handle of the newly created axes
a.axes_visible="on"; // makes the axes visible
a.font_size=3; //set the tics label font size
a.x_location="top"; //set the x axis position
a.data_bounds=[-100,-2,-1;100,2,1]; //set the boundary values for the x, y a
a.sub_tics=[5,0];
a.labels_font_color=5;
a.grid=[2,2];
a.box="off";

// Example with 3D axes
clf(); //clear the graphics window
x=0.1:0.1:2*%pi;plot2d(x-.3,sin(x)*7+.2);
a=gca(); // get the handle of the current axes
a.grid=[1 -1 -1]; //make x-grid
a.rotation_angles=[70 250]; //turn the axes with giving angles
```

```
a.grid=[1 6 -1]; //make y-grid
a.view="2d"; //return te the 2d view
a.box="back_half";
a.labels_font_color=5;
a.children.children.thickness=4;
a.children.children.polyline_style=3;
a.view="3d"; //return te the 3d view
a.children.children.thickness=1;
a.children.children.foreground=2;
a.grid=[1 6 3]; //make z-grid
a.parent.background=4;
a.background=7;
plot2d(cos(x)+1,3*sin(x)-3);
plot2d(cos(x)+7,3*sin(x)+3);
a.children(2).children.polyline_style=2;
a.children(1).children.polyline_style=4;
a.children(1).children.foreground=5;
a.children(2).children.foreground=14;
a.parent.figure_size= [1200,800];
a.box="on";
a.labels_font_size=4;
a.parent.background=8;
a.parent.figure_size= [400,200];
a.rotation_angles=[0 260];
delete(a.children(2));
delete(); // delete current object

a = gca();
a.labels_font_size=1;
a.auto_clear= "on";
x=0:0.1:2.5*%pi;plot2d(10*cos(x),sin(x));

a.data_bounds(:,1) = [1;15] ; // set positive bounds for X axe
a.log_flags = "lnn" ; // set X axes to logarithmic scale
a.log_flags = "nnn" ; // switch back to linear scale

a=gca();
a.rotation_angles=[45 45];
a.data_bounds=[-20,-3,-2;20 3 ,2];
xrect([-4 0.5 8 1]);
a.auto_clear = "off" ;
a.isoview="on"; // isoview mode
xrect([-2 0.25 4 0.5]);
a.children(1).fill_mode="on";
a.axes_visible="off";
a.children(1).data=[-2 0.25 -1 4 0.5];
a.children(2).data=[-4 0.5 1 8 1];
x=2*%pi*(0:7)/8;
xv=[.2*sin(x);.9*sin(x)];yv=[.2*cos(x);.9*cos(x)];
xsegs(10*xv,yv,1:8)
s=a.children(1);
s.arrow_size=1;
s.segs_color=5;
a.data_bounds //the boundary values for the x,y and z coordinates
a.view="2d";
a.data_bounds=[-10,-1; 10,1]; // set the boundary values for the two-dimensi

// Example on axes model
```

```
da=gda() // get the handle on axes model to view and edit the fields
// title by default
da.title.text="My Default@Title"
da.title.foreground = 12;
da.title.font_size = 4;
// x labels default
da.x_label.text="x";
da.x_label.font_style = 8;
da.x_label.font_size = 2;
da.x_label.foreground = 5;
da.x_location = "middle";
// y labels default
da.y_label.text="y";
da.y_label.font_style = 3;
da.y_label.font_size = 5;
da.y_label.foreground = 3;
da.y_location = "right";
da.thickness = 2;
da.foreground = 7;
// the plot
x=(0:0.1:2*%pi)';
plot2d(x,[sin(x),sin(2*x),sin(3*x)],style=[1,2,3],rect=[0,-2,2*%pi,2]);
sda() // back to default axes model
```

See Also

lines , set , get , gca , gda , gcf , sda , sdf , scf , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

axis_properties — description of the axis entity properties

Description

The Axis entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for axis scaling and appearance.

Axis properties

parent:
: This property contains the handle of the parent. The parent of the axis entity should be of the type `"Axes"` or `"Compound"`.

visible:
: This field contains the `visible` property value for the entity . It should be `"on"` or `"off"` . By default, the axis entity is visible, the value's property is `"on"`. If `"off"`, the axis is not drawn on the screen.

tics_direction:
: Specify the direction of the tics drawn under the horizontal axis and the vertical axis. The possible values of this property are:

  - `"top"`. In this case, tics are drawn at the top of the horizontal axis.
  - `"bottom"`. In this case, tics are drawn at the bottom of the horizantal axis.
  - `"left"`. In this case, tics are going left on the vertical axis.
  - `"right"`. In this case, tics are going right on the vertical axis.

  The defaults values are `"top"` for the horizontal axis and `"right"` for vertical axis.

xtics_coord:
: This field represent the x-coordinate of the axis. It is a row vector containing values increasing from left to right which give tics positions for a horizontal axis. Other case, the entity is a vertical axis, this property contain a scale which defines the x-origin of the axis.

ytics_coord:
: This field represent the y-coordinate of the axis. It is a row vector containing values increasing from bottom to top which give tics positions for a vertical axis. Other case, the entity is a horizantal axis, this property contain a scale which defines the y-origin of the axis.

tics_color:
: The value of this properties is index of the color used to draw the axis'lines and tics.

tics_segment:
: This field contains a flag which controls the display of the base segment of the axis. The default is `"on"`, else if to not display it, the property takes `"off"` as value.

tics_style:
: This property describes how the tics are given. It is a string flag which can have these possible values:

  - `"v"`. It's the default value, In this case, tics positions are given by the row factor `xtics_coord` for horizontal axis (`ytics_coord` for the vertical one).
  - `"r"`. In this case, tics positions are given by the vector `[min,max,n]` where n is the number of intervals.

- `"i"`. In this case the vector given tics positions is of size 4, `[k1,k2,a,n]` then values are increasing between `k1*10^a` and `k2*10^a`, n is the number of intervals.

sub_tics:
This field sets the number of tics to draw between two main tics.

tics_labels:
This filed is a string matrix, which contains the strings to be drawn along the axis at tics positions.

labels_font_color:
This property determines the color of the tics labels.

labels_font_size:
It is a scalar specifying the character size of tics labels. If `fractional_font` property is `"off"` only the integer part of the value is used. For more information see graphics_fonts.

fractional_font:
This property specify whether ticks labels are displayed using fractional font sizes. Its value must be either `"on"` or `"off"`. If `"on"` the floating point value of `font_size` is used for display and the font is anti-aliased. If `"off"` only the integer part is used and the font is not smoothed.

clip_state:
This field contains the `clip_state` property value for the arc. Clip_state value should be :

- `"off"` this means that the axis is not clipped
- `"clipgrf"` this means that the axis is clipped outside the Axes box.
- `"on"` this means that the axis is clipped outside the arc given by property clip_box.

clip_box:
This field is to determinate the `clip_box` property. By Default its value should be an empty matrix if clip_state is "off". Other cases the vector `[x,y,w,h]` (upper-left point width height) defines the portions of the axis to display, however `clip_state` property value will be changed.

user_data:
This field can be use to store any scilab variable in the axis data structure, and to retreive it.

Examples

```
a=get("current_axes");//get the handle of the newly created axes
a.data_bounds=[-1,-1;10,10];

drawaxis(x=2:7,y=4,dir='u');
a1=a.children(1)
a1.xtics_coord=[1 4 5  8 10];
a1.tics_color=2;
a1.labels_font_size=3;
a1.tics_direction="bottom";
a1.tics_labels= [" February" "May"  "june" "August"  "October"];

drawaxis(x=1.2:1:10,y=5,dir='u',textcolor=13);
a2=get("hdl")
a2.sub_tics=0;
```

```
a2.tics_segment="off";
a2.ytics_coord=4;

drawaxis(x=-1,y=0:1:7,dir='r',fontsize=10,textcolor=5,ticscolor=6,sub_int=10)
a3=get("hdl");
a3.tics_labels= 'B'  +string(0:7);
a3.tics_direction="left";
```

See Also

set , get , delete , drawaxis , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

bar — bar histogram

```
bar(y)
bar(x,y)
bar([h],x,y [,width [,color [,style]]])
```

Parameters

h
: an axes handle, (default: h=gca()).

y
: a real scalar, vector of size N, or a matrice N*M.

x
: a real scalar or a vector of size N, (default: if y is a vector then x is a vector and x length equals to y length. If y is a matrix then x is a vector and x length equals to the lines number of y.

width
: (optional), a real scalar, defines the width (a percentage of the available room) for the bar (default: 0.8, i.e 80%).

color
: (optional), a string (default: 'blue'), specifing the inside color bar.

style:
: a string, 'grouped' or 'stacked' (default: 'grouped').

Description

`bar(y,...)` : if y is a vector then bar function draws a polyline which has the `polyline_style` type 6. If y is a vector, bar draws vector y versus vector 1:size(y,'*') . If y is a matrix N*M, bar draws M polylines (type 6), each polyline corresponds to a column of y versus vector x=1:size(y,1).

`bar(x,y,...)` : if y is a vector then bar function draws a polyline which has the `polyline_style` type 6, where x length = y length.If y is a matrix NxM then bar function draws M polylines which have the type 6. Each polyline corresponds to a column of y versus vector x.

`bar(h,...)` : defines the current axes where the drawing is performed.

`bar(...,width,...)` : defines the width of the bar(s) in percentage (generally: 0<width<=1).

`bar(...,style,...)` : defines how the bar is drawn. If y is a matrix N*M (so M polylines of type 6) then there are two ways to draw the M bars. the style option = 'grouped' allows to center the M polylines versus each components of x, and the style option 'stacked' allows to stack them.

`bar(...,color,...)` : defines the bar color. Bar functions uses the same colormap than in the `plot` function.

If there are several bar calls, the `barhomogenize` function allows to homegenize the width and style of all bars (i.e polylines of type 6) included in the current working axes.

Examples

```
// First example: draw a bar (i.e a polyline with polyline_style type =6) with
```

```
// width=0.5 and color='yellow' and default style='grouped', x=1:length(y).
scf(0);
y=[1 -3 5];
bar(y,0.5,'yellow');

// Second example: draw 3 bars (i.e 3 polylines with polyline_style type =6),de
scf(1);
x=[1 2 5]; y=[1  -5 6;3 -2 7;4  -3 8];
bar(x,y);

// Third example : style='stacked'.
scf(2);
x=[1 2 5]; y=[1 4 7;2 5 8;3 6  9];
bar(x,y,'stacked');

// Fourth example: width=0.2;color='green'; default style='grouped'
scf(3);
x=[1 2 5]; y=[1 4 7;2 5 8;3 6 9];
bar(x,y,0.2,'green');
```

See Also

barh , barhomogenize , plot , polyline_properties

Authors

Farid Belahcene

Name

barh — horizontal display of bar histogram

```
barh(y)
barh(x,y)
barh([h],x,y [,width [,color [,style]]])
```

Parameters

h
: an axes handle, (default: h=gca()).

y
: a real scalar, vector of size N, or a matrice N*M.

x
: a real scalar or a vector of size N, (default: if y is a vector then x is a vector and x length equals to y length. If y is a matrix then x is a vector and x length equals to the lines number of y.

width
: (optional), a real scalar, defines the width (a percentage of the available room) for the bar (default: 0.8, i.e=80%).

color
: (optional), a string (default: 'blue'), specifing the inside color bar.

style:
: a string, 'grouped' or 'stacked' (default: 'grouped').

Description

`barh(y,...)` : if y is a vector then bar function draws a polyline which has the `polyline_style` type 6. If y is a vector, bar draws vector y versus vector x=1:size(y,'*') . If y is a matrix N*M, bar draws M polylines (type 6), each polyline corresponds to a column of y versus vector x=1:size(y,1).

`barh(x,y,...)` : if y is a vector then bar function draws a polyline which has the `polyline_style` type 6, where x length = y length. If y is a matrix NxM then bar function draws M polylines which have the type 6. Each polyline corresponds to a column of y versus vector x.

`barh(h,...)` : defines the current axes where the drawing is performed.

`barh(...,width,...)` : defines the width of the bar(s) in percentage (generally: 0<width<1).

`barh(...,style,...)` : defines how the bar is drawn. If y is a matrix N*M (so M polylines of type 6) then there are two ways to draw the M bars. the style option = 'grouped' allows to center the M polylines versus each components of x, and the style option = 'stacked' allows to stack them.

`barh(...,color,...)` : defines the bar color. Bar functions uses the same colormap than in the `plot` function.

If there are several bar calls, the `barhomogenize` function allows to homegenize the width and style of all bars (i.e polylines of type 6) included in the current working axes.

Examples

```
// First example: draw a bar (i.e a polyline with polyline_style type =6),defau
scf(0);
y=[1 -3 5];
barh(y,0.5,'yellow');

// Second example: draw 3 bars (i.e 3 polylines with polyline_style type =6),de
scf(1);
x=[1 2 5]; y=[1  -5 6;3 -2 7;4  -3 8];
barh(x,y);

// Third example : style='stacked'.
scf(2);
x=[1 2 5]; y=[1 4 7;2 5 8;3 6  9];
barh(x,y,'stacked');

// Fourth example: width=0.2;color='green'; default style='grouped'
scf(3);
x=[1 2 5]; y=[1 4 7;2 5 8;3 6 9];
barh(x,y,0.2,'green');
```

See Also

bar , barhomogenize , plot , polyline_properties

Authors

Farid Belahcene

Name

barhomogenize — homogenize all the bars included in the current working axes

```
barhomogenize()
barhomogenize([h[,'style'[,'width']]])
```

Parameters

h
: an axes handle, (default: h=gca()).

style
: a string, 'grouped' or 'stacked' (default: 'grouped').

width
: (optional), a real scalar, defines the width (a percentage of the available room) for the bar (default: 0.8).

Description

If there are several bar calls, the `barhomogenize` function allows to homogenize the width and style of all bars (i.e which has the `polyline_style` type 6) included in the current working axes. These bars must have the same x data.

`barhomogenize( )` : takes the default values, h=gca(), width=0.8, style='grouped'.

`barhomogenize(h,...)` : defines the current axes where the drawing is performed.

`barhomogenize(...,width,...)` : defines the width of the bar(s) in percentage (generally: 0<width<=1).

`barhomogenize(...,style,...)` : defines how the bars are drawn. The 'grouped' option allows to center the M polylines versus each components of x, and the 'stacked' option allows to stack them.

Examples

```
// First example: creation of 1 yellow bar (i.e 1 polyline with polyline_style=
subplot(2,3,1)
xtitle('ex1: creation of 1 yellow bar and 3 bars ')
x=1:3; y1=1:3; y2=[4 3 5;6 7 8;9 10 11];
bar(x,y1,'yellow');bar(x,y2);
// grouped homogenisation of these 4 bars
subplot(2,3,2)
xtitle('grouped homogenisation')
x=1:3; y1=1:3; y2=[4 3 5;6 7 8;9 10 11];
bar(x,y1,'yellow');bar(x,y2);
barhomogenize();
// stacked homogenisation of thes 4 bars
subplot(2,3,3)
xtitle('stacked homogenisation')
x=1:3; y1=1:3; y2=[4 3 5;6 7 8;9 10 11];
bar(x,y1,'yellow');bar(x,y2);
barhomogenize('stacked',1);

// Second example : creation of 1 red bar (i.e 1 polyline with polyline_style=6
```

```
subplot(2,3,4)
xtitle('ex2: creation of 1 bar and 2 polylines')
x=1:10; y=sin(x)/2;
bar(x,y,'red')
x1=1:10;y1=[sin(x);cos(x)]
plot(x1,y1)
// modify the polyline_style type of the second polyline from plot (this polyli
subplot(2,3,5)
xtitle('transformation of the second polyline to bar')
x=1:10; y=sin(x)/2;
bar(x,y,'red')
x1=1:10;y1=[sin(x);cos(x)]
plot(x1,y1)
e=gce(); e2=e.children(2); e2.polyline_style=6;
// homogenisation of the first bar (from bar function) and second bar (from the
subplot(2,3,6)
xtitle('grouped homogenisation')
x=1:10; y=sin(x)/2;
bar(x,y,'red')
x1=1:10;y1=[sin(x);cos(x)]
plot(x1,y1)
e=gce(); e2=e.children(2); e2.polyline_style=6;
barhomogenize();
// change the style and the width
//barhomogenize('stacked',0.5);
//barhomogenize('stacked',1);
```

See Also

bar , polyline_properties

Authors

Farid Belacehne

Name

bonecolormap — gray colormap with a light blue tone

```
cmap=bonecolormap(n)
```

Parameters

n
: integer >= 3, the colormap size.

cmap
: matrix with 3 columns `[R,G,B]`.

Description

`bonecolormap` computes a gray colormap with a light blue tone.

Examples

```
f = scf();
plot3d1();
f.color_map = bonecolormap(32);
```

See Also

colormap , autumncolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

captions — draw graph captions

```
hl=captions(h, strings [,location])
```

Parameters

h
: vector of handles on polyline entities.

strings
: n vector of strings, strings(i) is the caption of the ith polyline

hl
: a handle of type "`Legend`", points to the structure containing all the captions information (see legend_properties.

location
: a character string with possible values:

- `"in_upper_right"` : captions are drawn in the upper right corner of the axes box.
- `"in_upper_left"`: captions are drawn in the upper left corner of the axes box.
- `"in_lower_right"`: captions are drawn in the lower right corner of the axes box.
- `"in_lower_left"`: captions are drawn in the lower left corner of the axes box.
- `"out_upper_right"`: captions are drawn at the right of the upper right corner of the axes box.
- `"out_upper_left"`: captions are drawn at the left of the upper left corner of the axes box.
- `"out_lower_right"`: captions are drawn at the right of the lower right corner of the axes box.
- `"out_lower_left"`: captions are drawn at the left of the lower left corner of the axes box.
- `"upper_caption"`: captions are drawn above the upper left corner of the axes box.
- `"lower_caption"`: captions are drawn below the lower left corner of the axes box. This option correspond to the `leg` argument of plot2d
- `"by_coordinates"`: the upper left corner of the captions box is given by the "position" field of the associated data structure. The `x` and `y` positions are given as fractions of the `axes_bounds sizes.`

Description

Puts captions on the current plot at the in the bottom left corner of the graphic window using the specified strings as labels. `captions` prepends labels by a recall of the corresponding polylines. The type and properties are recovered from the given handles:

The `captions` function creates a Legend data structure.

There is at most one `Legend` associated with each axes. If the caption function is recalled while a `Legend` still exist the old one is replaced.

Examples

```
t=0:0.1:2*%pi;
a=gca();a.data_bounds=[t(1) -1.8;t($) 1.8];
a.margins(4)=0.2;

plot2d(t,[cos(t'),cos(2*t'),cos(3*t')],[1,2 3]);
e=gce();
e.children(1).thickness=3;
e.children(2).line_style=4;

hl=captions(e.children,['cos(t)';'cos(2*t)';'cos(3*t)']);
hl=captions(e.children,['cos(t)';'cos(2*t)';'cos(3*t)'],'in_upper_right');

hl.legend_location='in_upper_right'
hl.fill_mode='on';
```

See Also

plot2d, legend, polyline_properties, legend_properties

Name

champ — 2D vector field plot

```
champ(x,y,fx,fy,[arfact,rect,strf])
champ(x,y,fx,fy,<opt_args>)
```

Parameters

x,y
: two vectors which define the grid.

fx
: a matrix which describes the x component of the vector field. `fx(i,j)` is the x component of the vector field at point `(x(i),y(j))`.

fy
: a matrix which describes the y component of the vector field. `fy(i,j)` is the y component of the vector field at point `(x(i),y(j))`.

<opt_args>
: This represents a sequence of statements `key1=value1,key2=value2`,... where `key1`, `key2,...` can be one of the following: arfact, rect, strf (see below).

arfact
: an optional argument of type real which gives a scale factor for the display of the arrow heads on the plot (default value is 1.0).

rect
: a vector `rect=[xmin,ymin,xmax,ymax]` which gives the boundaries of the graphics frame to use.

strf
: a string of length 3 "xyz" which has the same meaning as the `strf` parameter of `plot2d`. The first character x has no effect with `champ`.

Description

`champ` draws a 2D vector field. The length of the arrows is proportional to the intensity of the field.

If you want colored arrows with the color of the arrows depending on the intensity of the field, use `champ1`.

Enter the command `champ()` to see a demo.

Examples

```
// using rect as plot boundaries
champ(-5:5,-5:5,rand(11,11),rand(11,11),rect=[-10,-10,10,10],arfact=2)
// using (x,y) to get boundaries
clf()
champ(-5:5,-5:5,rand(11,11),rand(11,11),2,[-10,-10,10,10],"021")
```

See Also

champ1 , fchamp , plot2d

Authors

J.Ph.C.

Name

champ1 — 2D vector field plot with colored arrows

```
champ1(x,y,fx,fy,[arfact,rect,strf])
```

Parameters

x,y
: two vectors which define the grid.

fx
: a matrix which describes the x component of the vector field. `fx(i,j)` is the x component of the vector field at point `(x(i),y(j))`.

fy
: a matrix which describes the y component of the vector field. `fy(i,j)` is the y component of the vector field at point `(x(i),y(j))`.

arfact
: an optional argument of type real which gives a scale factor for the display of the arrow heads on the plot (default value is 1.0).

rect
: a vector `rect=[xmin,ymin,xmax,ymax]` which gives the boundaries of the graphics frame to use.

frameflag
: controls the computation of the actual coordinate ranges from the minimal requested values. The associated value should be an integer ranging from 0 to 8.

axesflag
: specifies how the axes are drawn. The associated value should be an integer ranging from 0 to 5.

strf
: a string of length 3 "xyz" which has the same meaning as the `strf` parameter of `plot2d`. The first character x has no effect with `champ1`.

Description

`champ1` draws a 2D vector field with colored arrows. The color of the arrows depends on the intensity of the field.

If you want arrows proportional to the intensity of the field, use `champ`.

Enter the command `champ1()` to see a demo.

Examples

```
champ1(-5:5,-5:5,rand(11,11),rand(11,11),rect=[-10,-10,10,10],arfact=2)
```

See Also

champ , fchamp , plot2d

Authors

J.Ph.C.

Name

champ_properties — description of the 2D vector field entity properties

Description

The Champ entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for a 2D vector field.

visible:
: This properties contains the `visible` property value for the entity . It should be `"on"` or `"off"` . If `"on"` the vector field is drawn, If `"off"` the vector field is not displayed on the screen.

data:
: This field defines a `tlist` data structure of type `"champdata"` composed of a row and column indices of each element : the x and y grid coordinates are contained respectively in `data.x` and `data.y`. The complementary fields named `data.fx` and `data.fy` are matrices which describe respectively the `x` and `y` component of the vector field at point `(x(i),y(j))`.

user_data:
: This field can be use to store any scilab variable in the champ data structure, and to retreive it.

line_style:
: The `line_style` property value should be an integer in [0 9]. 0 stands for solid the other value stands for a selection of dashes. This property applies to all lines used to draw the vector field.

thickness:
: This property contains the `thickness` property for all lines used to draw the vector field. Its value should be a non negative integer..

colored:
: If this this property value is `"on"`, fields vectors are drawn using a color proportional to the intensity of the field.

clip_state:
: This field contains the `clip_state` property value for the champ. It should be :

  - `"off"` this means that the vector field is not clipped
  - `"clipgrf"` this means that the vector field is clipped outside the Axes box.
  - `"on"` this means that the vector field is clipped outside the rectangle given by property clip_box.

clip_box:
: This property contains the `clip_box` property. Its value should be an empty matrix if clip_state is "off" .Other cases the vector `[x,y,w,h]` (upper-left point width height) defines the portions of the vector field to display, however `clip_state` property value will be changed.

parent:
: This property contains the handle of the parent. The parent of the 2D vector field entity should be of the type `"Axes"` or `"Compound"`.

Examples

```
a=get("current_axes");//get the handle of the newly created axes
a.data_bounds=[-10,-10;10,10];
```

```
champ(-5:5,-5:5,rand(11,11),rand(11,11))

c=a.children

c.colored="on";
c.thickness=2;
c.data // display the tlist of type "scichampdata"
a.data_bounds=[-5,-5;5,5];
```

See Also

set , get , delete , champ , champ1 , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

clear_pixmap — erase the pixmap buffer

```
clear_pixmap()
```

Description

If a graphic window `pixmap` property is `"on"` the drawings are send to a pixmap memory instead of the screen display.

The `clear_pixmap()` instruction erase the pixmap, but not the screen.

The pixmap mode can be used to obtain smooth animations.

See Also

figure_properties , show_pixmap

Authors

Serge Steer INRIA

Name

clf — clear or reset the current graphic figure (window) to default values

```
clf(<opt_job_arg>)
clf(h,<opt_job_arg>)
clf(num,<opt_job_arg>)
```

Parameters

h
: a handle, the figure handle

num
: a number, the figure_id

<opt_job_arg>
: a string (`'clear'` or `'reset'`) specifying the job for `clf`.

Description

The `clf` command is used to reset a figure to its default values and/or to delete all its children.

If `opt_job_arg` string value is `'clear'` then all children of the specified figure are deleted.

If `opt_job_arg` string value is `'reset'` then not only all children of the specified figure are deleted but the figure properties are reset to their default values using the default figure model (see gdf). The only exception are the `axes_size` and `figure_size` properties which can not be reset if the figure is docked with other elements.

`clf(num)` clear or reset the figure with `figure_id==num`.

`clf(h)` clear or reset the figure pointed to by the handle `h`.

`clf()` clear or reset the current figure.

`clf` function delete every children of specified windows including menus and uicontrols added by user. To prevent menus and uicontrols from being deleted, the `delete(gca())` command might be used instead.

Examples

```
f4=scf(4);  //creates figure with id==4 and make it the current one
f4.color_map = jetcolormap(64);
f4.figure_size = [400, 200];
plot2d()
clf(f4,'reset')

f0=scf(0); //creates figure with id==0 and make it the current one
f0.color_map=hotcolormap(128); // change color map
t=-%pi:0.3:%pi;
plot3d1(t,t,sin(t)'*cos(t));

clf() // equivalent to clf(gcf(),'clear')
plot3d1(t,t,sin(t)'*cos(t));  // color_map unchanged

clf(gcf(),'reset')
```

```
  plot3d1(t,t,sin(t)'*cos(t));  // color_map changed (back to the default one
```

See Also

set , get , gcf , gdf , scf , graphics_entities

Authors

S. Steer & F.Leray INRIA

Name

color — returns the color id of a color

```
id=color(name)
id=color(r,g,b)
```

Parameters

name
: name of a color.

r,g,b
: RGB integer values of a color.

id
: id of the color.

Description

`color` returns the color id corresponding to its argument:

- `name` must be the name of a known color (see color_list).
- `r`, `g` and `b` must be integers between 0 and 255 corresponding to colors components red, green and blue. As usual 0 means no intensity and 255 means all the intensity of the color.

If the requested color does not exist in the current colormap it is added to the colormap.

This function can be used to specify the foreground or background colors when plotting.

Examples

```
x=linspace(-2*%pi,2*%pi,100)';
// using existing colors
plot2d(x,[sin(x),cos(x)],style=[color("red"),color("green")]);
// new colors: there are added to the colormap
e=gce(); p1=e.children(1); p2=e.children(2);
p1.foreground=color("purple"); p2.foreground=color("navy blue");
// using RGV values
p1.foreground=color(255,128,128);
```

See Also

colormap , color_list , getcolor

Name

color_list — list of named colors

Description

You will find below the names of the colors known by Scilab. The RGB values are given after the name. Note that sometimes colors have more than 1 name.

scilab blue4	0	0	144
scilabblue4	0	0	144
scilab blue3	0	0	176
scilabblue3	0	0	176
scilab blue2	0	0	208
scilabblue2	0	0	208
scilab green4	0	144	0
scilabgreen4	0	144	0
scilab green3	0	176	0
scilabgreen3	0	176	0
scilab green2	0	208	0
scilabgreen2	0	208	0
scilab cyan4	0	144	144
scilabcyan4	0	144	144
scilab cyan3	0	176	176
scilabcyan3	0	176	176
scilab cyan2	0	208	208
scilabcyan2	0	208	208
scilab red4	144	0	0
scilabred4	144	0	0
scilab red3	176	0	0
scilabred3	176	0	0
scilab red2	208	0	0
scilabred2	208	0	0
scilab magenta4	144	0	144
scilabmagenta4	144	0	144
scilab magenta3	176	0	176
scilabmagenta3	176	0	176
scilab magenta2	208	0	208
scilabmagenta2	208	0	208
scilab brown4	128	48	0
scilabbrown4	128	48	0
scilab brown3	160	64	0
scilabbrown3	160	64	0
scilab brown2	192	96	0
scilabbrown2	192	96	0

scilab pink4	255	128	128
scilabpink4	255	128	128
scilab pink3	255	160	160
scilabpink3	255	160	160
scilab pink2	255	192	192
scilabpink2	255	192	192
scilab pink	255	224	224
scilabpink	255	224	224

snow	255	250	250
ghost white	248	248	255
ghostwhite	248	248	255
white smoke	245	245	245
whitesmoke	245	245	245
gainsboro	220	220	220
floral white	255	250	240
floralwhite	255	250	240
old lace	253	245	230
oldlace	253	245	230
linen	250	240	230
antique white	250	235	215
antiquewhite	250	235	215

papaya whip	255	239	213
papayawhip	255	239	213
blanched almond	255	235	205
blanchedalmond	255	235	205
bisque	255	228	196

peach puff	255	218	185
peachpuff	255	218	185
navajo white	255	222	173
navajowhite	255	222	173
moccasin	255	228	181
cornsilk	255	248	220
ivory	255	255	240
lemon chiffon	255	250	205
lemonchiffon	255	250	205
seashell	255	245	238
honeydew	240	255	240
mint cream	245	255	250
mintcream	245	255	250
azure	240	255	255
alice blue	240	248	255

aliceblue	240	248	255
lavender	230	230	250
lavender blush	255	240	245
lavenderblush	255	240	245

misty rose	255	228	225
mistyrose	255	228	225
white	255	255	255
black	0	0	0
dark slate gray	47	79	79
darkslategray	47	79	79
dark slate grey	47	79	79
darkslategrey	47	79	79
dim gray	105	105	105
dimgray	105	105	105
dim grey	105	105	105
dimgrey	105	105	105
slate gray	112	128	144
slategray	112	128	144
slate grey	112	128	144
slategrey	112	128	144
light slate gray	119	136	153
lightslategray	119	136	153
light slate grey	119	136	153
lightslategrey	119	136	153
gray	190	190	190
grey	190	190	190
light grey	211	211	211
lightgrey	211	211	211
light gray	211	211	211
lightgray	211	211	211

midnight blue	25	25	112
midnightblue	25	25	112
navy	0	0	128
navy blue	0	0	128
navyblue	0	0	128
cornflower blue	100	149	237
cornflowerblue	100	149	237
dark slate blue	72	61	139
darkslateblue	72	61	139
slate blue	106	90	205
slateblue	106	90	205

medium slate blue	123	104	238
mediumslateblue	123	104	238
light slate blue	132	112	255
lightslateblue	132	112	255
medium blue	0	0	205
mediumblue	0	0	205
royal blue	65	105	225
royalblue	65	105	225
blue	0	0	255
dodger blue	30	144	255
dodgerblue	30	144	255
deep sky blue	0	191	255
deepskyblue	0	191	255
sky blue	135	206	235
skyblue	135	206	235
light sky blue	135	206	250
lightskyblue	135	206	250
steel blue	70	130	180
steelblue	70	130	180
light steel blue	176	196	222
lightsteelblue	176	196	222
light blue	173	216	230
lightblue	173	216	230
powder blue	176	224	230
powderblue	176	224	230

pale turquoise	175	238	238
paleturquoise	175	238	238
dark turquoise	0	206	209
darkturquoise	0	206	209
medium turquoise	72	209	204
mediumturquoise	72	209	204
turquoise	64	224	208
cyan	0	255	255
light cyan	224	255	255
lightcyan	224	255	255
cadet blue	95	158	160
cadetblue	95	158	160
medium aquamarine	102	205	170
mediumaquamarine	102	205	170
aquamarine	127	255	212

dark green	0	100	0

darkgreen	0	100	0
dark olive green	85	107	47
darkolivegreen	85	107	47
dark sea green	143	188	143
darkseagreen	143	188	143
sea green	46	139	87
seagreen	46	139	87
medium sea green	60	179	113
mediumseagreen	60	179	113
light sea green	32	178	170
lightseagreen	32	178	170
pale green	152	251	152
palegreen	152	251	152
spring green	0	255	127
springgreen	0	255	127
lawn green	124	252	0
lawngreen	124	252	0
green	0	255	0
chartreuse	127	255	0
medium spring green	0	250	154
mediumspringgreen	0	250	154
green yellow	173	255	47
greenyellow	173	255	47
lime green	50	205	50
limegreen	50	205	50
yellow green	154	205	50
yellowgreen	154	205	50
forest green	34	139	34
forestgreen	34	139	34
olive drab	107	142	35
olivedrab	107	142	35

dark khaki	189	183	107
darkkhaki	189	183	107
khaki	240	230	140
pale goldenrod	238	232	170
palegoldenrod	238	232	170
light goldenrod yellow	250	250	210
lightgoldenrodyellow	250	250	210
light yellow	255	255	224
lightyellow	255	255	224
yellow	255	255	0

gold	255	215	0
light goldenrod	238	221	130
lightgoldenrod	238	221	130
goldenrod	218	165	32
dark goldenrod	184	134	11
darkgoldenrod	184	134	11

rosy brown	188	143	143
rosybrown	188	143	143
indian red	205	92	92
indianred	205	92	92
saddle brown	139	69	19
saddlebrown	139	69	19
sienna	160	82	45
peru	205	133	63
burlywood	222	184	135
beige	245	245	220
wheat	245	222	179
sandy brown	244	164	96
sandybrown	244	164	96
tan	210	180	140
chocolate	210	105	30
firebrick	178	34	34
brown	165	42	42
dark salmon	233	150	122
darksalmon	233	150	122
salmon	250	128	114
light salmon	255	160	122
lightsalmon	255	160	122

orange	255	165	0
dark orange	255	140	0
darkorange	255	140	0
coral	255	127	80
light coral	240	128	128
lightcoral	240	128	128
tomato	255	99	71
orange red	255	69	0
orangered	255	69	0
red	255	0	0
hot pink	255	105	180
hotpink	255	105	180
deep pink	255	20	147

deeppink	255	20	147
pink	255	192	203
light pink	255	182	193
lightpink	255	182	193

pale violet red	219	112	147
palevioletred	219	112	147
maroon	176	48	96
medium violet red	199	21	133
mediumvioletred	199	21	133
violet red	208	32	144
violetred	208	32	144
magenta	255	0	255
violet	238	130	238
plum	221	160	221
orchid	218	112	214
medium orchid	186	85	211
mediumorchid	186	85	211
dark orchid	153	50	204
darkorchid	153	50	204
dark violet	148	0	211
darkviolet	148	0	211
blue violet	138	43	226
blueviolet	138	43	226

purple	160	32	240
medium purple	147	112	219
mediumpurple	147	112	219
thistle	216	191	216
snow1	255	250	250
snow2	238	233	233
snow3	205	201	201
snow4	139	137	137
seashell1	255	245	238
seashell2	238	229	222
seashell3	205	197	191
seashell4	139	134	130
antiquewhite1	255	239	219
antiquewhite2	238	223	204
antiquewhite3	205	192	176
antiquewhite4	139	131	120

bisque1	255	228	196
bisque2	238	213	183

bisque3	205	183	158
bisque4	139	125	107
peachpuff1	255	218	185
peachpuff2	238	203	173
peachpuff3	205	175	149
peachpuff4	139	119	101
navajowhite1	255	222	173
navajowhite2	238	207	161
navajowhite3	205	179	139
navajowhite4	139	121	94
lemonchiffon1	255	250	205
lemonchiffon2	238	233	191
lemonchiffon3	205	201	165
lemonchiffon4	139	137	112
cornsilk1	255	248	220
cornsilk2	238	232	205
cornsilk3	205	200	177
cornsilk4	139	136	120
ivory1	255	255	240
ivory2	238	238	224
ivory3	205	205	193
ivory4	139	139	131
honeydew1	240	255	240
honeydew2	224	238	224
honeydew3	193	205	193
honeydew4	131	139	131

lavenderblush1	255	240	245
lavenderblush2	238	224	229
lavenderblush3	205	193	197
lavenderblush4	139	131	134
mistyrose1	255	228	225
mistyrose2	238	213	210
mistyrose3	205	183	181
mistyrose4	139	125	123

azure1	240	255	255
azure2	224	238	238
azure3	193	205	205
azure4	131	139	139
slateblue1	131	111	255
slateblue2	122	103	238
slateblue3	105	89	205

slateblue4	71	60	139
royalblue1	72	118	255
royalblue2	67	110	238
royalblue3	58	95	205
royalblue4	39	64	139
blue1	0	0	255
blue2	0	0	238
blue3	0	0	205
blue4	0	0	139
dodgerblue1	30	144	255
dodgerblue2	28	134	238
dodgerblue3	24	116	205
dodgerblue4	16	78	139
steelblue1	99	184	255
steelblue2	92	172	238
steelblue3	79	148	205
steelblue4	54	100	139
deepskyblue1	0	191	255
deepskyblue2	0	178	238
deepskyblue3	0	154	205
deepskyblue4	0	104	139
skyblue1	135	206	255
skyblue2	126	192	238
skyblue3	108	166	205
skyblue4	74	112	139
lightskyblue1	176	226	255
lightskyblue2	164	211	238
lightskyblue3	141	182	205
lightskyblue4	96	123	139

slategray1	198	226	255
slategray2	185	211	238
slategray3	159	182	205
slategray4	108	123	139
lightsteelblue1	202	225	255
lightsteelblue2	188	210	238
lightsteelblue3	162	181	205
lightsteelblue4	110	123	139
lightblue1	191	239	255
lightblue2	178	223	238
lightblue3	154	192	205
lightblue4	104	131	139

lightcyan1	224	255	255
lightcyan2	209	238	238
lightcyan3	180	205	205
lightcyan4	122	139	139

paleturquoise1	187	255	255
paleturquoise2	174	238	238
paleturquoise3	150	205	205
paleturquoise4	102	139	139
cadetblue1	152	245	255
cadetblue2	142	229	238
cadetblue3	122	197	205
cadetblue4	83	134	139
turquoise1	0	245	255
turquoise2	0	229	238
turquoise3	0	197	205
turquoise4	0	134	139
cyan1	0	255	255
cyan2	0	238	238
cyan3	0	205	205
cyan4	0	139	139

darkslategray1	151	255	255
darkslategray2	141	238	238
darkslategray3	121	205	205
darkslategray4	82	139	139
aquamarine1	127	255	212
aquamarine2	118	238	198
aquamarine3	102	205	170
aquamarine4	69	139	116
darkseagreen1	193	255	193
darkseagreen2	180	238	180
darkseagreen3	155	205	155
darkseagreen4	105	139	105
seagreen1	84	255	159
seagreen2	78	238	148
seagreen3	67	205	128
seagreen4	46	139	87
palegreen1	154	255	154
palegreen2	144	238	144
palegreen3	124	205	124
palegreen4	84	139	84
springgreen1	0	255	127

springgreen2	0	238	118
springgreen3	0	205	102
springgreen4	0	139	69
green1	0	255	0
green2	0	238	0
green3	0	205	0
green4	0	139	0
chartreuse1	127	255	0
chartreuse2	118	238	0
chartreuse3	102	205	0
chartreuse4	69	139	0
olivedrab1	192	255	62
olivedrab2	179	238	58
olivedrab3	154	205	50
olivedrab4	105	139	34
darkolivegreen1	202	255	112
darkolivegreen2	188	238	104
darkolivegreen3	162	205	90
darkolivegreen4	110	139	61
khaki1	255	246	143
khaki2	238	230	133
khaki3	205	198	115
khaki4	139	134	78

lightgoldenrod1	255	236	139
lightgoldenrod2	238	220	130
lightgoldenrod3	205	190	112
lightgoldenrod4	139	129	76
lightyellow1	255	255	224
lightyellow2	238	238	209
lightyellow3	205	205	180
lightyellow4	139	139	122
yellow1	255	255	0
yellow2	238	238	0
yellow3	205	205	0
yellow4	139	139	0
gold1	255	215	0
gold2	238	201	0
gold3	205	173	0
gold4	139	117	0
goldenrod1	255	193	37
goldenrod2	238	180	34

goldenrod3	205	155	29
goldenrod4	139	105	20
darkgoldenrod1	255	185	15
darkgoldenrod2	238	173	14
darkgoldenrod3	205	149	12
darkgoldenrod4	139	101	8

rosybrown1	255	193	193
rosybrown2	238	180	180
rosybrown3	205	155	155
rosybrown4	139	105	105
indianred1	255	106	106
indianred2	238	99	99
indianred3	205	85	85
indianred4	139	58	58
sienna1	255	130	71
sienna2	238	121	66
sienna3	205	104	57
sienna4	139	71	38
burlywood1	255	211	155
burlywood2	238	197	145
burlywood3	205	170	125
burlywood4	139	115	85

wheat1	255	231	186
wheat2	238	216	174
wheat3	205	186	150
wheat4	139	126	102
tan1	255	165	79
tan2	238	154	73
tan3	205	133	63
tan4	139	90	43

chocolate1	255	127	36
chocolate2	238	118	33
chocolate3	205	102	29
chocolate4	139	69	19
firebrick1	255	48	48
firebrick2	238	44	44
firebrick3	205	38	38
firebrick4	139	26	26
brown1	255	64	64
brown2	238	59	59

brown3	205	51	51
brown4	139	35	35
salmon1	255	140	105
salmon2	238	130	98
salmon3	205	112	84
salmon4	139	76	57
lightsalmon1	255	160	122
lightsalmon2	238	149	114
lightsalmon3	205	129	98
lightsalmon4	139	87	66

orange1	255	165	0
orange2	238	154	0
orange3	205	133	0
orange4	139	90	0
darkorange1	255	127	0
darkorange2	238	118	0
darkorange3	205	102	0
darkorange4	139	69	0
coral1	255	114	86
coral2	238	106	80
coral3	205	91	69
coral4	139	62	47
tomato1	255	99	71
tomato2	238	92	66
tomato3	205	79	57
tomato4	139	54	38
orangered1	255	69	0
orangered2	238	64	0
orangered3	205	55	0
orangered4	139	37	0
red1	255	0	0
red2	238	0	0
red3	205	0	0
red4	139	0	0
deeppink1	255	20	147
deeppink2	238	18	137
deeppink3	205	16	118
deeppink4	139	10	80
hotpink1	255	110	180
hotpink2	238	106	167
hotpink3	205	96	144

hotpink4	139	58	98
pink1	255	181	197
pink2	238	169	184
pink3	205	145	158
pink4	139	99	108
lightpink1	255	174	185
lightpink2	238	162	173
lightpink3	205	140	149
lightpink4	139	95	101

palevioletred1	255	130	171
palevioletred2	238	121	159
palevioletred3	205	104	137
palevioletred4	139	71	93
maroon1	255	52	179
maroon2	238	48	167
maroon3	205	41	144
maroon4	139	28	98
violetred1	255	62	150
violetred2	238	58	140
violetred3	205	50	120
violetred4	139	34	82

magenta1	255	0	255
magenta2	238	0	238
magenta3	205	0	205
magenta4	139	0	139
orchid1	255	131	250
orchid2	238	122	233
orchid3	205	105	201
orchid4	139	71	137
plum1	255	187	255
plum2	238	174	238
plum3	205	150	205
plum4	139	102	139
mediumorchid1	224	102	255
mediumorchid2	209	95	238
mediumorchid3	180	82	205
mediumorchid4	122	55	139
darkorchid1	191	62	255
darkorchid2	178	58	238
darkorchid3	154	50	205
darkorchid4	104	34	139

purple1	155	48	255
purple2	145	44	238
purple3	125	38	205
purple4	85	26	139
mediumpurple1	171	130	255
mediumpurple2	159	121	238
mediumpurple3	137	104	205
mediumpurple4	93	71	139

thistle1	255	225	255
thistle2	238	210	238
thistle3	205	181	205
thistle4	139	123	139

gray0	0	0	0
grey0	0	0	0
gray1	3	3	3
grey1	3	3	3
gray2	5	5	5
grey2	5	5	5
gray3	8	8	8
grey3	8	8	8
gray4	10	10	10
grey4	10	10	10
gray5	13	13	13
grey5	13	13	13
gray6	15	15	15
grey6	15	15	15
gray7	18	18	18
grey7	18	18	18
gray8	20	20	20
grey8	20	20	20
gray9	23	23	23
grey9	23	23	23
gray10	26	26	26
grey10	26	26	26
gray11	28	28	28
grey11	28	28	28
gray12	31	31	31
grey12	31	31	31
gray13	33	33	33
grey13	33	33	33
gray14	36	36	36

grey14	36	36	36
gray15	38	38	38
grey15	38	38	38
gray16	41	41	41
grey16	41	41	41
gray17	43	43	43
grey17	43	43	43
gray18	46	46	46
grey18	46	46	46
gray19	48	48	48
grey19	48	48	48
gray20	51	51	51
grey20	51	51	51
gray21	54	54	54
grey21	54	54	54
gray22	56	56	56
grey22	56	56	56
gray23	59	59	59
grey23	59	59	59
gray24	61	61	61
grey24	61	61	61
gray25	64	64	64
grey25	64	64	64
gray26	66	66	66
grey26	66	66	66
gray27	69	69	69
grey27	69	69	69
gray28	71	71	71
grey28	71	71	71
gray29	74	74	74
grey29	74	74	74
gray30	77	77	77
grey30	77	77	77
gray31	79	79	79
grey31	79	79	79
gray32	82	82	82
grey32	82	82	82
gray33	84	84	84
grey33	84	84	84
gray34	87	87	87
grey34	87	87	87
gray35	89	89	89

grey35	89	89	89
gray36	92	92	92
grey36	92	92	92
gray37	94	94	94
grey37	94	94	94
gray38	97	97	97

grey38	97	97	97
gray39	99	99	99
grey39	99	99	99
gray40	102	102	102
grey40	102	102	102
gray41	105	105	105
grey41	105	105	105
gray42	107	107	107
grey42	107	107	107
gray43	110	110	110
grey43	110	110	110
gray44	112	112	112
grey44	112	112	112
gray45	115	115	115
grey45	115	115	115
gray46	117	117	117
grey46	117	117	117
gray47	120	120	120
grey47	120	120	120
gray48	122	122	122
grey48	122	122	122
gray49	125	125	125
grey49	125	125	125
gray50	127	127	127
grey50	127	127	127
gray51	130	130	130
grey51	130	130	130
gray52	133	133	133
grey52	133	133	133
gray53	135	135	135
grey53	135	135	135
gray54	138	138	138
grey54	138	138	138
gray55	140	140	140
grey55	140	140	140

gray56	143	143	143
grey56	143	143	143
gray57	145	145	145
grey57	145	145	145
gray58	148	148	148
grey58	148	148	148
gray59	150	150	150
grey59	150	150	150
gray60	153	153	153
grey60	153	153	153

gray61	156	156	156
grey61	156	156	156
gray62	158	158	158
grey62	158	158	158
gray63	161	161	161
grey63	161	161	161
gray64	163	163	163
grey64	163	163	163
gray65	166	166	166
grey65	166	166	166
gray66	168	168	168
grey66	168	168	168
gray67	171	171	171
grey67	171	171	171
gray68	173	173	173
grey68	173	173	173
gray69	176	176	176
grey69	176	176	176
gray70	179	179	179
grey70	179	179	179
gray71	181	181	181
grey71	181	181	181
gray72	184	184	184
grey72	184	184	184
gray73	186	186	186
grey73	186	186	186
gray74	189	189	189
grey74	189	189	189

gray75	191	191	191
grey75	191	191	191
gray76	194	194	194

grey76	194	194	194
gray77	196	196	196
grey77	196	196	196
gray78	199	199	199
grey78	199	199	199
gray79	201	201	201
grey79	201	201	201
gray80	204	204	204
grey80	204	204	204
gray81	207	207	207
grey81	207	207	207
gray82	209	209	209
grey82	209	209	209
gray83	212	212	212
grey83	212	212	212
gray84	214	214	214
grey84	214	214	214
gray85	217	217	217
grey85	217	217	217
gray86	219	219	219
grey86	219	219	219
gray87	222	222	222
grey87	222	222	222

gray88	224	224	224
grey88	224	224	224
gray89	227	227	227
grey89	227	227	227
gray90	229	229	229
grey90	229	229	229
gray91	232	232	232
grey91	232	232	232
gray92	235	235	235
grey92	235	235	235
gray93	237	237	237
grey93	237	237	237
gray94	240	240	240
grey94	240	240	240
gray95	242	242	242
grey95	242	242	242
gray96	245	245	245
grey96	245	245	245

gray97	247	247	247
grey97	247	247	247
gray98	250	250	250
grey98	250	250	250
gray99	252	252	252
grey99	252	252	252
gray100	255	255	255
grey100	255	255	255

dark grey	169	169	169
darkgrey	169	169	169
dark gray	169	169	169
darkgray	169	169	169
dark blue	0	0	139
darkblue	0	0	139
dark cyan	0	139	139
darkcyan	0	139	139
dark magenta	139	0	139
darkmagenta	139	0	139
dark red	139	0	0
darkred	139	0	0
light green	144	238	144
lightgreen	144	238	144

See Also

color , name2rgb , rgb2name

Name

colorbar — draw a colorbar

```
colorbar(umin, umax [, colminmax, fmt])
```

Parameters

umin
: real scalar, the minimum value associated with the plot

umax
: real scalar, the maximum value associated with the plot

colminmax
: (optional) a vector with 2 integer components

fmt
: (optional) a string to set up the display format for colorbar graduations

Description

Draw a colorbar for a plot3d, fec, Sgrayplot, etc... The function may be called **BEFORE** the plot3d, fec, Sgrayplot,... because its sets and changes the frame for the plot. This way the colorbar is not part of the associated plot and so is not modified by a zoom or a rotation.

The optional argument `colminmax` may be used to precise the first color (associated with umin) and the the last color (associated with umax) of the current colormap. By default `colminmax=[1 nb_colors]` where nb_colors is the number of colors of the current colormap.

The optional argument `fmt` is a string containing a C-format, like `"%.2f"`, `"%e"`, etc...

For the 2 optional arguments you can use the syntax `keyword=value` which is useful to give the format without giving `colminmax` (see the last example).

Examples

```
// example 1
x = linspace(0,1,81);
z = cos(2*%pi*x)'*sin(2*%pi*x);
zm = min(z); zM = max(z);
xbasc()
xset("colormap",jetcolormap(64))
colorbar(zm,zM)
Sgrayplot(x,x,z)
xtitle("The function cos(2 pi x)sin(2 pi y)")

// example 2
x = linspace(0,1,81);
z = cos(2*%pi*x)'*sin(2*%pi*x);
zm = min(z); zM = max(z);
zz = abs(0.5*cos(2*%pi*x)'*cos(2*%pi*x));
zzm = min(zz); zzM = max(zz);
xbasc();
xset("colormap",jetcolormap(64))

drawlater() ;
```

```
subplot(2,2,1)
   colorbar(zm,zM)
   Sgrayplot(x,x,z, strf="031", rect=[0 0 1 1])
   xtitle("a Sgrayplot with a colorbar")
subplot(2,2,2)
   colorbar(zm,zM)
   plot3d1(x,x,z)
   xtitle("a plot3d1 with a colorbar")
subplot(2,2,3)
   colorbar(zzm,zzM)
   plot3d1(x,x,zz)
   xtitle("a plot3d1 with a colorbar")
subplot(2,2,4)
   colorbar(zzm,zzM)
   Sgrayplot(x,x,zz, strf="031", rect=[0 0 1 1])
   xtitle("a Sgrayplot with a colorbar")
drawnow() ;


// example 3
x = linspace(0,1,81);
zz = abs(0.5*cos(2*%pi*x)'*cos(2*%pi*x));
zzm = min(zz); zzM = max(zz);
[xf,yf,zf]=genfac3d(x,x,zz);
nb_col = 64;
xbasc()
xset("colormap",hotcolormap(nb_col))
drawlater() ;
colorbar(zzm,zzM,fmt="%.1f")
nbcol = xget("lastpattern")
zcol = dsearch(zf, linspace(zzm, zzM, nb_col+1));
plot3d(xf, yf, list(zf, zcol), flag = [-2 6 4])
xtitle("a plot3d with shaded interpolated colors")
drawnow() ;
xselect()
```

See Also

colormap

Authors

B. Pincon, S. Steer

Name

colordef — Set default color values to display different color schemes

```
colordef(color_scheme)
colordef(f,color_scheme)
colordef('new',color_scheme)
```

Parameters

color_scheme
: a character string with possible value: 'white', 'black','none'

f
: a handle on a graphic figure

Description

- `colordef('white')` sets the default figure (see gdf) colormap to `jetcolormap(64)`, the default figure background color to light gray and the default axes (see gda) background color to white, the axes lines foreground and font color to black.
- `colordef('black')` sets the default figure (see gdf) colormap to `jetcolormap(64)`, the default figure background color to dark gray and the default axes (see gda) background color to black, the axes lines foreground and font color to white.
- `colordef('none')` sets the default figure (see gdf) colormap to `hsvcolormap(64)`, the default figure background color to dark gray and the default axes (see gda) background color to black, the axes lines foreground and font color to white.
- `colordef(f,color_scheme)` sets the color properties of figure given by the handle `f` as well as the color properties of its current axes.
- `colordef('new',color_scheme)` creates a new graphic window and its color properties as well as the properties of its axes.

Examples

```
// Add here scilab instructions and comments
```

See Also

gdf , gda , figure_properties , axes_properties

Authors

S. Steer
INRIA

Name

colormap — using colormaps

Description

A colormap `cmap` is defined by a m x 3 matrix. m is the number of colors. Color number i is given as a 3-uple `cmap(i,1)`, `cmap(i,2)` `cmap(i,3)` corresponding respectively to red, green and blue intensity between 0 and 1.

At the beginning, 32 colors are defined in the colormap. You can change the colormap of a figure by using `set(f,"color_map",cmap)` where `f` is the handle of the figure.

Each color in the colormap has an id you have to use to specify color in most plot functions. To see the ids, use function getcolor.

The functions hotcolormap, jetcolormap and graycolormap provide colormaps with continuous variation of the colors.

You can get the default colormap by `cmap=get(sdf(),"color_map")`.

Examples

```
n=64;
r=linspace(0,1,n)';
g=linspace(1,0,n)';
b=ones(r);
cmap=[r g b];
f=gcf(); f.color_map=cmap;
plot3d1()
f.color_map=get(sdf(),"color_map");
```

See Also

autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap , color , getcolor

Name

Compound_properties — description of the Compound entity properties

Description

The Compound entity is a third of the graphics entities hierarchy. This entity defines interdependencies of the various graphics entities and their global visibility property.

parent:
: This property contains the handle of the parent. The parent of the text entity should be of the type `"Axes"` or `"Compound"`.

children:
: A vector containing the handles of all graphics objects children of the Compound These graphics objects can be of type `"Compound"`, `"Rectangle"`, `"Polyline"`, `"Patch"`, `"Segs"`, `"Arc"`, `"Grayplot"`,..

visible:
: This field contains the `visible` property value for the entity . It should be `"on"` or `"off"`. By default, value is `"on"` where graphics entities children of the Compound are drawn according to their visibility property. If `"off"` all children of Compound are not displayed on the screen.

user_data:
: This field can be use to store any scilab variable in the figure data structure, and to retreive it.

See Also

glue , unglue , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

contour — level curves on a 3D surface

```
contour(x,y,z,nz,[theta,alpha,leg,flag,ebox,zlev])
contour(x,y,z,nz,<opt_args>)
```

Parameters

x,y
: two real row vectors of size n1 and n2.

z
: real matrix of size (n1,n2), the values of the function or a Scilab function which defines the surface `z=f(x,y)`.

nz
: the level values or the number of levels.

  -
  : If `nz` is an integer, its value gives the number of level curves equally spaced from zmin to zmax as follows:

    ```
    z= zmin + (1:nz)*(zmax-zmin)/(nz+1)
    ```

    Note that the `zmin` and `zmax` levels are not drawn (generically they are reduced to points) but they can be added with

    ```
    [im,jm] = find(z == zmin);     // or zmax
    plot2d(x(im)',y(jm)',-9,"000")
    ```

  -
  : If `nz` is a vector, `nz(i)` gives the value of the ith level curve. Note that it can be useful in order to see `zmin` and `zmax` level curves to add an epsilon tolerance: `nz=[zmin+%eps,..,zmax-%eps]`.

<opt_args>
: a sequence of statements `key1=value1, key2=value2`, ... where keys may be `theta`,`alpha`,`leg`,`flag`, `ebox`,`zlev` (see below). In this case, the order has no special meaning.

theta, alpha
: real values giving in degree the spherical coordinates of the observation point.

leg
: string defining the captions for each axis with @ as a field separator, for example "X@Y@Z".

flag
: a real vector of size three `flag=[mode,type,box]`.

  mode
  : string representation mode.

mode=0:
the level curves are drawn on the surface defined by (x,y,z).

mode=1:
the level curves are drawn on a 3D plot and on the plan defined by the equation z=zlev.

mode=2:
the level curves are drawn on a 2D plot.

type
an integer (scaling).

type=0
the plot is made using the current 3D scaling (set by a previous call to `param3d`, `plot3d`, `contour` or `plot3d1`).

type=1
rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument `ebox`.

type=2
rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data.

type=3
3d isometric with box bounds given by optional `ebox`, similarily to `type=1`

type=4
3d isometric bounds derived from the data, to similarily`type=2`

type=5
3d expanded isometric bounds with box bounds given by optional `ebox`, similarily to `type=1`

type=6
3d expanded isometric bounds derived from the data, similarily to `type=2`

box
an integer (frame around the plot).

box=0
nothing is drawn around the plot.

box=1
unimplemented (like box=0).

box=2
only the axes behind the surface are drawn.

box=3
a box surrounding the surface is drawn and captions are added.

box=4
a box surrounding the surface is drawn, captions and axes are added.

ebox
used when `type` in `flag` is 1. It specifies the boundaries of the plot as the vector `[xmin,xmax,ymin,ymax,zmin,zmax]`.

zlev
real number.

Description

`contour` draws level curves of a surface z=f(x,y). The level curves are drawn on a 3D surface. The optional arguments are the same as for the function `plot3d` (except `zlev`) and their meanings are the same. They control the drawing of level curves on a 3D plot. Only `flag(1)=mode` has a special meaning.

mode=0
: the level curves are drawn on the surface defined by (x,y,z).

mode=1
: the level curves are drawn on a 3D plot and on the plan defined by the equation z=zlev.

mode=2
: the level curves are drawn on a 2D plot.

You can change the format of the floating point number printed on the levels by using `xset("fpf",string)` where `string` gives the format in C format syntax (for example `string="%.3f"`). Use `string=""` to switch back to default format and Use `string=" "` to suppress printing.

Usually we use `contour2d` to draw levels curves on a 2D plot.

Enter the command `contour()` to see a demo.

Examples

```
t=linspace(-%pi,%pi,30);
function z=my_surface(x,y),z=x*sin(x)^2*cos(y),endfunction

contour(t,t,my_surface,10)
// changing the format of the printing of the levels
xset("fpf","%.1f")
xbasc()
contour(t,t,my_surface,10)
// 3D
xbasc()
z=feval(t,t,my_surface);
plot3d(t,t,z);contour(t,t,z+0.2*abs(z),20,flag=[0 2 4]);
//
```

See Also

contour2d , plot3d

Authors

J.Ph.C.

Name

contour2d — level curves of a surface on a 2D plot

```
contour2d(x,y,z,nz,[style,strf,leg,rect,nax])
contour2d(x,y,z,nz,<opt_args>)
```

Parameters

x,y
: two real row vectors of size n1 and n2: the grid.

z
: real matrix of size (n1,n2), the values of the function on the grid or a Scilab function which defines the surface `z=f(x,y)`.

nz
: the level values or the number of levels.

  If `nz` is an integer, its value gives the number of level curves equally spaced from zmin to zmax as follows:

  ```
  z= zmin + (1:nz)*(zmax-zmin)/(nz+1)
  ```

  Note that the `zmin` and `zmax` levels are not drawn (generically they are reduced to points) but they can be added with

  ```
  [im,jm] = find(z == zmin); // or zmax
  plot2d(x(im)',y(jm)',-9,"000")
  ```

  If `nz` is a vector, `nz(i)` gives the value of the ith level curve.

<opt_args>
: This represents a sequence of statements `key1=value1,key2=value2`,... where `key1`, `key2,...` can be one of the following: style, leg, rect, nax, strf or axesflag and frameflag (see plot2d)

style,strf,leg,rect,nax
: see plot2d. The argument `style` gives the dash styles or colors which are to be used for level curves. It must have the same size as the number of levels.

Description

`contour2d` draws level curves of a surface `z=f(x,y)` on a 2D plot. The values of `f(x,y)` are given by the matrix `z` at the grid points defined by `x` and `y`.

You can change the format of the floating point number printed on the levels by using `xset("fpf",string)` where `string` gives the format in C format syntax (for example `string="%.3f"`). Use `string=""` to switch back to default format and Use `string=" "` to suppress printing. This last feature is useful in conjunction with legends to display the level numbers in a legend and not directly onto the level curves as usual (see Examples).

The optional arguments `style,strf,leg,rect,nax`, can be passed by a sequence of statements `key1=value1, key2=value2`, ... where keys may be `style`,`strf`,`leg`,`rect`,`nax`. In this case, the order has no special meaning.

Use `contour` to draw levels curves on a 3D surface.

Examples

```
contour2d(1:10,1:10,rand(10,10),5,rect=[0,0,11,11])
// changing the format of the printing of the levels
xset("fpf","%.2f")
clf()
contour2d(1:10,1:10,rand(10,10),5,rect=[0,0,11,11])

// now an example with level numbers drawn in a legend
// Caution there are a number of tricks...
x = linspace(0,4*%pi,80);
z = cos(x')*cos(x);
clf(); f=gcf();
xset("fpf"," ")  // trick 1: this implies that the level numbers are not
                 //          drawn on the level curves
f.color_map=jetcolormap(7);
// trick 2: to be able to put the legend on the right without
//          interfering with the level curves use rect with a xmax
//          value large enough
contour2d(x,x,z,-0.75:0.25:0.75,frameflag=3,rect=[0,0,5*%pi,4*%pi])
// trick 3: use legends (note that the more practical legend function
//          will not work as soon as one of the level is formed by 2 curves)
legends(string(-0.75:0.25:0.75),1:7,"lr");
xtitle("Some level curves of the function cos(x)cos(y)")
```

See Also

contour , contour2di , plot2d

Authors

J.Ph.C.

Name

contour2di — compute level curves of a surface on a 2D plot

```
[xc,yc]=contour2di(x,y,z,nz)
```

Parameters

x,y
: two real row vectors of size n1 and n2: the grid.

z
: real matrix of size (n1,n2), the values of the function.

nz
: the level values or the number of levels.

  If `nz` is an integer, its value gives the number of level curves equally spaced from zmin to zmax as follows:

```
z= zmin + (1:nz)*(zmax-zmin)/(nz+1)
```

  Note that the `zmin` and `zmax` levels are not drawn (generically they are reduced to points) but they can be added with

```
[im,jm] = find(z == zmin);     // or zmax
plot2d(x(im)',y(jm)',-9,"000")
```

  If `nz` is a vector, `nz(i)` gives the value of the ith level curve.

xc,yc
: vectors of identical sizes containing the contours definitions. See below for details.

Description

`contour2di` computes level curves of a surface `z=f(x,y)` on a 2D plot. The values of `f(x,y)` are given by the matrix `z` at the grid points defined by `x` and `y`.

`xc(1)` contains the level associated with first contour path, `yc(1)` contains the number `N1` of points defining this contour path and (`xc(1+(1:N1))`, `yc(1+(1:N1))`) contain the coordinates of the paths points. The second path begin at `xc(2+N1)` and `yc(2+N1)` and so on.

Examples

```
[xc,yc]=contour2di(1:10,1:10,rand(10,10),5);
k=1;n=yc(k);c=1;
while k+yc(k)<size(xc,'*')
  n=yc(k);
```

```
  plot2d(xc(k+(1:n)),yc(k+(1:n)),c)
  c=c+1;
  k=k+n+1;
end
```

See Also

contour , fcontour , fcontour2d , contour2d , plot2d , xset

Authors

J.Ph.C.

Name

contourf — filled level curves of a surface on a 2D plot

```
contourf(x,y,z,nz,[style,strf,leg,rect,nax])
```

Parameters

x,y
: two real row vectors of size n1 and n2: the grid.

z
: real matrix of size (n1,n2), the values of the function.

nz
: the level values or the number of levels.

  - If `nz` is an integer, its value gives the number of level curves equally spaced from zmin to zmax as follows:

    ```
    z= zmin + (1:nz)*(zmax-zmin)/(nz+1)
    ```

    Note: that the `zmin` and `zmax` levels are not drawn (generically they are reduced to points) but they can be added with

    ```
    [im,jm] = find(z == zmin);     // or zmax
    plot2d(x(im)',y(jm)',-9,"000")
    ```

  - If `nz` is a vector, `nz(i)` gives the value of the ith level curve.

style,strf,leg,rect,nax
: see `plot2d`. The argument `style` gives the colors which are to be used for level curves. It must have the same size as the number of levels.

Description

`contourf` paints surface between two consecutives level curves of a surface `z=f(x,y)` on a 2D plot. The values of `f(x,y)` are given by the matrix `z` at the grid points defined by `x` and `y`.

You can change the format of the floating point number printed on the levels by using `xset("fpf",string)` where `string` gives the format in C format syntax (for example `string="%.3f"`). Use `string=""` to switch back to default format.

Enter the command `contourf()` to see a demo.

Examples

```
contourf(1:10,1:10,rand(10,10),5,1:5,"011"," ",[0,0,11,11])

function z=peaks(x,y)
x1=x(:).*.ones(1,size(y,'*'));
y1=y(:)'.*.ones(size(x,'*'),1);
z =  (3*(1-x1).^2).*exp(-(x1.^2) - (y1+1).^2) ...
   - 10*(x1/5 - x1.^3 - y1.^5).*exp(-x1.^2-y1.^2) ...
   - 1/3*exp(-(x1+1).^2 - y1.^2)
endfunction

function z=peakit()
x=-4:0.1:4;y=x;z=peaks(x,y);
endfunction

z=peakit();

levels=[-6:-1,-logspace(-5,0,10),logspace(-5,0,10),1:8];
m=size(levels,'*');
n = fix(3/8*m);
r = [(1:n)'/n; ones(m-n,1)];
g = [zeros(n,1); (1:n)'/n; ones(m-2*n,1)];
b = [zeros(2*n,1); (1:m-2*n)'/(m-2*n)];
h = [r g b];
xset('colormap',h);
xset('fpf',' ');
xbasc();
contourf([],[],z,[-6:-1,-logspace(-5,0,10),logspace(-5,0,10),1:8],0*ones(1,m))

xset('fpf','');
xbasc();
contourf([],[],z,[-6:-1,-logspace(-5,0,10),logspace(-5,0,10),1:8]);
```

See Also

contour, fcontour, fcontour2d, contour2di, plot2d, xset

Authors

J.Ph.C.

Name

coolcolormap — cyan to magenta colormap

```
cmap=coolcolormap(n)
```

Parameters

n
: integer >= 3, the colormap size.

cmap
: matrix with 3 columns `[R,G,B]`.

Description

`coolcolormap` computes a colormap with `n` colors varying from cyan to magenta.

Examples

```
f = scf();
plot3d1();
f.color_map = coolcolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

coppercolormap — black to a light copper tone colormap

```
cmap=coppercolormap(n)
```

Parameters

n
: integer >= 3, the colormap size.

cmap
: matrix with 3 columns `[R,G,B]`.

Description

`coppercolormap` computes a colormap with `n` colors varying from black to a light copper tone.

Examples

```
f = scf();
plot3d1();
f.color_map = coppercolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

copy — copy a graphics entity.

```
copy(h)
copy(h,h_axes)
h_copy=copy(h)
```

Parameters

h
: a handle, the handle of the entity to copy.

h_axes
: a handle, the handle of the parent entity for the destination. It should be an axes entity.

h_copy
: a handle, the handle on the entity copied.

Description

This routine can be used to make a copy of a graphics entity with all its propeties'values, it returns the handle on this new entity. `h_axes` omitted, graphics entity is cloned and it's copied in the same parent axes entity.

Examples

```
subplot(211);a1=gca();
plot2d()
e=gce();
p1=e.children(1);
p2=copy(p1);p2.data(:,2)=p2.data(:,2)-0.5;
subplot(212);a2=gca();
a2.data_bounds=a1.data_bounds;
copy(p1,a2);
```

See Also

get , set , delete , move , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

delete — delete a graphic entity and its children.

```
delete(h)
delete(h,"callback")
delete()
delete("all")
```

Parameters

h
: a handle, the handle of the entity to delete. `h` can be a vector of handles, in which case all objects identified by h(i) will be deleted.

"all"
: string keyword (optional).

Description

This routine can be used to delete a graphics entity identified by the handle given as argument. In this case, All children of this graphics entity will be deleted. Without any argument `delete` removes the current entity. With `"all"` argument it deletes all the entities of the current figure.

The `"callback"` argument is not yet handled.

Examples

```
subplot(211);
t=1:10;plot2d(t,t.^2),
subplot(223);
plot3d();
subplot(224);
plot2d();
xfrect(1,0,3,1);
a=get("current_axes")
delete(); //delete the graphics object newly created
delete(a.children); //delete all children of the current axes
delete(a); //delete the axes
delete("all"); //delete all the graphics objects of the figure
```

See Also

get , set , copy , move , is_handle_valid , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

dragrect — Drag rectangle(s) with mouse

```
[final_rect,btn]=dragrect(initial_rect)
```

Parameters

initial_rect
: 4 4xn matrix containing the initial rectangles definition. Each column contains [x_left; y_top; width; height]. If only one rectangle is present the initial_rect may also be a vector.

final_rect
: :a rectangle defined by [x_left, y_top, width, height]

btn
: :an integer, the number of the mouse button clicked

Description

`dragrect` drags one or more rectangles anywhere on the screen. The 4xn matrix rect defines the rectangles. Each column of `initial_rect` must contain the initial rectangle position as [left;top;width;height] values. When a button is clicked `dragrect` returns the final rectangles definition in `final_Rect`.

Examples

```
xsetech(frect=[0,0,100,100])
r=dragrect([10;10;30;10])
xrect(r)
```

See Also

xrect , xrects , xclick , xgetmouse

Name

draw — draw an entity.

```
draw()
draw(h)
```

Parameters

h

a handle, the handle on an entity to draw. h can be a vector of handles, in which case all objects identified by h(i) will be drawn.

Description

This function can be used to draw entities specified by h even if its parent figure immediate_drawing property is "off". If no argument is specified, the current object is drawn. Note that the window must not be resized ; if not, those objects are hidden back.

Examples

```
subplot(212)
a=gca();
 plot2d
drawlater

 subplot(211)
 plot2d1 // default drawing mode

 e=gce();
 draw(e.children(2)) // draw a single polyline of the second axes

 e.children(1).visible='off'; // We can choose to make a line invisible

 draw(e) // draw Compound and its children <=> draw all the visible polyline



 drawnow // ...now!

 e.children(1).visible='on';
```

See Also

get , set , drawnow , drawlater , graphics_entities

Authors

Djalel ABDEMOUCHE, F.Leray

Name

drawaxis — draw an axis

```
drawaxis([options])
// options: x,y,dir,sub_int,fontsize,format_n,seg,textcolor,ticscolor,tics
```

Parameters

dir=string
: used to specify the tics direction. `string` can be chosen among 'u','r','d','l' and 'l' is the default value. the values 'u','r','d','l' stands respectively for up, right, down, left

tics=string
: A flag which describes how the tics are given. `string` can be chosen among 'v','r', and 'i', and, 'v' is the default value

x,y
: two vectors which give tics positions.

val= string matrix
: A string matrix, which, when given, gives the string to be drawn along the axis at tics positions.

fontsize=int
: specifies the fontsize to use for displaying values along the axis. Default value is `-1` which stands for current fontsize

format_n=string
: format to use for displaying numbers along the axis

seg= 1 or 0
: A flag which controls the display of the base segment of the axis (default value is 1).

sub_int=integer
: an integer which gives the number of sub-intervals to draw between large tics.

textcolor=integer
: specify the color to use for displaying values along the axis. Default value is `-1` which stands for current color.

ticscolor=integer
: specify the color to use for tics drawing. Default value is `-1` which stands for current color.

Description

`drawaxis` is used to draw an axis in vertical or horizontal direction. the direction of the axis is given by dir `dir = 'u'` or `'d'` gives a horizontal axis with tics going up (`'u'`) or down (`'d'`). `dir = 'r'` or `'l'` give a vertical axis with tics going right (`'r'`) or left (`'l'`).

`x` and `y` give the axis tics positions. If the axis is horizontal then `y` must be a scalar or can be omitted and `x` is a Scilab vector. The meaning of `x` is controlled by `tics`.

If `tics='v'` then `x` gives the tics positions along the x-axis.

If `tics='r'` then `x` must be of size 3. `x=[xmin,xmax,n]` and `n` gives the number of intervals.

If `tics='i'` then `x` must be of size 4, `x=[k1,k2,a,n]`. then `xmin=k1*10^a`, `xmax=k2*10^a` and `n` gives the number of intervals

If `y` is omitted then the axis will be positioned at the top of the frame if `dir='u'` or at the bottom if `dir='d'`

By default, numbers are drawn along the axis. They are drawn using a default format which can be changed with `format_n`. It is also possible to display given strings and not numbers, this is done if `val` is provided. The size of `val` must match the number of tics.

Examples

```
plot2d(1:10,1:10,1,"020")
// horizontal axis
drawaxis(x=2:7,y=4,dir='u',tics='v')
// horizontal axis on top of the frame
drawaxis(x=2:7,dir='u',tics='v')
// horizontal axis at the bottom of the frame
drawaxis(x=2:7,dir='d',tics='v')

// horizontal axis given by a range
drawaxis(x=[2,7,3],y=4,dir='d',tics='r')

// vertical axis
drawaxis(x=4,y=2:7,dir='r',tics='v')
drawaxis(x=2,y=[2,7,3],dir='l',tics='r')
drawaxis(y=2:7,dir='r',tics='v')
drawaxis(y=2:7,dir='l',tics='v')

// horizontal axis with strings displayed at tics positions
drawaxis(x=2:7,y=8,dir='u',tics='v',val='A'+string(1:6));
// vertical axis with strings displayed at tics positions
drawaxis(x=8,y=2:7,dir='r',tics='v',val='B'+string(1:6));

// horizontal axis given with a 'i' range.
drawaxis(x=[2,5,0,3],y=9,dir='u',tics='i');
drawaxis(x=9,y=[2,5,0,3],dir='r',tics='i',sub_int=5);

// horizontal axis again
drawaxis(x=2:7,y=4,dir='u',tics='v',fontsize=10,textcolor=9,ticscolor=7,seg=0,s
```

Authors

J.Ph.C.

Name

drawlater — makes axes children invisible.

```
drawlater()
```

Description

This function can be used not to display immediatly onto the `current figure` the next created graphics objects - i.e. by calling high level functions such as plotting functions or setting properties to existant objects. The `immediate_drawing` property of the current figure is set to `'off'` in order to postpon the next drawings.

It can specially be used with the `drawnow` function.

To enable back the `immediate_drawing` for the current figure, you can use `drawnow` function.

Warning : note that between drawlater and drawnow calls, the current figure may have changed. Therefore, this must be used carefully.

Examples

```
//Example :  one axes / one figure
drawlater();
xfarc(.25,.55,.1,.15,0,64*360);
xfarc(.55,.55,.1,.15,0,64*360);
xfrect(.3,.8,.3,.2);
xfrect(.2,.7,.5,.2);
xfrect(.32,.78,.1,.1);
xfrect(.44,.78,.14,.1);
xfrect(-.2,.4,1.5,.8);
xstring(0.33,.9,"A Scilab Car");
a=get("current_axes");
a.children(1).font_size=4;
a.children(1).font_style=4;
a.children(1).background=5;
a.children(3).background=8;
a.children(4).background=8;
a.children(5).background=17;
a.children(6).background=17;
a.children(7).background=25;
a.children(8).background=25;
xclick();drawnow();

//Example 2:: two axes / one figure

subplot(212)
a=gca();
drawlater // what will be present in this axes will be displayed later
plot2d // draw these axes and children later...

subplot(211) // Warning: we change the axes
plot2d1 // default drawing mode

draw(a) // ...axes only is visible providing you not redraw the window
drawnow() // all is visible
```

See Also

get , set , drawnow , graphics_entities

Authors

Djalel ABDEMOUCHE, F.Leray

Name

drawnow — draw hidden graphics entities.

```
drawnow()
```

Description

Considering the current figure, this function can be used to draw the hidden graphics entities and all its chidren, that may have been postponed by a previous call to `drawlater`. The `immediate_drawing` property of the `current figure` is set to `"on"` .

It can specially be used with the `drawlater` function.

Examples

```
f=get("current_figure") // handle of the current figure

drawlater()
subplot(221);
t=1:10;plot2d(t,t.^2)
subplot(222);
x=0:1:7;plot2d(x,cos(x),2)
subplot(234);
plot2d(t,cos(t),3);
subplot(235);
plot2d(x,sin(2*x),5);
subplot(236);
plot2d(t,tan(2*t));

draw(gca()); //draw the current axes and its children
draw(f.children([3 4])); // draw the specified axes and their children
drawnow(); // draw the current figure
```

See Also

get , set , drawlater , graphics_entities

Authors

Djalel ABDEMOUCHE, F.Leray

Name

edit_curv — interactive graphic curve editor

```
[x,y,ok,gc] = edit_curv(y)
[x,y,ok,gc] = edit_curv(x,y)
[x,y,ok,gc] = edit_curv(x,y,job)
[x,y,ok,gc] = edit_curv(x,y,job,tit)
[x,y,ok,gc] = edit_curv(x,y,job,tit,gc)
```

Parameters

x
: vector of x coordinates

y
: vector of y coordinates

job
: a character string formed by one to three of the characters 'a','x','y'

  'a'
  : to add points to the edited curve

  'x'
  : to modify x coordinates of the edited curve points

  'y'
  : to modify y coordinates of the edited curve points

tit
: a vector of three character strings which give the curve legend

gc
: a list of graphic window parameters: `gc=list(rect,nax)`

  rect
  : bounds of the graphics (see plot2d for details)

  nax
  : graduation parameters (see plot2d for details)

ok
: indicator if ok==%t user as returned with 'ok' menu else user as returned with 'abort' menu : list (graphical objects created under `edit_curv`

Description

`edit_curv` is an interactive graphic curve editor. To add a new point simply click at the desired location, the added point will be connected to the nearest end-point. to move a point click on it, drag the mouse to the new position and click to set the new position

Authors

Serge Steer ; ; ; ; ;

Name

errbar — add vertical error bars on a 2D plot

```
errbar(x,y,em,ep)
```

Parameters

x,y,em,ep
: four matrices of the same size.

Description

`errbar` adds vertical error bars on a 2D plot. `x` and `y` have the same meaning as in `plot2d`. `em(i,j)` and `ep(i,j)` stands for the error interval on the value `y(i,j)`: `[y(i,j)-em(i,j),y(i,j)+ep(i,j)]`.

Enter the command `errbar()` to see a demo.

Examples

```
t=[0:0.1:2*%pi]';
y=[sin(t) cos(t)]; x=[t t];
plot2d(x,y)
errbar(x,y,0.05*ones(x),0.03*ones(x))
```

See Also

plot2d

Authors

J.Ph.C.

Name

eval3d — values of a function on a grid

```
[z]=eval3d(fun,x,[y])
```

Parameters

fun
: function accepting vectors as arguments.

x,y
: 2 vectors of size (1,n1) and (1,n2). (default value for `y` : `y=x`).

z
: matrix of size (n1,n2).

Description

This function returns a matrix `z(n1,n2)`. `z(i,j)=fun(x(i),y(j))`. If the function `fun` doesn't accept arguments of type vector use the primitive `feval`.

Examples

```
  x=-5:5;y=x;
  deff('[z]=f(x,y)',['z= x.*y']);
  z=eval3d(f,x,y);
  plot3d(x,y,z);
//
  deff('[z]=f(x,y)',['z= x*y']);
  z=feval(x,y,f);
  plot3d(x,y,z);
```

See Also

feval

Authors

Steer S.; ; ;

Name

eval3dp — compute facets of a 3D parametric surface

```
[Xf,Yf,Zf]=eval3dp(fun,p1,p2)
```

Parameters

Xf,Yf,Zf
: matrices of size (4,n-1*m-1). `Xf(:,i)` ,`Yf(:,i)` and `Zf(:,i)` are respectively the x-axis, y-axis and z-axis coordinates of the 4 points of the ith four sided facet.

fun
: a Scilab function.

p1
: a vector of size `n`.

p2
: a vector of size `m`.

Description

`eval3dp` computes a four sided facets representation of a 3D parametric surface defined by the function `fun`. `fun(p1,p2)` computes the x-axis ,y-axis and z-axis coordinates of the corresponding points on the surface, as [x(i),y(i),z(i)]=fun(p1(i),p2(i)). This is used for efficiency.

Examples

```
p1=linspace(0,2*%pi,10);
p2=linspace(0,2*%pi,10);
deff("[x,y,z]=scp(p1,p2)",["x=p1.*sin(p1).*cos(p2)";..
                           "y=p1.*cos(p1).*cos(p2)";..
                           "z=p1.*sin(p2)"])
[Xf,Yf,Zf]=eval3dp(scp,p1,p2);
plot3d(Xf,Yf,Zf)
```

See Also

genfac3d , plot3d

Name

event handler functions — Prototype of functions which may be used as event handler.

```
envent_handler_function(win,x,y,ibut)
```

Parameters

win
: window number where the event had occured.

x
: X coordinate in pixels of the mouse pointer when the events occured.

y
: Y coordinate in pixels of the mouse pointer when the events occured.

ibut
: number corresponding to the event type.

Description

When the event handler mode is enable, Scilab will call the specified event handler function each time an event occures in the graphic window. The event handler function must comply with the above prototype. Each time an event occured, the function is called and its four parameters are set accordingly to the window number, mouse position and type of event.

The `ibut` parameter takes one of the following value depending on event type:

ibut==0
: Left mouse button has been pressed

ibut==1
: Middle mouse button has been pressed

ibut==2
: Right mouse button has been pressed

ibut==3
: Left mouse button has been clicked

ibut==4
: Middle mouse button has been clicked

ibut==5
: Right mouse button has been clicked

ibut==10
: Left mouse button has been double-clicked

ibut==11
: Middle mouse button has been double-clicked

ibut==12
: Right mouse button has been double-clicked

ibut==-5
Left mouse button has been released

ibut==-4
Middle mouse button has been released

ibut==-3
Right mouse button has been released

ibut==-1
mouse pointer has moved

ibut > =32
key with ascii code ascii(ibut) has been pressed

ibut < =-32
key with ascii code ascii(-ibut) has been released

ibut > =1000+32
key with ascii code ascii(ibut-1000) has been pressed while CTRL key pressed

ibut==-1000
graphic window has been closed

For example, let say that the name of the event handler function is fooHandler for window number 0. A left click in the window at position [100,150] (in pixels) will be equivalent as calling fooHandler(0, 100, 150, 3).

See figure_properties or seteventhandler for information on how to specify the event_handler name.

Examples

```
function my_eventhandler(win,x,y,ibut)
   if ibut==-1000 then return,end
   [x,y]=xchange(x,y,'i2f')
    xinfo(msprintf('Event code %d at mouse position is (%f,%f)',ibut,x,y))
endfunction
plot2d()
fig = gcf() ;
fig.event_handler = 'my_eventhandler' ;
fig.event_handler_enable = "on" ;
//now:
//   -  move the mouse over the graphic window
//   -  press and release keys shifted or not with Ctrl pressed or not
//   -  press button, wait a little release
//   -  press and release button
//   -  double-click button


fig.event_handler_enable = "off" ; //suppress the event handler
```

See Also

figure_properties , seteventhandler , xgetmouse , xclick

Authors

Jean-Baptiste Silvy

Name

fac3d — 3D plot of a surface (obsolete)

```
fac3d(x,y,z,[theta,alpha,leg,flag,ebox])
fac3d1(x,y,z,[theta,alpha,leg,flag,ebox])
```

Description

These functions are obsolete and have been replaced by `plot3d` and `plot3d1`.

See Also

plot3d , plot3d1

Name

fchamp — direction field of a 2D first order ODE

```
fchamp(f,t,xr,yr,[arfact,rect,strf])
fchamp(f,t,xr,yr,<opt_args>)
```

Parameters

f
: An external (function or character string) or a list which describes the ODE.

  -
  : It can be a function name `f`, where `f` is supposed to be a function of type `y=f(t,xy[p1,..pn])`. `f` returns a column vector of size 2, `y`, which gives the value of the direction field `f` at point `xy=[x,y]` and at time `t`.

  -
  : It can also be an object of type list, `list(f,P1,..Pn)` where `f` is a function of type `y=f(t,xy,p1,..pn)` and `Pi` gives the value of the parameter `pi`.

t
: The selected time.

xr,yr
: Two row vectors of size n1 and n2 which define the grid on which the direction field is computed.

<opt_args>
: This represents a sequence of statements `key1=value1,key2=value2`, ... where `key1`, `key2,...` can be one of the following: arfact, rect, strf (see below).

arfact,rect,strf
: Optional arguments, see `champ`.

Description

`fchamp` is used to draw the direction field of a 2D first order ODE defined by the external function `f`. Note that if the ODE is autonomous, argument `t` is useless, but it must be given.

Enter the command `fchamp()` to see a demo.

Examples

```
deff("[xdot] = derpol(t,x)",..
        ["xd1 = x(2)";..
         "xd2 = -x(1) + (1 - x(1)**2)*x(2)";..
         "xdot = [ xd1 ; xd2 ]"])
xf= -1:0.1:1;
yf= -1:0.1:1;
fchamp(derpol,0,xf,yf)
clf()
fchamp(derpol,0,xf,yf,1,[-2,-2,2,2],"011")
```

See Also

champ , champ1

Authors

J.Ph.C.

Name

fcontour — level curves on a 3D surface defined by a function

```
fcontour(xr,yr,f,nz,[theta,alpha,leg,flag,ebox,zlev])
fcontour(xr,yr,f,nz,<opt_args>)
```

Description

This function is obsolete. It is now included in the contour function.

Authors

J.Ph.C.

Name

fcontour2d — level curves of a surface defined by a function on a 2D plot

```
fcontour2d(xr,yr,f,nz,[style,strf,leg,rect,nax])
fcontour2d(xr,yr,f,nz,<opt_args>)
```

Description

This function is obsolete. It is now included in the contour2d function.

Authors

J.Ph.C.

Name

fec — pseudo-color plot of a function defined on a triangular mesh

```
fec(x,y,triangles,func,<opt_args>)
fec(x,y,triangles,func,[strf,leg,rect,nax,zminmax,colminmax,colout,mesh])
```

Parameters

x,y
: two vectors of size `n`, `(x(i),y(i))` gives the coordinates of node `i`

func
: a vector of size `n` : `func(i)` gives the value at node `i` of the function for which we want the pseudo-color plot.

triangles
: is a `[Ntr,5]` matrix. Each line of `triangles` specifies a triangle of the mesh `triangle(j) = [number,node1,node2,node3,flag]`. `node1,node2,node3` are the number of the nodes which constitutes the triangle. number is the number of the triangle and flag is an integer not used in the fec function

<opt_args>
: This represents a sequence of statements `key1=value1, key2=value2`,... where `key1`, `key2,...` can be one of the following: strf, leg, rect, nax, zminmax, colminmax, colout, mesh (see plot2d for the 4 first ones).

strf,leg,rect,nax
: see plot2d

zminmax
: vector with 2 components [zmin zmax] (useful in particular for animation)

colminmax
: vector of 2 positives integers [colmin colmax]

colout
: vector of 2 integers [under_min_col upper_max_col]

mesh
: boolean scalar, default value %f (must be true if you want also display the mesh)

Description

This function is the good one to draw linear triangular finite element solutions or simply to display a function defined on a triangulation. The color interpolation is done through software computation and so it is not too fast.

The function colorbar may be used to see the color scale (see the example section).

The `zminmax` argument gives the z values associated with the first and the last color (of the current colormap). More exactly if the colormap have nc colors and if we note *dz = (zmax-zmin)/nc*, then the part of the triangulation where *zmin + (i-1)dz <= z < zmin + i dz* is filled with the color *i*). By default *zmin = min(func)* and *zmax = max(func)*. If you want to do an animation with func values that varie in time, take for zmin and zmax the global minimum and maximum or something close.

The `colout` argument lets the user choosing the colors for the 2 extremes regions *{func < zmin}* and *{func > zmax}*, `under_min_col` and `upper_max_col` may be equal (independantly) to:

-1
: in this case the same color than in the neighbouring zone is used (CAUTION: you don't see that the corresponding limit is crossed), this is the default case.

0
: in this case the extreme region is not painting at all.

k
: (k being a valid index to a color of the current colormap) the extreme region is painting in color k.

If you don't want to use the complete colormap you may use the `colminmax` argument with *1 <= colmin < colmax <= nc* (nc being the number of colors of the current colormap) so as to use only the [colmin,colmax] sub-part of the colormap. (by default all the colors of the colormap are used).

See the demo files `demos/fec`:

`fec.ex1` is a simple demo file in which a mesh and a function on that mesh is completely built in Scilab syntax

`fec.ex2` is an example for which the mesh and the function value where computed by an external mesh builder (amdba type mesh) and an external program. A set of macros (provided in file `macros.sci`) can be used to read the data files in Scilab and plot the results.

Examples

```
// define a mini triangulation (4 vertices, 2 triangles)
x = [0 1 0 -1];
y = [0 0 1  1];
T = [1 1 2 3 1;
     2 3 4 1 1];
z = [0 1 0 -1];  // values of the func at each vertices
xbasc()
xset("colormap",jetcolormap(64))
subplot(1,2,1)
   colorbar(-1,1)
   fec(x,y,T,z,strf="040",mesh=%t)
   xtitle("fec example (with the mesh)")
subplot(1,2,2)
   colorbar(-1,1)
   fec(x,y,T,z,strf="040")  // rmq: mesh=%f by default
   xtitle("fec example (without the mesh)")
xselect()


// this example shows the effect of zminmax and uses the
// previous example datas (you have to execute the it before)
xbasc()
xset("colormap",jetcolormap(64))
colorbar(-0.5,0.5)  // be careful colorbar must be set by hands !
fec(x,y,T,z,strf="040", zminmax=[-0.5 0.5], mesh=%t)
xtitle("fec example : using zminmax argument")
xselect()


// this example shows the effect of zminmax and colout. It uses
// also the datas of the first example (you have to execute the it before)
xbasc()
xset("colormap",jetcolormap(64))
```

```
subplot(2,2,1)
   colorbar(-0.5,0.5)
   fec(x,y,T,z,strf="040", zminmax=[-0.5 0.5], colout=[0 0], mesh=%t)
   xtitle("fec example : using zminmax and colout =[0 0]")
subplot(2,2,2)
   colorbar(-0.5,0.5)
   fec(x,y,T,z,strf="040", zminmax=[-0.5 0.5], colout=[32 32], mesh=%t)
   xtitle("fec example : using zminmax and colout =[32 32]")
subplot(2,2,3)
   colorbar(-0.5,0.5)
   fec(x,y,T,z,strf="040", zminmax=[-0.5 0.5], colout=[-1 0], mesh=%t)
   xtitle("fec example : using zminmax and colout =[-1 0]")
subplot(2,2,4)
   colorbar(-0.5,0.5)
   fec(x,y,T,z,strf="040", zminmax=[-0.5 0.5], colout=[0 -1], mesh=%t)
   xtitle("fec example : using zminmax and colout =[0 -1]")
xselect()


// this example shows a feature from colminmax:
// playing with 2 colormaps for 2 subplots. It
// uses also the data of the first example.
xbasc()
xset("colormap",[hotcolormap(64);jetcolormap(64)])
subplot(1,2,1)
   colorbar(-1,1,[1 64])
   fec(x,y,T,z,strf="040", colminmax=[1 64], mesh=%t)
   xtitle("fec using the hot colormap")
subplot(1,2,2)
   colorbar(-1,1,[65 128])
   fec(x,y,T,z,strf="040", colminmax=[65 128], mesh=%t)
   xtitle("fec using the jet colormap")
xselect()
```

See Also

colorbar , Sfgrayplot , Sgrayplot

Name

fec_properties — description of the fec entities properties

Description

The Fec entity is a leaf of the graphics entities hierarchy. It represents 2D finite elements plots (see `fec`, `Sgrayplot`).

parent:
: This property contains the handle of the parent. The parent of the fec entity should be of the type `"Axes"` or `"Compound"`.

children:
: This property contains a vector with the `children` of the handle. However, `Fec` handles currently do not have any `children`.

visible:
: This field contains the `visible` property value for the entity . It should be `"on"` or `"off"` . By default, the plot is visible, the value's property is `"on"`. If `"off"` the plot is not drawn on the screen.

data:
: This is a three column matrix `[x,y,f]`, where `x(i)` and `y(i)` are the coordinates of the `i`'th node. `f(i)` is the value associated to the node `i`.

triangles:
: This is a five column matrix `[tn,n1,n2,n3,flag]`. `tn(j)` is the triangle number. `n1(j)`, `n2(j)` and `n3(j)` are the index of the nodes which constitute the triangle. (`flag(j)` is not used).

z_bounds:
: This vector of size 2, `[zmin,zmax]`, gives the z values associated with the first and the last color (of the current colormap). More exactly if the colormap have `nc` colors and if we note `dz = (zmax-zmin)/nc`, then the part of the triangulation where `zmin + (i-1)dz <= z < zmin + i dz` is filled with the color `i`). By default the `z_bounds` property value is `[0,0]`. In this case, the `zmin` and `zmax` are autommatically set to the minumum and maximum of the `func` argument.

outside_color:
: This vector of size 2, `[cmin, cmax]`, defines the color used when nodes values are outside the `z_bounds = [zmin,zmax]` interval. When node values are lower than `zmin` the color with index `cmin` is used. When node values are greater than `zmax` the color with index `cmax` is used. By default, the `outside_color` property value is `[0,0]`. In this case, `cmin` and `cmax` are automatically set to the two bounds of the colormap. If `cmin` or `cmax` are negative, then values outside z_bounds interval are not displayed, they appear transparent.

color_range:
: This vector of size 2, `[rmin, rmax]`, allows to use only a part of the colormap for display. `rmin` and `rmax` stand for colormap indices. If they are both greater than 1, then the actual colormap used to display the fec entity is `colormap(rmin:rmax)` where `colormap` is the colormap of the parent figure. By default, the `color_range` property value is `[0,0]`. In this case, the whole colormap is used.

line_mode:
: If `"on"`, the wireframe enclosing triangles is drawn. If `"off"`, only the inside of triangles are drawn.

foreground:
: This color index specifies the color of the mesh. If `line_mode` property is `"on"`, the wireframe is drawn using this color.

clip_state:
This field contains the `clip_state` property value for the fec. It should be :

- `"off"` this means that the fec is not clipped.
- `"clipgrf"` this means that the fec is clipped outside the Axes box.
- `"on"` this means that the fec is clipped outside the rectangle given by property clip_box.

clip_box:
This field is to determinate the `clip_box` property. By Default its value should be an empty matrix if clip_state is "off". Other cases the vector `[x,y,w,h]` (upper-left point width height) defines the portions of the fec to display, however `clip_state` property value will be changed.

user_data:
This field can be use to store any scilab variable in the fec data structure, and to retreive it.

Examples

```
x=-10:10; y=-10:10;m =rand(21,21);
Sgrayplot(x,y,m);
a=get("current_axes");
f=a.children.children(1)
f.data(:,3)=(1:size(f.data,1))';
a.parent.color_map=hotcolormap(64);
```

See Also

set , get , delete , fec , Sgrayplot , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

fgrayplot — 2D plot of a surface defined by a function using colors

```
fgrayplot(x,y,f,[strf,rect,nax])
fgrayplot(x,y,f,<opt_args>)
```

Parameters

x,y
: real row vectors.

f
: external of type `y=f(x,y)`.

<opt_args>
: This represents a sequence of statements `key1=value1, key2=value2`,... where `key1`, `key2,...` can be one of the following: rect, nax, strf or axesflag and frameflag (see `plot2d`).

strf,rect,nax
: see `plot2d`.

Description

`fgrayplot` makes a 2D plot of the surface given by `z=f(x,y)` on a grid defined by `x` and `y`. Each rectangle on the grid is filled with a gray or color level depending on the average value of `z` on the corners of the rectangle.

Enter the command `fgrayplot()` to see a demo.

Examples

```
t=-1:0.1:1;
deff("[z]=my_surface(x,y)","z=x**2+y**2")
fgrayplot(t,t,my_surface,rect=[-2,-2,2,2])
```

See Also

grayplot , plot2d , Sgrayplot , Sfgrayplot

Authors

J.Ph.C.

Name

figure_properties — description of the graphics figure entity properties

Description

The figure entity is the top level of the graphics entities hierarchy. This entity contain a number of properties designed to control many aspects of displaying Scilab graphics objects. These properties fall into two categories. Properties that contain information about figure itself and others related to set default values for the children creation.

Figure properties:

children:
: This handles represent the vector of the figure's children . Note that all figure children are of type `"Axes"`. Also keep in mind that, when creating a figure entity (using scf command), an `Axes` entity is simultaneously built too.

figure_style:
: The value of this field defines the figure style. Scince Scilab 5.0, old graphic mode has been disable. This property is kept for compatibility but can not be modified.

figure_position:
: This field contains the position in pixel of the graphic window on the screen. This is a vector `[x,y]` defining the position of the upper-left corner of the window. The position `[0,0]` is the upper-left corner of the screen.

  The initial position of graphic windows is taken from the default figure entity (see gdf). The only exception is when default figure `figure_position` value is `[-1,-1]`. In this case, the initial position of graphic windows is automatically set by the windowing system.

figure_size:
: This property controls the size in pixel of the screen's graphics window. The size is the vector `[width,height]`.

axes_size:
: Used to Specifies the size in pixel of the virtual graphics window. The size is the vector `[width,height]` . The virtual graphic window should be bigger than the part really visible on the screen. This property could not be modified if the figure is docked with other elements.

auto_resize:
: This property determines if graphics window is resized. If the value is `"on"` then the `axes_size` property is equaled to the `figure_size` and the axes children are zoomed accordingly. If the value is `"off"` that indicate that `axes_size` cannot be resized when `figure_size` is changed.

viewport:
: Postion of the visible part of graphics in the panner.

figure_name:
: This field contains the name of the figure. This name is a character string displayed at the top of the graphics_window. The name can contain a single substring `%d` which will be replaced by the `figure_id`. No other instance of the `%` character is allowed inside the name.

figure_id:
: This field contains the identifier of the figure. This is an integer number which is set at figure creation and cannot be changed after.

info_message:
: This character string set the text displayed in the info bar of the graphic window.

color_map:
Property which defines the colormap used by this figure. The colormap is a `m` by `3` matrix. `m` is the number of colors. Color number i is given as a 3-uple `R`, `G`, `B` corresponding respectively to red, green and blue intensity between 0 and 1.

pixmap:
This property controls the pixmap status of a Graphic Window. If this property value is `"off"` the graphics are directly displayed on the screen. If it is `"on"` the graphics are done on a pixmap and are sent to the graphics window with the command `show_pixmap()`.

pixel_drawing_mode:
This field defines the bitwise operation used to draw the pixel on the screen. The default mode is `copy` what is to say the pixel is drawn as required. More generally the bitwise operation is performed between the color of the source pixel and the color of the pixel already present on the screen. Operations are : `"clear"`, `"and"`, `"andReverse"` , `"copy"`, `"andInverted"` , `"noop"` , `"xor"` , `"or"` , `"nor"` , `"equiv"` , `"invert"` , `"orReverse"` , `"copyInverted"` , `"orInverted"` , `"nand"` , `"set"` ,

immediate_drawing:
This property controls the figure display. Its value can be `"on"` (default mode) or `"off"`. It is used to delay a huge succession of graphics commands (implying several drawings or redrawings). Note that, when using `drawlater` or `drawnow` commands, it affects the property value of the current figure (which is respectively turned to `'off'` or `'on'`).

background:
This property controls the figure window background color. It takes as value an index relative to the current colormap.

event_handler
A character string. The name of the Scilab function which is intended to handle the events. Not that setting an empty string will disable the event handler. For more information about event handler functions see the event handler functions help.

event_handler_enable
Enable or disable the event handler. Its value must be either "on" or "off".

user_data:
This field can be use to store any scilab variable in the figure data structure, and to retreive it.

tag:
This field can be use to store a character string generally used to identify the control. It allows to give it a "name". Mainly used in conjontion with findobj().

Children's default values:

visible:
This field ules if the contents of the figure as to be redrawn. Its value should be `"on"` or `"off"`.

rotation_style:
This field is related to the "3D Rot" button. It takes `unary` as value (default) in the aim to rotate only selected 3D plot. In the other case its value can be `multiple` : all 3D plots are rotated.

Note on default values :

All these listed properties and fields inherit from default values stored in a figure model. These default values can be seen and changed. To do so, use the `get("default_figure")` command : it returns a graphic handle on the figure model. Note that no graphic window is created by this command. The next created figures will inherit from this model (see example 2 below).

Examples

```
 lines(0) // disables vertical paging
//Example 1
 f=get("current_figure") //get the handle of the current figure :
                                          //if none exists, create a figure an
 f.figure_position
 f.figure_size=[200,200]
 f.background=2
 f.children  // man can see that an Axes entity already exists
 delete(f);
 f=gcf(); // macro shortcut <=> f=get("current_figure")
 f.pixmap = "on" // set pixmap status to on
 plot2d() // nothing happens on the screen...
 show_pixmap() // ...display the pixmap on screen
 //Example 2 : default_figure settings
 df=get("default_figure") // get the default values (shortcut is gdf() )
 // Let's change the defaults...
 df.color_map=hotcolormap(128)
 df.background= 110 // set background toa kind of yellow (Note that we are us
 scf(122); // creates new figure number 122 with the new default
 plot2d()
 scf(214);
 t=-%pi:0.3:%pi;
 plot3d(t,t,sin(t)'*cos(t),35,45,'X@Y@Z',[15,2,4]);
```

See Also

lines , set , get , scf , gcf , gdf , gca , gda , axes_properties , show_pixmap , clear_pixmap , , hotcolormap , event handler functions

Authors

Djalel ABDEMOUCHE

Name

fplot2d — 2D plot of a curve defined by a function

```
fplot2d(xr,f,[style,strf,leg,rect,nax])
fplot2d(xr,f,<opt_args>)
```

Parameters

xr
: vector.

f
: external of type `y=f(x)` i.e. a scilab function or a dynamically linked routine referred to as a string.

style,strf,leg,rect,nax
: see `plot2d`

<opt_args>
: see `plot2d`

Description

`fplot2d` plots a curve defined by the external function `f`. The curve is approximated by a piecewise linear interpolation using the points `(xr(i),f(xr(i))`. The values of f(x) are obtained by `feval(xr,f)`.

Enter the command `fplot2d()` to see a demo.

Examples

```
deff("[y]=f(x)","y=sin(x)+cos(x)")
x=[0:0.1:10]*%pi/10;
fplot2d(x,f)
clf();
fplot2d(1:10,'parab')
```

See Also

plot2d , feval , paramfplot2d

Authors

J.Ph.C.

Name

fplot3d — 3D plot of a surface defined by a function

```
fplot3d(xr,yr,f,[theta,alpha,leg,flag,ebox])
fplot3d(xr,yr,f,<opt_args>)
```

Parameters

xr
: row vector of size n1.

yr
: row vector of size n2.

f
: external of type z=f(x,y).

theta,alpha,leg,flag,ebox
: see `plot3d`.

<opt_args>
: see `plot3d`.

Description

`fplot3d` plots a surface defined by the external function `f` on the grid defined by `xr` and `yr`.

Enter the command `fplot3d()` to see a demo.

Examples

```
deff('z=f(x,y)','z=x^4-y^4')
x=-3:0.2:3 ;y=x ;
clf() ;fplot3d(x,y,f,alpha=5,theta=31)
```

See Also

plot3d

Authors

J.Ph.C.

Name

fplot3d1 — 3D gray or color level plot of a surface defined by a function

```
fplot3d1(xr,yr,f,[theta,alpha,leg,flag,ebox])
fplot3d1(xr,yr,f,<opt_args>)
```

Parameters

xr
: row vector of size n1.

yr
: row vector of size n2.

f
: external of type z=f(x,y).

theta,alpha,leg,flag,ebox
: see `plot3d1`.

<opt_args>
: see `plot3d`.

Description

`fplot3d1` plots a 3D gray or color level plot of a surface defined by the external function `f` on the grid defined by `xr` and `yr`.

Enter the command `fplot3d1()` to see a demo.

Examples

```
deff('z=f(x,y)','z=x^4-y^4')
x=-3:0.2:3 ;y=x ;
clf() ;fplot3d1(x,y,f,alpha=5,theta=31)
```

See Also

plot3d1

Authors

J.Ph.C.

Name

gca — Return handle of current axes.

```
a = gca()
```

Parameters

a
: handle, the handle of the current axes entity.

Description

This routine returns the handle of the current axes for the current figure.

Examples

```
subplot(211)
a=gca() //get the current axes
a.box="off";
t=-%pi:0.3:%pi;plot3d(t,t,sin(t)'*cos(t),80,50,'X@Y@Z',[5,2,4]);
subplot(212)
plot2d(); //simple plot
a=gca() //get the current axes
a.box="off";
a.x_location="middle";
a.parent.background=4;
delete(gca()) // delete the current axes
xdel(0) //delete a graphics window
```

See Also

gda , gcf , gce , get , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

gce — Get current entity handle.

```
e = gce()
```

Parameters

e
: handle, the handle of the current entity.

Description

This routine returns the handle of the last created (and still existent) entity.

Examples

```
a=gca() //get the handle of the newly created axes
a.data_bounds=[1,1;10,10];
a.axes_visible = 'on' ;
for i=1:5
  xfrect(7-i,9-i,3,3);
  e=gce();
  e.background=i;
 end
delete(); // delete current entity
delete(gce()); // delete current entity
delete(gcf()); // delete current figure
```

See Also

gcf , gca , get , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

gcf — Return handle of current graphic window.

```
h = gcf()
```

Parameters

h
: handle.

Description

This routine returns the handle of the current graphic window.

Examples

```
f=gcf(); // Create a figure
f.figure_size= [610,469]/2;
f.figure_name= "Foo";

f=figure(); // Create a figure
h=uicontrol(f,"style","listbox","position", [10 10 150 160]); // Create a l
set(h, "string", "item 1|item 2|item3");// fill the list
set(h, "value", [1 3]); // select item 1 and 3 in the list
gcf()

scf(0); // Make graphic window 0 the current figure
gcf()  // Return the graphic handle of the current figure

figure(f) // Make window f the current figure
gcf() // Return the graphic handle of the current figure
```

See Also

gdf , gca , gce , get , scf , set , graphics_entities , uicontrol

Authors

Serge Steer, INRIA

Name

gda — Return handle of default axes.

```
a = gda()
a = get("default_axes")
```

Parameters

a
: handle, the handle of the default axes.

Description

The default axes is a graphic entity which is never drawn. It is used as a reference for the axes properties default values. These default properties values are used to initialize new axes inside figures.

The `gda` function returns the handle on the default axes. The user can use this handle to set or get the axes properties default values.

Note that an equivalent default graphic entity exists for figure entities too (see `gdf`).

Examples

```
 a=gda() // get the handle of the model axes
 // set its' properties
 a.box="off";
 a.foreground=2;
 a.labels_font_size=3;
 a.labels_font_color=5;
 a.sub_tics=[5 5 3];
 a.x_location="top";

 //now used the model axes for drawing
 subplot(211) //create an axes entity
 plot2d()

// set other model's properties
 a.background=color('gray')
 a.grid=[5 5 5];
 subplot(212)
 t=0:0.1:5*%pi;
 plot2d(sin(t),cos(t))


 set(a,"default_values",1); // return to the  default values of the model
                            // see sda() function
 clf()
 plot2d(sin(t),cos(t))
```

See Also

gdf , sda , sdf , clf , gca , get , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

gdf — Return handle of default figure.

```
f = gdf()
f = get("default_figure")
```

Parameters

f
: handle, the handle of the default figure.

Description

The default figure is a graphic entity which is never drawn. It is used as a reference for the figure properties default values. These default properties values are used to initialize new figures.

The `gdf` function returns the handle on the default figure. The user can use this handle to set or get the figure properties default values.

Note that an equivalent default graphic entity exists for axes entities too (see `gda`).

Examples

```
f=gdf() // get the handle of the model figure
// setting its' properties
f.background=7;
f.figure_name="Function gdf()";
f.figure_position=[-1 100];
f.auto_resize="off";
f.figure_size=[300 461];
f.axes_size=[600 400];
plot2d() //create a figure
scf(1);
plot3d() //create a second figure
set(f,"default_values",1); // return to the  default values of figure's mode
                           // see sdf() function
scf(2);
plot2d()
```

See Also

gda , sdf , sda , gcf , get , scf , set , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

ged — Scilab Graphic Editor

```
ged(action, fignum)
```

Parameters

action
: Real: action to be executed on graphic window given by `fignum`:

  - 1: Select window `fignum` as current figure.
  - 2: Redraw window `fignum`.
  - 3: Clear window `fignum`.
  - 4: Ask the user to select a graphic entity to copy.
  - 5: Paste last graphic entity copied using action `4`.
  - 6: Ask the user to select a graphic entity and then move it.
  - 7: Ask the user to select the graphic entity to delete.
  - 8: Start a GUI to edit window properties.
  - 9: Start a GUI to edit current axes properties.
  - 10: Start an entity picker to select a graphic object and edit it using Graphic Editor GUI.
  - 11: Stop the entity picker.

fignum
: Real: Graphic window number, the figure to edit.

Description

`ged` starts Scilab Graphic Editor on figure number `fignum` and execute action given by `action`.

Authors

V.C.

Name

genfac3d — compute facets of a 3D surface

```
[xx,yy,zz]=genfac3d(x,y,z,[mask])
```

Parameters

xx,yy,zz
: matrices of size (4,n-1xm-1). `xx(:,i)` ,`yy(:,i)` and `zz(:,i)` are respectively the x-axis, y-axis and z-axis coordinates of the 4 points of the ith four sided facet.

x
: x-axis coordinates vector of size m.

y
: y-axis coordinates vector of size n.

z
: matrix of size (m,n). `z(i,j)` is the value of the surface at the point (x(i),y(j)).

mask
: boolean optional matrix with same size as `z` used to select the entries of `z` to be represented by facets.

Description

`genfac3d` computes a four sided facets representation of a 3D surface `z=f(x,y)` defined by `x`, `y` and `z`.

Examples

```
t=[0:0.3:2*%pi]'; z=sin(t)*cos(t');
[xx,yy,zz]=genfac3d(t,t,z);
plot3d(xx,yy,zz)
```

See Also

eval3dp , plot3d

Name

geom3d — projection from 3D on 2D after a 3D plot

```
[x,y]=geom3d(x1,y1,z1)
```

Parameters

x1,y1,z1
: real vectors of the same size (points in 3D).

x,y
: real vectors of the same size as `x1`, `y1` and `z1`.

Description

After having used a 3D plot function such as `plot3d`, `plot3d1` or `param3d`, `geom3d` gives the mapping between a point in 3D space `(x1(i),y1(i),z1(i))` and the corresponding point `(x(i),y(i))` in the projected 2D plan. Then all the 2D graphics primitives working on `(x,y)` can be used for superposition on the 3D plot.

Examples

```
deff("[z]=surface(x,y)","z=sin(x)*cos(y)")
t=%pi*(-10:10)/10;
// 3D plot of the surface
fplot3d(t,t,surface,35,45,"X@Y@Z")
// now (t,t,sin(t).*cos(t)) is a curve on the surface
// which can be drawn using geom3d and xpoly
[x,y]=geom3d(%pi/2,0,surface(%pi/2,0))
```

Authors

J.Ph.C.

Name

get — Retrieve a property value from a graphics entity or an User Interface object.

```
h=get(prop)
val=get(h,prop)
val=h.prop
```

Parameters

h
: handle, the handle of the entity to retrieve a property. `h` can be a vector of handles, in which case get returns the property value for all objects contained in h. `h` can also be *0* to get the root object properties.

prop
: character string name of the property.

val
: value of the property.

Description

This routine can be used to retrieve the value of a specified property from a graphics entity or a GUI object. In this case it is equivalent to use the dot operator on an handle. For exemple, `get(h,"background")` is equivalent to `h.background`.

Property names are character strings. To get the list of all existing properties see graphics_entities or uicontrol for User Interface objects.

`get` function can be also called with only a property as argument. In this case, the property must be one of the following:

current_entity or hdl
: returns a handle on the lastly created (and still existent) entity. `get('current_entity')` and `get('hdl')` are equivalent to gce.

current_figure
: returns a handle on the current graphic figure. `get('current_figure')` is equivalent to gcf.

current_axes
: returns a handle on the current axes entity. `get('current_axes')` is equivalent to gca.

default_figure
: returns a handle on the default figure entity. `get('default_figure')` is equivalent to gdf.

default_axes
: returns a handle on the default axes entity. `get('default_axes')` is equivalent to gda.

figures_id
: returns a row vector containing ids of all opened graphic figures. `get('figures_id')` is equivalent to winsid.

Examples

```
// for graphics entities
  clf()
```

```
  // simple graphics objects
  subplot(121);
  x=[-.2:0.1:2*%pi]';
  plot2d(x-2,x.^2);
  subplot(122);
  xrect(.2,.7,.5,.2);
  xrect(.3,.8,.3,.2);
  xfarc(.25,.55,.1,.15,0,64*360);
  xfarc(.55,.55,.1,.15,0,64*360);
  xstring(0.2,.9,"Example <<A CAR>>");

  h=get("current_entity") //get the newly object created
  h.font_size=3;

  f=get("current_figure") //get the current figure
  f.figure_size
  f.figure_size=[700 500];
  f.children
  f.children(2).type
  f.children(2).children
  f.children(2).children.children.thickness=4;

  a=get("current_axes") //get the current axes
  a.children.type
  a.children.foreground //get the foreground color of a set of graphics objec
  a.children.foreground=9;

// for  User Interface objects
 h=uicontrol('string', 'Button'); // Opens a window with a  button.
 p=get(h,'position'); // get the geometric aspect of the button
 disp('Button width: ' + string(p(3))); // print the width of the button
 close(); // close figure
```

See Also

uicontrol , root_properties , graphics_entities , set

Authors

Djalel ABDEMOUCHE

Name

get_figure_handle — get a figure handle from its id

```
f = get_figure_handle(figure_id)
```

Parameters

figure_id
: Integer, id of the figure to retrieve.

f
: Handle of the corresponding figure.

Description

`get_figure_handle` function allows to retrieve the handle of a graphic figure knowing its id. If a figure with the specified id exists the function returns it. Otherwise is returns an empty matrix.

Examples

```
// create some figures
scf(0);
scf(5);
scf(12);

// get handle on the figure with id 5
f5 = get_figure_handle(5);
// current figure remains the one with id 12
gcf()
// get a non existing figure
f42 = get_figure_handle(42);
```

See Also

set , get , gcf , scf , graphics_entities

Authors

Jean-Baptiste Silvy INRIA

Name

getcolor — opens a dialog to show colors in the current colormap

```
c=getcolor(title,[cini])
c=getcolor()
```

Parameters

title
: string, dialog title.

cini
: initial selected color id. Default value is 1.

c
: selected color id or [] if the selection is cancelled.

Description

`getcolor` opens a window displaying the palette of the current colormap. The user can click on a color to show its id and RGB values. `getcolor` returns the id of the selected color or [] if the "Cancel" button has been clicked or the window closed.

See Also

color , colormap , getmark , getfont

Name

getfont — dialog to select font **. Obsolete function.**

```
[fId,fSize]=getfont()
[fId,fSize]=getfont(str)
fnt=getfont()
fnt=getfont(str)
fnt=getfont(S=str,font=[fId,fSize])
```

Parameters

str
: character (e.g. "a")

fId
: integer, the number of the selected font

fSize
: integer, the size of the selected font

fnt
: vector [fId,fSize]

Description

This function designed to work with the xset function **is also obsolete**. Use the property editor ged instead.

`getfont` opens a graphic window to select a font. User has to select a font and a size clicking on the corresponding displayed character. Killing a keyboard key changes the displayed character.

Examples

```
[fId,fSize]=getfont();
xset("font",fId,fSize)
plot2d(0,0,rect=[0 0 10 10],axesflag=0)
xstring(5,5,"string")
```

See Also

ged, text_properties

Name

getlinestyle — dialog to select linestyle. **Obsolete function.**

```
k=getlinestyle()
```

Parameters

k
: integer, selected linestyle or [] if the "Cancel" button has been clicked.

Description

This function designed to work with the xset function **is also obsolete**. Use the property editor ged instead.

`getlinestyle` opens a graphic window to select a line style.

Examples

```
x=0:0.1:10;
plot2d(x,sin(x))
e=gce(); // store the Compound containing the plot
e.children(1).line_style = getlinestyle();
```

See Also

ged, set, segs_properties, segs_properties

Name

getmark — dialog to select mark (symbol). **Obsolete function**

```
[mark,mrkSize]=getmark()
```

Parameters

mark
: integer, the number of the selected mark

mrkSize
: integer, the size of the selected mark

Description

This function designed to work with the xset function **is also obsolete**. Use the property editor ged instead.

`getmark` opens a graphic window to select a mark (symbol).

Examples

```
x=0:0.1:10;
[mark,mrkSize]=getmark();
plot2d(x,sin(x),style=-mark);
clf();
plot2d(x,sin(x))
e=gce(); // store the Compound containing the plot
[mark,mrkSize]=getmark();
e.children(1).mark_style = mark;
```

See Also

ged, set, segs_properties, segs_properties

Name

getsymbol — dialog to select a symbol and its size. **Obsolete function**

```
c=getsymbol([title])
```

Parameters

title
: string, dialog title.

c
: vector of size 2 `[n,sz]`.

Description

This function designed to work with the xset function **is also obsolete**. Use the property editor ged instead.

`getsymbol` opens a dialog choice box with title `title` if given where the user can select a symbol and its size. `getsymbol` returns the id of the mark `n` and the id of its size `sz`.

See Also

ged, set, segs_properties, segs_properties

Name

glue — glue a set of graphics entities into an Compound.

```
glue(H)
h_agreg=glue(H)
```

Parameters

H
: a vector of handle.

h_agreg
: a handle, the handle on the Compound entity.

Description

Given a vector of handles, this function glue the corresponding entities in a single Compound and returns the handle on this new entity.

Examples

```
```

See Also

get , set , move , unglue , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

graduate — pretty axis graduations

```
[xi,xa,np]=graduate( xmi, xma,n1,n2)
[xi,xa,np]=graduate( xmi, xma)
```

Parameters

xmi,xma
: real scalars

n1, n2
: integers with default values 3,10

xi, xa
: :real scalars

np
: integer

Description

graduate looks for the minimum interval `[xi,xa]` and a number of tics `np` such that:

`xi <= xmi <= xma <= xa`

`xa - xi / np = k(10^n)`,`k` in `[1 3 5]` for an integer `n`

`n1 < np < n2`

Examples

```
y=(0:0.33:145.78)';
xbasc();plot2d1('enn',0,y)
[ymn,ymx,np]=graduate(mini(y),maxi(y))
rect=[1,ymn,prod(size(y)),ymx];
xbasc();plot2d1('enn',0,y,1,'011',' ',rect,[10,3,10,np])
```

See Also

xsetech , plot2d

Authors

S. Steer 1992;

Name

graphics_entities — description of the graphics entities data structures
new graphics — description of the graphics entities data structures

Description

In Scilab, graphics window and the drawing it contains are represented by hierchical entities. The hierachy top level is the Figure. Each Figure defines at least one child of type Axes. Each Axes entity contains a set of leaf entities which are the basic graphics objects like Polylines, Rectangles, Arcs, Segs,... It can also contain an Compound type which are recursive sets of entities. The main interest of the new graphic mode is to make property change easier. This new graphics'mode provides a set of high-level graphing routines (see set, get) used to control objects' properties such as data, coordinates and scaling, color and appearences without requiring to replay the initial graphics commands.

Graphics entities are associated to Scilab variables of type `handle`. The handle is a unique identifier which is associated to each instance of a created graphical entity. Using this handle, it will be possible to reach entities' properties through "set" and "get" routines. The handles are also used to manipulate graphics objects, to move them, to make copies or delete them.

Figure:
: The figure entity is the top level of the graphics entities hierarchy. This entity defines the parameters for the figure itself as well as the parameters' default values for the children creation. The figure children are the Axes entities.

  The handle on the current figure (the figure used where the drawing are sent) may be got using `get("current_figure")` and it may be set using `set("current_figure",h)`, where `h` is either a handle on a figure or a `figure_id` in this last case if the figure does not already exists , it is created

  See figure_properties for details.

Axes:
: The Axes entity is the second level of the graphics entities hierarchy. This entity defines the parameters for the change of coordinates and the axes drawing as well as the parameters' default values for its children creation. See axes_properties for details. The handle on the current Axes may be got using `get("current_axes")`.

Compound:
: The Compound entity is just a vector of children and with a single property (visibility property). It is used to glue a set of entities together.

  See glue, unglue and Compound_properties functions.

Axis:
: The Axis entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for axis scaling and appearance.

  See axis_properties for details.

Polyline:
: The polyline entity is a leaf of the graphics entities hierarchy. It defines 2D and 3D polylines and polylines extensions drawing properties.

  See polyline_properties for details.

Arc:
: The Arc entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for ellipses and part of ellipses.

See arc_properties for details.

Rectangle:
The Rectangle entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for rectangles and filled rectangles.

See rectangle_properties for details.

Surface:
The Surface entity is a leaf of the graphics entities hierarchy. It has sub types `Fac3d` or `Plot3d`. This entity defines the parameters for 3d surface plots.

See surface_properties for details.

Fec:
The Fec entity is a leaf of the graphics entities hierarchy. It represents 2D finite elements plots .

See fec_properties for details.

Grayplot:
The Grayplot entity is a leaf of the graphics entities hierarchy. It represents 2D plots of surface using colors and images.

See grayplot_properties for details.

Matplot:
The Matplot entity is a leaf of the graphics entities hierarchy. It represents 2D plots using integer matrices.

See Matplot_properties for details.

Segs:
The Segs entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for a set of colored segments or colored arrows.

See segs_properties for details.

Champ:
The Champ entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for a 2D vector field.

See champ_properties for details.

Text:
The Text entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for string drawing.

See text_properties for details.

Label:
The Labels entity are children of the `Axes` graphics entity. This entity defines the parameters for the 3 x,y and z labels and title drawn on a graphics window.

See label_properties for details.

Legend:
The Legend entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for legends drawn below `plot2dx` graphs. This entity requires further developments.

See legend_properties for details.

Examples

```
//Play this example line per line

scf() //create a figure in entity mode

//get the handle on the Figure entity and display its properties
f=get("current_figure")
a=f.children // the handle on the Axes child
x=(1:10)';  plot2d(x,[x.^2 x.^1.5])
e=a.children //Compound of 2 polylines

p1=e.children(1) //the last drawn polyline properties
p1.foreground=5;  // change the polyline color
e.children.thickness=5; // change the thickness of the two polylines

delete(e.children(2))

move(e.children,[0,30]) //translate the polyline

a.axes_bounds=[0 0 0.5 0.5];

subplot(222) //create a new Axes entity
plot(1:10);
a1=f.children(1); //get its handle
copy(e.children,a1); //copy the polyline of the first plot in the new Axes
a1.data_bounds=[1 0;10 100]; //change the Axes bounds

set("current_figure",10) //create a new figure with figure_id=10
plot3d() //the drawing are sent to figure 10
set("current_figure",f) //make the previous figure the current one
plot2d(x,x^3)  //the drawing are sent to the initial figure
```

See Also

set , get , move , draw , delete , object_editor , plot , surf

Name

graphics_fonts — description of fonts used in graphic figures

Description

Some Graphic entities such as Text, Axes, Label or Legend entities display one or more character strings in graphic figures. The appearance of the displayed strings can be modified by specifying different fonts and character sizes.

Changing font

Fonts used in graphic figures are selected from a set of fonts called loaded fonts. In other words, the loaded fonts are the ones currenlty available in graphic figures. The list of these fonts can be obtained using the `xlfont` function without argument. By default, Scilab contains a set of 11 loaded fonts. This set can be modified and extended using the `xlfont` function with a font name as argument. The added font can either be loaded from a file or be one of the system. To know the list of fonts available on the system use the `xlfont('AVAILABLE_FONTS')` command. For more information on how to manipulate fonts see xlfont.

Here is the list of the 11 default fonts.

Font number	**Font Family**	**Bold**	**Italic**
0	Monospaced	No	No
1	ScilabSymbols	No	No
2	Serif	No	No
3	Serif	No	Yes
4	Serif	Yes	No
5	Serif	Yes	Yes
6	SansSerif	No	No
7	SansSerif	No	Yes
8	SansSerif	Yes	No
9	SansSerif	Yes	Yes
10	SansSerif	Yes	Yes

The font used by a graphic entities can be modified with its `font_style` property. This is a positive integer referecing one of the loaded fonts. Its value must be between 0, referecing the first font, and the number of loaded fonts minus one, referencing the last font.

The `fractional_font` controls the font anti-aliasing. Its value can be either `'on'` or `'off'`. If its value is `'on'` the font is smoothed, otherwise it's not.

Changing character size

The text size of a graphic entity is modified using the `font_size` property. It is a scalar specifying the displayed character size.

The Scilab character size is different from the Java size. Here is the equivalence between the two scales.

Scilab Size	**Java Size**
0	8
1	10
2	12

3	14
4	18
5	24
6	30
7	36
8	42
9	48
10	54

The character size might not be an integer. In this case, the result depends on the entities `fractional_font` property. If `fractional_font` property is `'on'` then the displayed font size is interpolated between the two closest integer. For example, a font size of 2.5 displays characters with a Java size of 13. If `fractional_font` property is `'off'` then the displayed font size is truncated to its integer part. For example, a font size of 2.5 displays characters using a Java size of 12.

See Also

xlfont , graphics_entities

Name

graycolormap — linear gray colormap

```
cmap=graycolormap(n)
```

Parameters

n
: integer >= 1, the colormap size.

cmap
: matrix with 3 columns `[R,G,B]`.

Description

`graycolormap` computes a colormap with `n` gray colors varying linearly from black to white.

Examples

```
f = scf();
plot3d1();
f.color_map = graycolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap , xset

Name

grayplot — 2D plot of a surface using colors

```
grayplot(x,y,z,[strf,rect,nax])
grayplot(x,y,z,<opt_args>)
```

Parameters

x,y
: real row vectors of size n1 and n2.

z
: real matrix of size (n1,n2). `z(i,j)` is the value of the surface at the point (x(i),y(j)).

<opt_args>
: This represents a sequence of statements `key1=value1, key2=value2`,... where `key1`, `key2,...` can be one of the following: rect, nax, strf or axesflag and frameflag (see `plot2d` and `plot2d_old_version`).

strf,rect,nax
: see `plot2d`.

Description

`grayplot` makes a 2D plot of the surface given by `z` on a grid defined by `x` and `y`. Each rectangle on the grid is filled with a gray or color level depending on the average value of `z` on the corners of the rectangle. If z contains %nan values, the surounding rectangles are not displayed.

Enter the command `grayplot()` to see a demo.

Examples

```
x=-10:10; y=-10:10;m =rand(21,21);
grayplot(x,y,m,rect=[-20,-20,20,20])
t=-%pi:0.1:%pi; m=sin(t)'*cos(t);
clf()
grayplot(t,t,m)
```

See Also

fgrayplot , plot2d , Sgrayplot , Sfgrayplot

Authors

J.Ph.C.

Name

grayplot_properties — description of the grayplot entities properties

Description

The Grayplot entity is a leaf of the graphics entities hierarchy. It represents 2D plots of surface using colors and images (see `grayplot`, `Sgrayplot`, `fgrayplot` and `Sfgrayplot`).

parent:
: This property contains the handle of the parent. The parent of the grayplot entity should be of the type `"Axes"`.

children:
: This property contains a vector with the `children` of the handle. However, `grayplot` handles currently do not have any `children`.

visible:
: This field contains the `visible` property value for the entity . It should be `"on"` or `"off"` . By default, the plot is visible, the value's property is `"on"`. If `"off"` the plot is not drawn on the screen.

data:
: This field defines a `tlist` data structure of type `"grayplotdata"` composed of a row and column indices of each element : the x and y grid coordinates are contained respectively in `data.x` and `data.y`. The complementary field named `data.z` is the value of the surface at the point `(x(i),y(j))` (scaled mode) or the centered value of the surface defined between two consecutive `x(i)` and `y(j)` (direct mode). If `data_mapping` (see below) is set to `"scaled"`, the entire z data is used to perform a color interpolation whereas, if `data_mapping`'s value is `"direct"`, the last line and column of the z data indices are ignored and the color is determined directly in the colormap by the indices of the submatrix `data.z(1:$-1,1:$-1)`.

data_mapping:
: By default the value of this property is `"scaled"` : the indices of painting colors are proportional to the value `z` coordinates. In the other case, the property takes as value `"direct"` where the plot is a grayplot and the indices of painting colors are given by the data (see above).

clip_state:
: This field contains the `clip_state` property value for the grayplot. It should be :

  - `"off"` this means that the grayplot is not clipped.
  - `"clipgrf"` this means that the grayplot is clipped outside the Axes box.
  - `"on"` this means that the grayplot is clipped outside the rectangle given by property clip_box.

clip_box:
: This field is to determinate the `clip_box` property. By Default its value should be an empty matrix if clip_state is "off". Other cases the vector `[x,y,w,h]` (upper-left point width height) defines the portions of the grayplot to display, however `clip_state` property value will be changed.

user_data:
: This field can be use to store any scilab variable in the grayplot data structure, and to retreive it.

Examples

```
m=5;n=5;
M=round(32*rand(m,n));
grayplot(1:m,1:n,M)

a=get("current_axes");
a.data_bounds=  [-1,-1;7,7]
h=a.children

h.data_mapping="direct";

// A 2D ploting of a matrix using colors
xbasc()
a=get("current_axes");
a.data_bounds=  [0,0;4,4];

b=5*ones(11,11); b(2:10,2:10)=4; b(5:7,5:7)=2;
Matplot1(b,[1,1,3,3])  ;

h=a.children
for i=1:7
 xclick(); // click the mouse to sets Matplot data
 h.data=h.data+4;
end
```

See Also

set , get , delete , grayplot , Matplot , Matplot1 , graphics_entities , Matplot_properties

Authors

Djalel ABDEMOUCHE & F.Leray

Name

graypolarplot — Polar 2D plot of a surface using colors

```
graypolarplot(theta,rho,z,[strf,rect])
```

Parameters

theta
: a vector with size n1, the discretization of the the angle in radian.

rho
: a vector with size n2, the discretization of the radius

z
: real matrix of size (n1,n2). `z(i,j)` is the value of the surface at the point (theta(i),rho(j)).

strf
: is a string of length 3 `"xy0"`.

  default
  : The default is `"030"`.

  x
  : controls the display of captions.

    x=0
    : no captions.

    x=1
    : captions are displayed. They are given by the optional argument `leg`.

  y
  : controls the computation of the frame.

    y=0
    : the current boundaries (set by a previous call to another high level plotting function) are used. Useful when superposing multiple plots.

    y=1
    : the optional argument `rect` is used to specify the boundaries of the plot.

    y=2
    : the boundaries of the plot are computed using min and max values of `x` and `y`.

    y=3
    : like `y=1` but produces isoview scaling.

    y=4
    : like `y=2` but produces isoview scaling.

    y=5
    : like `y=1` but `plot2d` can change the boundaries of the plot and the ticks of the axes to produce pretty graduations. When the zoom button is activated, this mode is used.

    y=6
    : like `y=2` but `plot2d` can change the boundaries of the plot and the ticks of the axes to produce pretty graduations. When the zoom button is activated, this mode is used.

    y=7
    : like `y=5` but the scale of the new plot is merged with the current scale.

y=8
: like `y=6` but the scale of the new plot is merged with the current scale.

leg
: a string. It is used when the first character x of argument `strf` is 1. `leg` has the form `"leg1@leg2@...."` where `leg1`, `leg2`, etc. are respectively the captions of the first curve, of the second curve, etc. The default is `""`.

rect
: This argument is used when the second character y of argument `strf` is 1, 3 or 5. It is a row vector of size 4 and gives the dimension of the frame: `rect=[xmin,ymin,xmax,ymax]`.

Description

Takes a 2D plot of the surface given by `z` on a polar coordinate grid defined by `rho` and `theta`. Each grid region if filled with a gray or color level depending on the average value of `z` on the corners of the grid.

Examples

```
rho=1:0.1:4;theta=(0:0.02:1)*2*%pi;
z=30+round(theta'*(1+rho^2));
f=gcf();
f.color_map= hotcolormap(128);
clf();graypolarplot(theta,rho,z)
```

Name

havewindow — return scilab window mode

```
havewindow()
```

Description

returns %t if scilab has it own window and %f if not, i.e. if scilab has been invoked by "scilab -nw". (`nw` stands for "no-window".

Name

hist3d — 3D representation of a histogram

```
hist3d(f,[theta,alpha,leg,flag,ebox])
hist3d(list(f,x,y),[theta,alpha,leg,flag,ebox])
```

Parameters

mtx
: matrix of size (m,n) defining the histogram `mtx(i,j)=F(x(i),y(j))`, where `x` and `y` are taken as `0:m` and `0:n`.

list(mtx,x,y)
: where mtx is a matrix of size (m,n)defining the histogram `mtx(i,j)=F(x(i),y(j))`, with `x` and `y` vectors of size (1,n+1) and (1,m+1).

theta,alpha,leg,flag,ebox
: see `plot3d`.

Description

`hist3d` represents a 2d histogram as a 3D plot. The values are associated to the intervals `[x(i) x(i+1)[ X [y(i) y(i+1)[`.

Enter the command `hist3d()` to see a demo.

Examples

```
hist3d(10*rand(10,10));

Z = zeros(100,5);
A = abs(rand(40,5));
Z(1:40,:) = A;
scf();
hist3d(Z);

Index = find(Z==0);
Z(Index) = %nan;
scf();
hist3d(Z);

A = abs(rand(10,5));
Z(91:100,:) = A;
scf();
hist3d(Z);
```

See Also

histplot , plot3d

Authors

Steer S. & JPhilippe C.

Name

histplot — plot a histogram

```
histplot(n, data, <opt_args>)
histplot(x, data, <opt_args>)
```

Parameters

n
: positive integer (number of classes)

x
: increasing vector defining the classes (`x` may have at least 2 components)

data
: vector (datas to be analysed)

<opt_args>
: This represents a sequence of statements `key1=value1,key2=value2` ,... where `key1`, `key2,...` can be any optional plot2d parameter (`style,strf,leg, rect,nax, logflag,frameflag, axesflag`) or normalization. For this last one the corresponding value must be a boolean scalar (default value %t).

Description

This function plot an histogram of the `data` vector using the classes `x`. When the number `n` of classes is provided instead of `x`, the classes are choosen equally spaced and *x(1) = min(data) < x(2) = x(1) + dx < ... < x(n+1) = max(data)* with *dx = (x(n+1)-x(1))/n*.

The classes are defined by C1 = [x(1), x(2)] and Ci = (x(i), x(i+1)] for i >= 2. Noting Nmax the total number of `data` (Nmax = length(data)) and Ni the number of `data` components falling in Ci, the value of the histogram for x in Ci is equal to *Ni/(Nmax (x(i+1)-x(i)))* when `normalization` is true (default case) and else, simply equal to *Ni*. When normalization occurs the histogram verifies:

```
 x(n+1)
/
|   h(x) dx = 1,  when x(1)<=min(data) and max(data) <= x(n+1))
/
x(1)
```

Any plot2d (optional) parameter may be provided; for instance to plot an histogram with the color number 2 (blue if std colormap is used) and to restrict the plot inside the rectangle [-3,3]x[0,0.5], you may use `histplot(n,data, style=2, rect=[-3,0,3,0.5])`.

Enter the command `histplot()` to see a demo.

Examples

```
// example #1: variations around an histogram of a gaussian random sample
d=rand(1,10000,'normal');  // the gaussian random sample
clf();histplot(20,d)
clf();histplot(20,d,normalization=%f)
clf();histplot(20,d,leg='rand(1,10000,''normal'')',style=5)
```

```
clf();histplot(20,d,leg='rand(1,10000,''normal'')',style=16, rect=[-3,0,3,0.5])

// example #2: histogram of a binomial (B(6,0.5)) random sample
d = grand(1000,1,"bin", 6, 0.5);
c = linspace(-0.5,6.5,8);
xbasc()
subplot(2,1,1)
   histplot(c, d, style=2)
   xtitle("normalized histogram")
subplot(2,1,2)
   histplot(c, d, normalization=%f, style=5)
   xtitle("non normalized histogram")

// example #3: histogram of an exponential random sample
lambda = 2;
X = grand(100000,1,"exp", 1/lambda);
Xmax = max(X);
xbasc()
histplot(40, X, style=2)
x = linspace(0,max(Xmax),100)';
plot2d(x,lambda*exp(-lambda*x),strf="000",style=5)
legend(["exponential random sample histogram" "exact density curve"]);
```

See Also

hist3d , plot2d , dsearch

Name

hotcolormap — red to yellow colormap

```
cmap=hotcolormap(n)
```

Parameters

n
: integer >= 3, the colormap size.

cmap
: matrix with 3 columns `[R,G,B]`.

Description

`hotcolormap` computes a colormap with `n` hot colors varying from red to yellow.

Examples

```
f = scf();
plot3d1();
f.color_map = hotcolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

hsv2rgb — Converts HSV colors to RGB

```
[r,g,b] = hsv2rgb(h,s,v)
rgb = hsv2rgb(h,s,v)
[r,g,b] = hsv2rgb(hsv)
rgb = hsv2rgb(hsv)
```

Parameters

h
: a vector of size n. The "hue" values.

s
: a vector of size n. The "saturation" values.

v
: a vector of size n. The "value" values

hsv
: a n x 3 matrix. Each row contains a [hue saturation value] tripple.

r
: a column vector of size n. The associated "red" values.

g
: a column vector of size n. The associated "green" values.

b
: a column vector of size n. The associated "blue" values.

rgb
: a n x 3 matrix. Each row contains a [red green blue] tripple.

Description

The function hsv2rgb converts colormaps between the RGB and HSV color spaces. As hue varies from 0 to 1.0, the corresponding colors vary from red through yellow, green, cyan, blue, magenta, and back to red, so that there are actually red values both at 0 and 1.0. As saturation varies from 0 to 1.0, the corresponding colors (hues) vary from unsaturated (shades of gray) to fully saturated (no white component). As value, or brightness, varies from 0 to 1.0, the corresponding colors become increasingly brighter.

Examples

```
t=[0:0.3:2*%pi]'; z=sin(t)*cos(t');
plot3d1(t,t,z)
f=gcf();f.pixmap='on';
for h=0:0.1:1
  hsv=[h*ones(32,1) linspace(0,1,32)' 0.7*ones(32,1)];
  f.color_map=hsv2rgb(hsv);
  show_pixmap()
  xpause(10000)
end
for v=0:0.1:1
  hsv=[ones(32,1) linspace(0,1,32)' v*ones(32,1)];
```

```
    f.color_map=hsv2rgb(hsv);
    show_pixmap()
    xpause(10000)
  end
```

Authors

Serge Steer
INRIA

Name

hsvcolormap — Hue-saturation-value colormap

```
cmap = hsvcolormap(n)
```

Parameters

n
: integer >= 1, the colormap size.

cmap
: matrix with 3 columns `[R,G,B]`.

Description

`hsvcolormap` computes a colormap with ncolors. This colormap varies the hue component of the hsv color model. The colors begin with red, pass through yellow, green, cyan, blue, magenta, and return to red. The map is particularly useful for displaying periodic functions.

Examples

```
t=[0:0.1:2*%pi]'; z=sin(t)*cos(t');
f=gcf();f.color_map=hsvcolormap(64);
 plot3d1(t,t,z,35,45,"X@Y@Z",[-2,2,2])
```

Authors

Serge Steer
: INRIA

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

is_handle_valid — Check wether a set of graphic handles is still valid.

```
isValid = is_handle_valid(h)
```

Parameters

h
: Matrix of graphic handles

isValid
: Matrix of boolean with the same size as `h`

Description

`is_handle_valid` function tests wether a set of graphic handle is still valid. A valid handle is a handle which has not been deleted. The result, `isValid`, is a boolean matrix such as `isValid(i,j)` is true if `h(i,j)` is valid and false otherwise.

Examples

```
// check that current objects are valid
is_handle_valid([gcf(), gca(), gce()])

// create 11 polylines
plot([0:10; 0:10; 0:10], [0:10; 0:0.5:5; 0:2:20]);

// check polylines validity
axes = gca();
polylines = axes.children(1).children
is_handle_valid(polylines)

// delete some polylines
delete(polylines(3:7));
// print validity
is_handle_valid(polylines)
```

See Also

delete , graphics_entities

Authors

Jean-Baptiste Silvy INRIA

Name

isoview — set scales for isometric plot (do not change the size of the window)

```
isoview(xmin,xmax,ymin,ymax)
```

Parameters

xmin,xmax,ymin,ymax
: four real values

Description

This function is obsolete, use preferably the `frameflag=4` plot2d option which enable window resizing.

`isoview` is used to have isometric scales on the x and y axes. It does not change the size of the graphics window. The rectangle `xmin, xmax, ymin, ymax` will be contained in the computed frame of the graphics window. `isoview` set the current graphics scales and can be used in conjunction with graphics routines which request the current graphics scale (for instance `strf="x0z"` in `plot2d`).

Examples

```
t=[0:0.1:2*%pi]';
plot2d(sin(t),cos(t))
xbasc()
isoview(-1,1,-1,1)
plot2d(sin(t),cos(t),1,"001")
xset("default")

plot2d(sin(t),cos(t),frameflag=4)
```

See Also

square , xsetech

Authors

Steer S.

Name

jetcolormap — blue to red colormap

```
cmap=jetcolormap(n)
```

Parameters

n
: integer >= 3, the colormap size.

cmap
: matrix with 3 columns `[R,G,B]`.

Description

`jetcolormap` computes a colormap with `n` colors varying from blue, lightblue, green, yellow, orange then red

Examples

```
f = scf();
plot3d1();
f.color_map = jetcolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

label_properties — description of the Label entity properties

Description

The Label entity is a child of an `Axes` entity. When an Axes entity is built, the Title and Labels handles come with it and are part of the Axes properties. Therefore, the properties of those sub-objects are editable via the Axes handle (see `gca` and `gda`). This entity defines the parameters for label drawing:

parent:
: This property contains the handle of the parent. The parent of the label entity should be of type `"Axes"` .

  Note that, for now, Label entity is exclusively used in `title`, `x_label`, `y_label` and `z_label` building.

visible:
: This field contains the `visible` property value for the entity . It should be `"on"` or `"off"` .By default, the label is visible, the value's property is `"on"`. If `"off"` the label is not displayed on the screen.

text:
: The matrix containing the strings of the object. The rows of the matrix are displayed horizontally and the columns vertically.

font_foreground:
: This field contains the color used to display the label `text`. Its value should be a color index (relative to the current colormap).

foreground:
: This field contains the color used to display the line around the box if any. Its value should be a color index (relative to the current colormap).

background:
: This field contains the color used to fill the box if any. Its value should be a color index (relative to the current colormap).

fill_mode:
: This field takes the values `"on"` or `"off"`. If `"on"` a box is draw around the text with a line on its edge and a background.

font_style:
: Specifies the font used to display the label. This is a positive integer referecing one of the loaded fonts. Its value must be between 0, referecing the first font, and the number of loaded fonts minus one, referencing the last font. For more information see graphics_fonts.

font_size:
: It is a scalar specifying the displayed characters size. If `fractional_font` property is `"off"` only the integer part of the value is used. For more information see graphics_fonts.

fractional_font:
: This property specify whether text is displayed using fractional font sizes. Its value must be either `"on"` or `"off"`. If `"on"` the floating point value of `font_size` is used for display and the font is anti-aliased. If `"off"` only the integer part is used and the font is not smoothed.

font_angle:
: This scalar allows you to turn the label. The font is turned clockise with the angle given in degrees. Note that changing the `font_angle` will automatically disable the `auto_rotation` property.

auto_rotation:
If "on", Scilab computes automaticaly the best angle to turn the label for the display. If "off", the label can be manually turned with the font_angle property.

position:
This 2d vector allows you to place manually the label on the screen. The position is stored in the data units of the axes. Note that changing the font_angle will automatically disable the auto_position property.

auto_position:
If "on", Scilab computes automaticaly the best position in the graphic window for the display. If "off", the label can be manually places with the position property.

Examples

```
a=get("current_axes");
a.title
type(a.title)
plot3d()
a.x_label
a.y_label
a.z_label
xtitle("My title","my x axis label", "Volume","Month")

t=a.title;
t.foreground=9;
t.font_size=5;
t.font_style=5;
t.text="SCILAB";

x_label=a.x_label;
x_label.text=" Weight"
x_label.font_style= 5;
a.y_label.foreground = 12;
```

See Also

set , get , delete , xtitle , graphics_entities , axes_properties , text_properties

Authors

Djalel ABDEMOUCHE

Name

legend — draw graph legend

```
hl=legend([h,] string1,string2, ... [,pos] [,boxed])
hl=legend([h,] strings [,pos] [,boxed])
```

Parameters

h
: graphic handle on an Axes entity or vector of handles on polyline entities. The default value is the handle on `current_axes`.

string1,string2, ...
: character strings stringsi is the legend of the ith curve

strings
: n vector of strings, strings(i) is the legend of the ith curve

pos
: (optional) specify where to draw the legend; this parameter may be a integer flag (or equivalently a string flag) or a vector [x,y] which gives the coordinates of the upper left corner of the legend box. In the first case the possible values are:

  1
  : the legend is drawn in the upper right-hand corner (default).

  2
  : the legend is drawn in the upper left-hand corner.

  3
  : the legend is drawn in the lower left-hand corner.

  4
  : the legend is drawn in the lower right-hand corner.

  5
  : interactive placement with the mouse .

  -1
  : the legend is drawn at the right of the upper right-hand corner.

  -2
  : the legend is drawn at the left of the upper left-hand corner.

  -3
  : the legend is drawn at the left of the lower left-hand corner.

  -4
  : the legend is drawn at the right of the lower right-hand corner.

  -5
  : the legend is drawn above the upper left-hand corner.

  -6
  : the legend is drawn below the lower left-hand corner.

boxed
: a boolean (default value %t) which sets ot not the drawing of the box.

hl
: a handle, points to the Compound containing all the legend .

Description

Puts a legend on the current plot using the specified strings as labels. legend prepends labels by a recall of the corresponding line or patch. The recall type and properties are recovered from the given handles:

when called without handle argument (or with a handle on a axes entity) the function first build the vectors of handle on polylines entities which are the children of the given axes.

In the interactive placement mode (opt=5) move the legend box with the mouse and press the left button to release it.

Examples

```
t=linspace(0,%pi,20);
a=gca();a.data_bounds=[t(1) -1.8;t($) 1.8];

plot2d(t,[cos(t'),cos(2*t'),cos(3*t')],[-5,2 3]);
e=gce();
e1=e.children(1);e1.thickness=2;e1.polyline_style=4;e1.arrow_size_factor = 1/2;
e.children(2).line_style=4;
e3=e.children(3);e3.line_mode='on';e3.mark_background=5;

hl=legend(['cos(t)';'cos(2*t)';'cos(3*t)']);
```

See Also

plot2d, xstring, captions, polyline_properties

Name

legend_properties — description of the Legend entity properties.

Description

The Legend entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for legends drawn below `plot2dx` graphs or created by the captions function. For selected line plotted, the legend shows a sample of the line type, marker symbol, and color.

parent:
: This property contains the handle of the parent. The parent of the legend entity should be of the type `"Compound"`. This Compound entity contains also the remainder of the graph's entities.

children:
: This property contains a vector with the `children` of the handle. However, `legend` handles currently do not have any `children`.

visible:
: This field contains the `visible` property value for the entity . It should be `"on"` or `"off"` . If `"on"` the legend is drawn , If `"off"` the legend is not displayed on the screen.

text:
: This field is the character string vector which contains the legends for each annotated objects.

font_size:
: It is a scalar specifying the displayed characters size. If `fractional_font` property is `"off"` only the integer part of the value is used. For more information see graphics_fonts.

font_style:
: Specifies the font used to display the legend labels. This is a positive integer referecing one of the loaded fonts. Its value must be between 0, referecing the first font, and the number of loaded fonts minus one, referencing the last font. For more information see graphics_fonts.

font_color
: A color index, this property determines the color of the text.

fractional_font:
: This property specify whether text is displayed using fractional font sizes. Its value must be either `"on"` or `"off"`. If `"on"` the floating point value of `font_size` is used for display and the font is anti-aliased. If `"off"` only the integer part is used and the font is not smoothed.

links:
: A row array of handles. They refer to the associated polylines.

legend_location
: A character string, specifies the location of the Legend.

  - `"in_upper_right"` : captions are drawn in the upper right corner of the axes box.
  - `"in_upper_left"`: captions are drawn in the upper left corner of the axes box.
  - `"in_lower_right"`: captions are drawn in the lower right corner of the axes box.
  - `"in_lower_left"`: captions are drawn in the lower left corner of the axes box.
  - `"out_upper_right"`: captions are drawn at the right of the upper right corner of the axes box.
  - `"out_upper_left"`: captions are drawn at the left of the upper left corner of the axes box.

- `"out_lower_right"`: captions are drawn at the right of the lower right corner of the axes box.
- `"out_lower_left"`: captions are drawn at the left of the lower left corner of the axes box.
- `"upper_caption"`: captions are drawn above the upper left corner of the axes box.
- `"lower_caption"`: captions are drawn below the lower left corner of the axes box. This option correspond to the `leg` argument of plot2d
- `"by_coordinates"`: the upper left corner of the captions box is given by the "position" field of the associated data structure. The `x` and `y` positions are given as fractions of the `axes_bounds`

position
: The coordinates of the upper left corner of the legend. The `x` and `y` positions are given as fractions of the `axes_bounds` sizes. This field may be set if `legend_location=="by_coordinates"` or get for the other `legend_location` settings.

line_mode
: This field specifies if a rectangle is drawn around the legeng or not. It should be `"on"` or `"off"`. If `"on"` the rectangle is drawn using the following properties.

thickness
: This field gives the thickness of the line used to draw the rectangle shape.

foreground
: This field gives the color index of the line used to draw the rectangle shape.

fill_mode
: This field specifies if the legeng background is painted or not. It should be `"on"` or `"off"`. If `"on"` the background is painted using the color index set in the `background` field.

background
: This field gives the color index of the line used to paint the rectangle area.

clip_state:
: This field contains the default `clip_state` property value for all objects. Its value should be :

- `"off"` this means that all objects created after that are not clipped (default value).
- `"clipgrf"` this means that all objects created after that are clipped outside the Axes boundaries.
- `"on"` this means that all objects created after that are clipped outside the rectangle given by property clip_box.

clip_box:
: This field contains the default `clip_box` property value for all objects. Its value should be an empty matrix if clip_state is `"off"`. Other case the clipping is given by the vector `[x,y,w,h]` (upper-left point width height).

user_data:
: This field can be use to store any scilab variable in the text data structure, and to retreive it.

Examples

```
// x initialisation
x=[0:0.1:2*%pi]';
plot2d(x,[sin(x) sin(2*x) sin(3*x)],..
  [1,2,3],leg="L1@L2@L3")
a=get("current_axes");
l=a.children(2);
l.links
l.text=["sin(x)";"sin(2*x)";"sin(3*x)"];
l.visible="off"; // invisible
l.font_size = 2;
l.font_style = 5;
l.visible='on';
```

See Also

plot2d, graphics_entities

Authors

Djalel ABDEMOUCHE

Name

legends — draw graph legend

```
legends(strings,style,<opt_args>)
```

Parameters

strings
: n vector of strings, strings(i) is the legend of the ith curve

style
: integer row vector of size n (the plot styles, third parameter of plot2d) or an integer 2 x n matrix, style(1,k) contains the plot style for the kth curve and style(2,k) contains the line style (if style(1,k)>0) or mark color (if style(1,k)<0).

<opt_args>
: This represents a sequence of statements `key1=value1,key2=value2`,... where `key1`, `key2,...` can be one of the following:

  opt
  : specify where to draw the legends; this parameter may be a integer flag (or equivalently a string flag) or a vector [x,y] which gives the coordinates of the upper left corner of the legend box. In the first case the possible values are:

    1 or "ur"
    : the legends are drawn in the upper right-hand corner.

    2 or "ul"
    : the legends are drawn in the upper left-hand corner.

    3 or "ll"
    : the legends are drawn in the lower left-hand corner.

    4 or "lr"
    : the legends are drawn in the lower right-hand corner.

    5 or "?"
    : interactive placement with the mouse (default).

    6 or "below"
    : the legends are drawn under the graph (which is resized accordingly).

  with_box
  : a boolean (default value %t) which sets ot not the drawing of the box.

  font_size
  : an integer (default value 1) which sets the size of the font used for the names in the legend.

Description

Puts a legend on the current plot using the specified strings as labels.

In the interactive placement (opt=5 or opt="?") move the legend box with the mouse and press the left button to release it.

This function allows more flexible placement of the legends than the leg plot2d argument.

Examples

```
// Example 1
t=0:0.1:2*%pi;
plot2d(t,[cos(t'),cos(2*t'),cos(3*t')],[-1,2 3]);
legends(['cos(t)';'cos(2*t)';'cos(3*t)'],[-1,2 3],opt="lr")

scf() ;
xset("line style",2);plot2d(t,cos(t),style=5);
xset("line style",4);plot2d(t,sin(t),style=3);
legends(["sin(t)";"cos(t)"],[[5;2],[3;4]], with_box=%f, opt="?")


// Example 2
scf() ;
subplot(221)
t=0:0.1:2*%pi;
plot2d(t,[cos(t'),cos(2*t'),cos(3*t')],[-1,2 3]);
legends(['cos(t)';'cos(2*t)';'cos(3*t)'],[-1,2 3], opt=3 )

subplot(222)
xset("line style",2);plot2d(t,cos(t),style=5);
xset("line style",4);plot2d(t,sin(t),style=3);
legends(["sin(t)";"cos(t)"],[[5;2],[3;4]], with_box=%f, opt=6 )

subplot(223)
xset("line style",2);plot2d(t,cos(t),style=5);
xset("line style",4);plot2d(t,sin(t),style=3);
legends(["sin(t)";"cos(t)"],[[5;2],[3;4]], with_box=%f, opt=1, font_size=2 )

subplot(224)
t=0:0.1:2*%pi;
plot2d(t,[cos(t'),cos(2*t'),cos(3*t')],[-1,2 3]);
legends(['cos(t)';'cos(2*t)';'cos(3*t)'],[-1,2 3], opt=2, font_size=1 )
```

See Also

plot2d , xstring , xtitle , legend

Name

locate — mouse selection of a set of points

```
x=locate([n,flag])
```

Parameters

x
: matrix of size (2,n1). n1=n if the parameter n is given.

n,flag
: integer values.

Description

`locate` is used to get the coordinates of one or more points selected with the mouse in a graphics window. The coordinates are given using the current graphics scale.

If n>0, n points are selected and their coordinates are returned in the matrix `x`.

If n<=0, points are selected until the user clicks with the left button of the mouse which stands for stop. The last point (clicked with the left button) is not returned.

x=locate() is the same as `x=locate(-1)`.

If `flag=1` a cross is drawn at the points where the mouse is clicked.

See Also

xclick , xgetmouse

Authors

S.S. & J.Ph.C

Name

mesh — 3D mesh plot

```
mesh(Z)
mesh(X,Y,Z)
mesh(...,<GlobalProperty>)
mesh(...,<color>,<GlobalProperty>)
mesh(<axes_handle>,...)
```

Parameters

Z
: a real matrix defining the surface height. It can not be omitted. The Z data is a `mxn` matrix.

X,Y
: two real matrices : always set together, these data defines a new standard grid. This new `X` and `Y` components of the grid must match `Z` dimensions (see description below).

color
: an optional real matrix defining a color value for each `(X(j),Y(i))` point of the grid (see description below).

<GlobalProperty>
: This optional argument represents a sequence of couple statements `{PropertyName,PropertyValue}` that defines global objects' properties applied to all the curves created by this plot. For a complete view of the available properties (see GlobalProperty).

<axes_handle>
: This optional argument forces the plot to appear inside the selected axes given by `axes_handle` rather than the current axes (see gca).

Description

`mesh` draws a parametric surface using a rectangular grid defined by `X` and `Y` coordinates (if `{X,Y}` are not specified, this grid is determined using the dimensions of the `Z` matrix) ; at each point of this grid, a Z coordinate is given using the `Z` matrix. `mesh` is based on the `surf` command with default option `color_mode` = white index (inside the current colormap) and `color_flag` = 0.

Data entry specification :

In this paragraph and to be more clear, we won't mention `GlobalProperty` optional arguments as they do not interfer with entry data (except for `"Xdata"`, `"Ydata"` and `"Zdata"` property, see GlobalProperty). It is assumed that all those optional arguments could be present too.

If `Z` is the only matrix specified, (Z) plots the matrix `Z` versus the grid defined by `1:size(Z,2)` along the x axis and `1:size(Z,1)` along the y axis.

Remarks

To enable the tranparency mode you should set the `color_mode` option to 0.

Examples

```
[X,Y]=meshgrid(-1:.1:1,-1:.1:1);
Z=X.^2-Y.^2;
```

```
xtitle('z=x2-y ^2');
mesh(X,Y,Z);
```

See Also

surf , meshgrid , plot2d , LineSpec , GlobalProperty

Authors

F.Belahcene

Name

milk_drop — milk drop 3D function

```
z=milk_drop(x,y)
```

Parameters

x,y
: two row vectors of size n1 and n2.

z
: matrix of size (n1,n2).

Description

`milk_drop` is a function representing the surface of a milk drop falling down into milk. It can be used to test functions `eval3d` and `plot3d`.

Examples

```
x=-2:0.1:2; y=x;
z=eval3d(milk_drop,x,y);
plot3d(x,y,z)
```

See Also

eval3d , plot3d

Authors

Steer S.

Name

move — move, translate, a graphic entity and its children.

```
move(h,xy)
move(h,xy,"alone")
```

Parameters

h
: a handle, the handle of the entity to move.

xy
: an array `[dx,dy]` which gives the translation vector to apply.

"alone"
: string keyword (optional).

Description

This routine can be used to apply a translation to a graphics entity. If the entity has children, they will be also translated.

Given the keyword `"alone"` , only the specified entity needs to be redraw. It must specially be used with the `pixel_drawing_mode` property of the figure entity (see draw objects under `"xor"` drawing mode).

Examples

```
```

See Also

get , set , draw , figure_properties , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

name2rgb — returns the RGB values of a named color

```
rgb=name2rgb(name)
```

Parameters

name
: name of the color.

rgb
: vector of RGB integer values of a color.

Description

`name2rgb` returns the RGB values of a color given by its name. The result is a vector `[r,g,b]` where `r`, `g` and `b` are integers between 0 and 255 corresponding to colors components red, green and blue. As usual 0 means no intensity and 255 means all the intensity of the color.

If no color is found `[]` is returned.

The list of all known colors is given by color_list.

Examples

```
rgb=name2rgb("gold")
rgb2name(rgb)
```

See Also

color , color_list , rgb2name

Name

newaxes — Creates a new Axes entity

```
a=newaxes()
```

Parameters

a
: a handle, the handle on the newly created Axes entity

Description

`newaxes()` is used to create a new `Axes` entity (see graphics_entities) in the current figure. The properties of this entity are inherited from the `default_axes` entity (see gda)

Examples

```
clf()
a1=newaxes();
a1.axes_bounds=[0,0,1.0,0.5];
t=0:0.1:20;
plot(t,acosh(t),'r')
a2=newaxes();
a2.axes_bounds=[0,0.5,1.0,0.5];
x=0:0.1:4;
plot(x,sinh(x))
legend('sinh')

sca(a1); //make first axes current
plot(t,asinh(t),'g')
legend(['acosh','asinh'])
```

See Also

subplot , gda , sca

Authors

S. Steer, INRIA

Name

nf3d — rectangular facets to plot3d parameters

```
[xx,yy,zz]=nf3d(x,y,z)
```

Parameters

x,y,x,xx,yy,zz
: 6 real matrices

Description

Utility function. Used for transforming rectangular facets coded in three matrices x,y,z to scilab code for facets accepted by plot3d.

Examples

```
//A sphere...
u = linspace(-%pi/2,%pi/2,40);
v = linspace(0,2*%pi,20);
x= cos(u)'*cos(v);
y= cos(u)'*sin(v);
z= sin(u)'*ones(v);
//plot3d2(x,y,z) is equivalent to...
[xx,yy,zz]=nf3d(x,y,z); plot3d(xx,yy,zz)
```

See Also

plot3d , plot3d2

Name

object_editor — description of the graphic object editor capacities
graphic — description of the graphic object editor capacities
menus — description of the graphic object editor capacities

Description

Scilab graphics allow the user to have interaction with graphics before and after having them drawn. Each graphics window and the drawing it contains are represented by hierchical entities. The hierachy top level is the Figure. Each Figure defines at least one child of type Axes. Each Axes entity contains a set of leaf entities which are the basic graphics objects like Polylines, Rectangles, Arcs, Segs,... It can also contain a Compound type which are recursive sets of entities.

The main interest of the new graphic mode is to make property changes easier. This new graphics'mode provides a set of high-level graphing routines (see set, get) used to control objects' properties such as data, coordinates and scaling, color and appearences without requiring to replay the initial graphics commands.

Graphics entities are associated to Scilab variables of type `handle`. The handle is a unique identifier which is associated to each instance of a created graphical entity. Using this handle, it will be possible to reach entities' properties through "set" and "get" routines. The handles are also used to manipulate graphics objects, to move them, to make copies or delete them.

To complete and use the graphics handle capacity at its maximum, a graphic object editor has been created too. It is a set of Tcl/Tk interfaces available for each kind of graphics objects (see graphics_entities for more details) that can be enabled for each graphic window. To make it work, select the `Edit` menu in the graphic window. Seven graphics editing operations are available :

Select figure as current:
Let this

figure be the current one.

Redraw figure:
Redraw the content of the graphics window.

Erase figure:
Erase the content of the graphics window. Its action corresponds to `clf` routine.

The last eight items are specially dedicated to the graphic editor :

Copy object:
Using the mouse, it allows the user to select a 2D object (like a curve, a rectangle...) and put it in the clipboard. Thus, by a next call to `Paste object`, the object is copied in the selected current axes.

Paste object:
Allow the user to paste a previous object put into in the clipboard inside the selected current axes.

Move object:
Using the mouse, it allows the user to move a 2D object (like a curve, a rectangle...) inside the selected current axes.

Delete object:
Using the mouse, it allows the user to pick up a 2D object (like a curve, a rectangle...) inside the selected current axes and to delete it instantly.

Figure Properties:
Launch the Tcl/Tk interface for the Figure object applied to the figure handle of the graphics window.

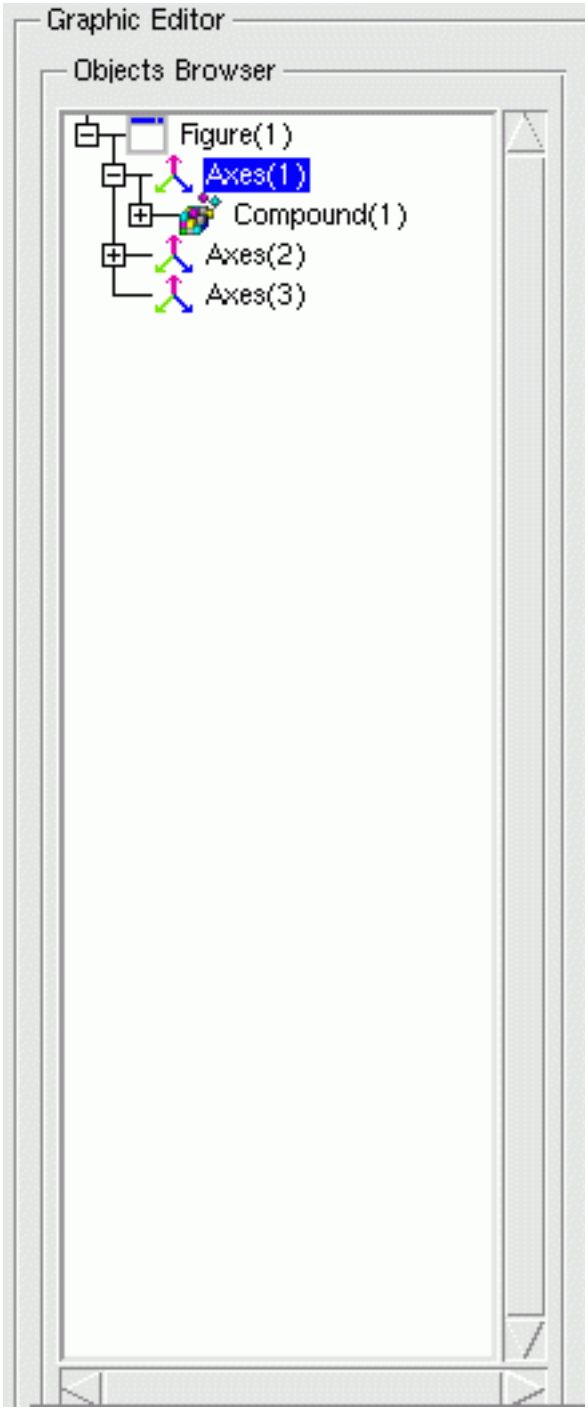
Current Axes Properties:
Launch the Tcl/Tk interface for the Axes object applied to the current axes handle of the graphics window.

Start Entity Picker:
Start an event handler on the graphics window to catch the mouse clicks on graphics objects and launch the corresponding Tcl/Tk interface. The left mouse-click allows object edition and the right click performs a move of the selected object. Note that, for now, this feature is applied to 2D objects only.

Stop Entity Picker:
Stop the action of the Entity Picker by terminating the event handler on the graphics window.

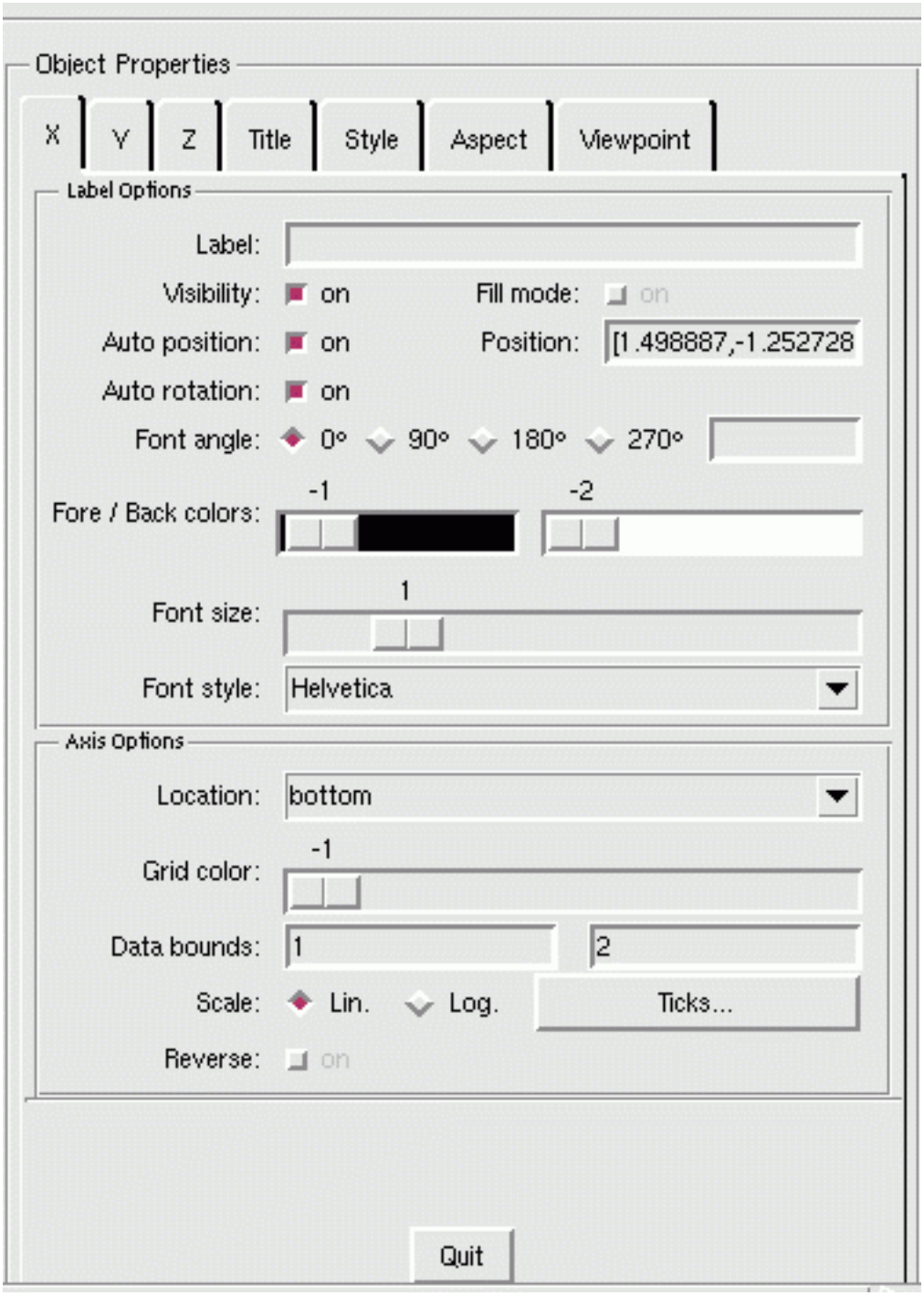
Once the graphic interface is enabled (using the `Figure Properties` or `Current Axes Properties` options), two main areas appear :

A tree selector:
Placed on the left side of the graphical editor, a hierarchical tree selector specifies which object is currently edited. It can be used to switch from a graphic object to another provided that they are in the same graphic window.



A notebook:

The second area represents a notebook composed with different properties pages (like `Style`, `Data`, `Clipping`...) depending on the selected graphic object. Using this editor, man can edit more easily the whole properties set of each graphic object (like through the `"set"` and `"get"` commands). Here is an example of the axes' notebook displaying axes properties:



Furthermore, you can legend/annotate your figure using sample primitives given inside the `Insert` menu in the graphic window. Using the mouse and following the instruction in the message subwindow, you can add a:

Line:

Draw a line between 2 left mouse clicks. The line lives in the axes where the first point was selected.

Polyline:

Draw a polyline by clicking on the left button to define the line path and right click at last to complete the drawing. The polyline lives in the axes where the first point was selected.

Arrow:
: Draw an arrow between 2 left mouse clicks. The arrow lives in the axes where the first point was selected.

Double arrow:
: Draw a double-sided arrow between 2 left mouse clicks. The double arrow lives in the axes where the first point was selected.

Text:
: Open a dialog box to enter the text, then click on the figure window to place it. The text lives in the axes where the point was selected.

Rectangle:
: Draw a rectangle : 2 left mouse clicks define respectively the upper left corner and the lower-right corner of the rectangle. The rectangle lives in the axes where the first point was selected.

Circle:
: Draw a circle : 2 left mouse clicks define respectively the upper left corner and the lower-right corner of the bounding-box where the circle lives. The rectangle lives in the axes where the first point was selected.

See Also

graphics_entities , set , get , clf , plot

Authors

F.Leray INRIA

Name

oceancolormap — linear blue colormap

```
cmap=oceancolormap(n)
```

Parameters

n
: integer >= 3, the colormap size.

cmap
: matrix with 3 columns `[R,G,B]`.

Description

`oceancolormap` computes a colormap with with `n` blue colors varying linearly from black to white.

Examples

```
f = scf();
plot3d1();
f.color_map = oceancolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

oldplot — simple plot (old version)

```
oldplot(x,y,[xcap,ycap,caption])
oldplot(y)
```

Parameters

x,y
: two vectors with same sizes

xcap,ycap,caption
: character strings or string matrices

Description

Plot `y` as function of `x`. `xcap` and `ycap` are captions for x-axis and y-axis respectively and `caption` is the caption of the plot.

Invoked with only one argument, `oldplot(y)` plots the `y` vector or, if `y` is a matrix, it plots all its row vectors on the same plot. This plot is done with respect to the vector 1:<number of columns of `y`>.

`oldplot` is obsolete. Use `plot2d` or `plot` instead.

Examples

```
x=0:0.1:2*%pi;
// simple plot
oldplot(sin(x))
// using captions
xbasc()
oldplot(x,sin(x),"sin","time","plot of sinus")
// plot 2 functions
xbasc()
oldplot([sin(x);cos(x)])
```

See Also

plot2d , plot

Authors

J.Ph.C.

Name

param3d — 3D plot of a parametric curve

```
param3d(x,y,z,[theta,alpha,leg,flag,ebox])
```

Parameters

x,y,z
: three vectors of the same size (points of the parametric curve).

theta, alpha
: real values giving in degree the spherical coordinates of the observation point. *The default values are 35 and 45 degree.*

leg
: string defining the labels for each axis with @ as a field separator, for example "X@Y@Z".

flag=[type,box]
: `type` and `box` have the same meaning as in `plot3d`:

  type
  : an integer (scaling).

    type=0
    : the plot is made using the current 3D scaling (set by a previous call to `param3d`, `plot3d`, `contour` or `plot3d1`).

    type=1
    : rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument `ebox`.

    type=2
    : rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data. *This is the default value.*

    type=3
    : 3d isometric with box bounds given by optional `ebox`, similarily to `type=1`.

    type=4
    : 3d isometric bounds derived from the data, similarily `to type=2`.

    type=5
    : 3d expanded isometric bounds with box bounds given by optional `ebox`, similarily to `type=1`.

    type=6
    : 3d expanded isometric bounds derived from the data, similarily to `type=2`.Note that axes boundaries can be customized through the axes entity properties (see axes_properties).

  box
  : an integer (frame around the plot).

    box=0
    : nothing is drawn around the plot.

    box=1
    : unimplemented (like box=0).

box=2
only the axes behind the surface are drawn.

box=3
a box surrounding the surface is drawn and captions are added.

box=4
a box surrounding the surface is drawn, captions and axes are added.Note that axes aspect can also be customized through the axes entity properties (see axes_properties). *This is the default value.*

ebox
It specifies the boundaries of the plot as the vector `[xmin,xmax,ymin,ymax,zmin,zmax]`. This argument is used together with `type` in `flag` : if it is set to `1`, `3` or `5` (see above to see the corresponding behaviour). If `flag` is missing, `ebox` is not taken into acoount. Note that, when specified, the `ebox` argument acts on the `data_bounds` field that can also be reset through the axes entity properties (see axes_properties). The ebox default value is [0,1,0,1,0,1].

Description

`param3d` is used to plot a 3D curve defined by its coordinates `x`, `y` and `z`. Note that data can also be got or modified through the surface entity properties (see surface_properties).

Note that properties like `rotation angles`, `colors` and `thickness` of the plotted curves can also be got or modified through the param3d entity properties (see param3d_properties).

Use `param3d1` to do multiple plots.

Enter the command `param3d()` to see a demo.

Examples

```
t=0:0.1:5*%pi;
param3d(sin(t),cos(t),t/10,35,45,"X@Y@Z",[2,3])

e=gce() //the handle on the 3D polyline

e.foreground=color('red');

a=gca(); //the handle on the axes
a.rotation_angles=[10 70];
```

See Also

param3d1, plot3d

Authors

J.Ph.C.

Name

param3d1 — 3D plot of parametric curves

```
param3d1(x,y,z,[theta,alpha,leg,flag,ebox])
param3d1(x,y,list(z,colors),[theta,alpha,leg,flag,ebox])
```

Parameters

x,y,z
: matrices of the same size (nl,nc).

  Each column i of the matrices corresponds to the coordinates of the ith curve. You can give a specific color for each curve by using `list(z,colors)` instead of `z`, where `colors` is a vector of size `nc`. If `color(i)` is negative the curve is plotted using the mark with id `abs(style(i))`; if `style(i)` is strictly positive, a plain line with color id `style(i)` or a dashed line with dash id `style(i)` is used.

theta,alpha
: real values giving in degree the spherical coordinates of the observation point. *The default values are 35 and 45 degree.*

leg
: string defining the captions for each axis with @ as a field separator, for example "X@Y@Z".

flag=[type,box]
: `type` and `box` have the same meaning as in `plot3d`:

  type
  : an integer (scaling).

    type=0
    : the plot is made using the current 3D scaling (set by a previous call to `param3d`, `plot3d`, `contour` or `plot3d1`).

    type=1
    : rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument `ebox`.

    type=2
    : rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data. *This is the default value.*

    type=3
    : 3d isometric with box bounds given by optional `ebox`, similarily to `type=1`.

    type=4
    : 3d isometric bounds derived from the data, similarily `to type=2`.

    type=5
    : 3d expanded isometric bounds with box bounds given by optional `ebox`, similarily to `type=1`.

    type=6
    : 3d expanded isometric bounds derived from the data, similarily to `type=2`.Note that axes boundaries can be customized through the axes entity properties (see axes_properties).

  box
  : an integer (frame around the plot).

box=0
: nothing is drawn around the plot.

box=1
: unimplemented (like box=0).

box=2
: only the axes behind the surface are drawn.

box=3
: a box surrounding the surface is drawn and captions are added.

box=4
: a box surrounding the surface is drawn, captions and axes are added.Note that axes aspect can also be customized through the axes entity properties (see axes_properties). *This is the default value.*

ebox
: It specifies the boundaries of the plot as the vector `[xmin,xmax,ymin,ymax,zmin,zmax]`. This argument is used together with `type` in `flag` : if it is set to `1`, `3` or `5` (see above to see the corresponding behaviour). If `flag` is missing, `ebox` is not taken into acount. Note that, when specified, the `ebox` argument acts on the `data_bounds` field that can also be reset through the axes entity properties (see axes_properties). The `ebox` default value is `[0,1,0,1,0,1]`.

Description

`param3d1` is used to plot 3D curves defined by their coordinates `x`, `y` and `z`. Note that data can also be got or modified through the surface entity properties (see surface_properties).

Note that properties like `rotation angles`, `colors` and `thickness` of the plotted curves can also be got or modified through the param3d entity properties (see param3d_properties).

Enter the command `param3d1()` to see a demo.

Examples

```
xset('window',20) // create a window number 20
t=[0:0.1:5*%pi]';
param3d1([sin(t),sin(2*t)],[cos(t),cos(2*t)],..
  list([t/10,sin(t)],[3,2]),35,45,"X@Y@Z",[2,3])

xdel(20) ;
 a=get("current_axes");//get the handle of the newly created axes
 t=[0:0.1:5*%pi]';
 param3d1([sin(t),sin(2*t)],[cos(t),cos(2*t)],[t/10,sin(t)])
 a.rotation_angles=[65,75];
 a.data_bounds=[-1,-1,-1;1,1,2]; //boundaries given by data_bounds
 a.thickness = 2;
 h=a.children //get the handle of the param3d entity: an Compound composed of 2
 h.children(1).foreground = 3 // first curve
 curve2 = h.children(2);
 curve2.foreground = 6;
 curve2.mark_style = 2;
```

See Also

param3d, plot3d, plot2d, gca, xdel, delete

Authors

J.Ph.C.

Name

paramfplot2d — animated 2D plot, curve defined by a function

```
paramfplot2d(f,x,theta)
paramfplot2d(f,x,theta,flag)
paramfplot2d(f,x,theta,flagrect)
```

Parameters

x
: real vector.

f
: function `y=f(x,t)` . f is a Scilab function or a dynamically linked routine (referred to as a string).

theta
: real vector (set of parameters).

flag
: string `'no'` or `'yes'`: If "yes" screen is cleared between two consecutive plots.

rect
: "rectangle" [xmin, xmax, ymin, ymax] (1 x 4 real vector),

Description

Animated plot of the function x-->f(x,t) for t=theta(1),theta(2),etc. f can be a either Scilab function or a dynamically linked routine since y=f(x,t) is evaluated as y=feval(x(:),t,f). See feval. f: mapping x,t -> f(x,t) = `R^N` valued function for x= vector of `R^N` and t=real number. x is a N-vector of x-values and for each t in theta, f(x,t)=N-vector of y-values.

Examples

```
deff('y=f(x,t)','y=t*sin(x)')
x=linspace(0,2*%pi,50);theta=0:0.05:1;
paramfplot2d(f,x,theta);
```

See Also

plot2d , feval , fplot2d

Name

pie — draw a pie

```
pie(x)
pie(x[,sp[,txt]])
```

Parameters

x
: a scalar or a vector of positive reals.

sp
: a real scalar or a vector of reals.

txt
: a cell or a vector of strings.

Description

`pie(x)`: if size of x is N then pie function draws a pie with N parts, the area of the ith part is equal to (x(i)/sum(x))*(surface of the unit cercle).

`pie(x,sp)`: the sp vector allows to cut one or several parts of the pie, (the size of sp must be equal to N). if the value of the ith index of sp is different of zero then the ith part is separated from the others by a space, else if it' s equal to zero then it is attached to the others.

`pie(x,txt)`: the txt vector allows to write a text for each part of the pie, the ith component of txt corresponds to the ith part (default : it's written the percentages which corresponds to the parts suface). The size of txt must be equal to N.

Examples

```
// First example : one input argument  x=[1 2 5]
scf(0);
pie([1 2 5]);

// Second example : two input arguments x=[5 9  4 6 3], sp=[0 1 0 1 0], the sec
scf(1);
pie([5 9 4 6 3],[0 1 0 1 0]);

// Third example : three input arguments, x=[3 4 6 2], sp=[0 1 0 0], txt=["part
scf(2);
pie([3 4 6 2],[0 1 0 0],["part1","part2","part3","part4"]);
```

See Also

xfpolys

Authors

Farid Belahcene

Name

pinkcolormap — sepia tone colorization on black and white images

```
cmap=pinkcolormap(n)
```

Parameters

n
: integer >= 3, the colormap size.

cmap
: matrix with 3 columns `[R,G,B]`.

Description

`pinkcolormap` computes a colormap that provides sepia tone colorization on black and white images

Examples

```
f = scf();
plot3d1();
f.color_map = pinkcolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

plot — 2D plot

```
plot(y,<LineSpec>,<GlobalProperty>)
plot(x,y,<LineSpec>,<GlobalProperty>)
plot(x1,y1,<LineSpec1>,x2,y2,<LineSpec2>,...xN,yN,<LineSpecN>,<GlobalProperty1>
plot(<axes_handle>,...)
```

Parameters

x
: a real matrice or vector. If omitted, it is assumed to be the vector `1:n` where `n` is the number of curve points given by the `y` parameter.

y
: a real matrice or vector. `y` can also be a function defined as a macro or a primitive.

<LineSpec>
: This optional argument must be a string that will be used as a shortcut to specify a way of drawing a line. We can have one `LineSpec` per `y` or `{x,y}` previously entered. `LineSpec` options deals with LineStyle, Marker and Color specifiers (see LineSpec). Those specifiers determine the line style, mark style and color of the plotted lines.

<GlobalProperty>
: This optional argument represents a sequence of couple statements `{PropertyName,PropertyValue}` that defines global objects' properties applied to all the curves created by this plot. For a complete view of the available properties (see GlobalProperty).

<axes_handle>
: This optional argument forces the plot to appear inside the selected axes given by `axes_handle` rather than the current axes (see gca).

Description

`plot` plots a set of 2D curves. `plot` has been rebuild to better handle Matlab syntax. To improve graphical compatibility, Matlab users should use `plot` (rather than plot2d).

Data entry specification :

In this paragraph and to be more clear, we won't mention `LineSpec` nor `GlobalProperty` optional arguments as they do not interfer with entry data (except for `"Xdata"`, `"Ydata"` and `"Zdata"` property, see GlobalProperty). It is assumed that all those optional arguments could be present too.

If `y` is a vector, plot(y) plots vector `y` versus vector `1:size(y,'*')`.

If `y` is a matrix, plot(y) plots each columns of `y` versus vector `1:size(y,1)`.

If `x` and `y` are vectors, plot(x,y) plots vector `y` versus vector `x`. `x` and `y` vectors should have the same number of entries.

If `x` is a vector and `y` a matrix plot(x,y) plots each columns of `y` versus vector `x`. In this case the number of columns of `y` should be equal to the number of `x` entries.

If `x` and `y` are matrices, plot(x,y) plots each columns of `y` versus corresponding column of `x`. In this case the `x` and `y` sizes should be the same.

Finally, if only `x` or `y` is a matrix, the vector is plotted versus the rows or columns of the matrix. The choice is made depending on whether the vector's row or column dimension matches the matrix row

or column dimension. In case of a square matrix (on `x` or `y` only), priority is given to columns rather than lines (see examples below).

`y` can also be a function defined as a macro or a primitive. In this case, `x` data must be given (as a vector or matrix) and the corresponding computation `y(x)` is done implicitly.

The `LineSpec` and `GlobalProperty` arguments sould be used to customize the plot. Here is a complete list of the available options.

LineSpec
: This option may be used to specify, in a short and easy manner, how the curves are drawn. It must always be a string containing references to LineStyle, Marker and Color specifiers.

  These references must be set inside the string (order is not important) in an unambiguous way. For example, to specify a red long-dashed line with the diamond mark enabled, you can write : `'r--d'` or `'--dire'` or `'--reddiam'` or another unambiguous statement... or to be totally complete `'diamondred--'` (see LineSpec).

  Note that the line style and color, marks color (and sizes) can also be (re-)set throught the polyline entity properties (see polyline_properties).

GlobalProperty
: This option may be used to specify how all the curves are plotted using more option than via `LineSpec`. It must always be a couple statement constituted of a string defining the `PropertyName`, and its associated value `PropertyValue` (which can be a string or an integer or... as well depending on the type of the `PropertyName`). Using `GlobalProperty`, you can set multiple properties : every properties available via LineSpec and more : the marker color (foreground and background), the visibility, clipping and thickness of the curves. (see GlobalProperty)

  Note that all these properties can be (re-)set throught the polyline entity properties (see polyline_properties).

Remarks

By default, successive plots are superposed. To clear the previous plot, use `clf()`. To enable `auto_clear` mode as the default mode, edit your default axes doing:

da=gda();

da.auto_clear = 'on'

For a better display `plot` function may modify the `box` property of its parent Axes. This happens when the parent Axes were created by the call to `plot` or were empty before the call. If one of the axis is centered at origin, the box is disable. Otherwise, the box is enable.

For more information about box property and axis positionning see axes_properties

A default color table is used to color plotted curves if you do not specify a color. When drawing multiple lines, the plot command automatically cycles through this table. Here are the used colors:

R	G	B
0.	0.	1.
0.	0.5	0.
1.	0.	0.
0.	0.75	0.75
0.75	0.	0.75
0.75	0.75	0.
0.25	0.25	0.25

Enter the command `plot` to see a demo.

Examples

```
// x initialisation
x=[0:0.1:2*%pi]';
//simple plot
plot(sin(x))
clf()
plot(x,sin(x))
//multiple plot
clf()
plot(x,[sin(x) sin(2*x) sin(3*x)])
clf()

// axis on the right
plot(x,sin(x))
a=gca(); // Handle on current axes entity
a.y_location ="right";
clf()

// axis centered at (0,0)
plot(x-4,sin(x),x+2,cos(x))
a=gca(); // Handle on axes entity
a.x_location = "middle";
a.y_location = "middle";

// Some operations on entities created by plot ...
a=gca();
a.isoview='on';
a.children // list the children of the axes : here it is an Compound child compo
poly1= a.children.children(2); //store polyline handle into poly1
poly1.foreground = 4; // another way to change the style...
poly1.thickness = 3;  // ...and the tickness of a curve.
poly1.clip_state='off' // clipping control
a.isoview='off';

//LineSpec and GlobalProperty examples:
clf();
t=0:%pi/20:2*%pi;
plot(t,sin(t),'ro-.',t,cos(t),'cya+',t,abs(sin(t)),'--mo')
scf(2)
plot([t ;t],[sin(t) ;cos(t)],'xdat',[1:2])
scf(3)
axfig3 = gca();
scf(4) // should remain blank
plot(axfig3,[t ;t],[sin(t) ;cos(t)],'zdat',[1:2],'marker','d','markerfac','gree
xdel(winsid())

//Data specification
t=-%pi:0.1:%pi;
size(t)
plot(t) // simply plots y versus t vector size
clf(); // clear figure

plot(t,sin(t)); // plots sin(t) versus t
clf();
```

```
t=[1     1     1     1
   2     3     4     5
   3     4     5     6
   4     5     6     7];

plot(t) // plots each t column versus row size
clf();

subplot(221)
plot(t,sin(t)); // plots sin(t) versus t column by column this time
xtitle("sin(t) versus t")
subplot(222)
plot(t,sin(t)')
xtitle("sin(t)'' versus t")
subplot(223)
plot(t',sin(t))
a=gca();
a.data_bounds=[0 -1;7 1]; // to see the vertical line hiddden by the y axis
xtitle("sin(t) versus t''")
subplot(224)
plot(t',sin(t)')
xtitle("sin(t)'' versus t''")

clf();

//Special case 1
//x : vector ([5 6 7 8]) and y : matrix (t)
x=[5 6 7 8]
plot(x,t);
plot(x',t); // idem, x is automatically transposed to match t (here the columns
clf()

// Only one matching possibility case : how to make 4 identical plots in 4 manne
// x is 1x4 (vector) and y is 4x5 (non square matrix)
subplot(221);
plot(x,[t [8;9;10;12]]');
subplot(222);
plot(x',[t [8;9;10;12]]');
subplot(223);
plot(x,[t [8;9;10;12]]');
subplot(224);
plot(x',[t [8;9;10;12]]');
clf()


//Special case 2
// Case where only x or y is a square matrix
//x : matrix (t) and y  : vector ([1 2 3 4])
plot(t,[1 2 3 4]) // equivalent to plot(t,[1 1 1 1;2 2 2 2;3 3 3 3;4 4 4 4])
plot(t,[1;2;3;4]) // the same plot
clf();

// t is transposed : notice the priority given to the columns treatment
plot(t',[1 2 3 4]) // equivalent to plot(t',[1 1 1 1;2 2 2 2;3 3 3 3;4 4 4 4])
plot(t',[1 2 3 4]') // the same plot
clf();
```

```
// y is a function defined by..
// ..a primitive
plot(1:0.1:10,sin) // equivalent to plot(1:0.1:10,sin(1:0.1:10))
clf();

// ..a macro:
deff('[y]=toto(x)','y=x.*x')
plot(1:10,toto)
```

See Also

plot2d , surf , scf , clf , xdel , delete , LineSpec , GlobalProperty

Authors

F.Leray

Name

plot2d — 2D plot

```
plot2d([x],y)
plot2d([x],y,<opt_args>)
```

Parameters

x
: a real matrice or vector. If omitted, it is assumed to be the vector `1:n` where `n` is the number of curve points given by the `y` parameter.

y
: a real matrice or vector.

<opt_args>
: This represents a sequence of statements `key1=value1,key2=value2,...` where `key1`, `key2,...` can be one of the following:

  style
  : sets the style for each curve. The associated value should be a real vector with integer (positive or negative) values

  rect
  : sets the mimimal bounds requested for the plot. The associated value should be a real vector with four entries: `[xmin,ymin,xmax,ymax]`.

  logflag
  : sets the scale (linear or logarithmic) along the axes. The associated value should be a a string with possible values: `"nn"`, `"nl"` , `"ln"` and `"ll"` .

  frameflag
  : controls the computation of the actual coordinate ranges from the minimal requested values. The associated value should be an integer ranging from 0 to 8.

  axesflag
  : specifies how the axes are drawn. The associated value should be an integer ranging from 0 to 5.

  nax
  : sets the axes labels and tics definition. The associated value should be a real vector with four integer entries `[nx,Nx,ny,Ny]`

  leg
  : sets the curves captions. The associated value should be a character string

Description

`plot2d` plots a set of 2D curves. If you are familiar with Matlab `plot` syntax, you should use plot.

If `x` and `y` are vectors, plot2d(x,y,<opt_args>) plots vector `y` versus vector `x`. `x` and `y` vectors should have the same number of entries.

If `x` is a vector and `y` a matrix plot2d(x,y,<opt_args>) plots each columns of `y` versus vector `x`. In this case the number of columns of `y` should be equal to the number of `x` entries.

If `x` and `y` are matrices, plot2d(x,y,<opt_args>) plots each columns of `y` versus corresponding column of `x`. In this case the `x` and `y` sizes should be the same.

If `y` is a vector, plot2d(y,<opt_args>) plots vector `y` versus vector `1:size(y,'*')`.

If `y` is a matrix, plot2d(y,<opt_args>) plots each columns of `y` versus vector `1:size(y,1)`.

The `<opt_args>` arguments sould be used to customize the plot

style
: This option may be used to specify how the curves are drawn. If this option is specified, the associated value should be a vector with as many entries as curves.

  - if `style(i)` is strictly positive, the curve is drawn as plain line and `style(i)` defines the index of the color used to draw the curve (see getcolor). Note that the line style and the thickness can be set throught the polyline entity properties (see polyline_properties).

    Piecewise linear interpolation is done between the given curve points.

  - if `style(i)` is negative or zero, the given curve points are drawn using marks, `abs(style(i))` defines the mark with id used. Note that the marks color and marks sizes can be set throught the polyline entity properties (see polyline_properties).

logflag
: This option may be used to set the scale (linear or logarithmic) along the axes. The associated value should be a a string with possible values: `"nn"`, `"nl"` , `"ln"` and `"ll"`. `"l"` stands for logarithmic scale and graduations and `"n"` for normal scale.

rect
: This option may be used to set the mimimal bounds requested for the plot. If this option is specified, the associated value should be a real vector with four entries:

  `[xmin,ymin,xmax,ymax]`. `xmin` and `xmax` defines the bounds on the abscissae while `ymin` and `ymax` defines the bounds on the ordinates.

  This argument may be used together with the `frameflag` option to specify how the axes boundaries are derived from teh given `rect` argument. If the `frameflag` option is not given, it is supposed to be `frameflag=7`.

  The axes boundaries can also be customized throught the axes entity properties (see axes_properties).

frameflag
: This option may be used to control the computation of the actual coordinate ranges from the minimal requested values. Actual ranges can be larger than minimal requirements.

  frameflag=0
  : no computation, the plot use the previous (or default) scale.

  frameflag=1
  : The actual range is the range given by the `rect` option.

  frameflag=2
  : The actual range is computed from the min/max of the `x` and `y` data.

  frameflag=3
  : The actual range is the range given by the `rect` option and enlarged to get an isometric scale.

  frameflag=4
  : The actual range is computed from the min/max of the `x` and `y` data and enlarged to get an isometric scale.

  frameflag=5
  : The actual range is the range given by the `rect` option and enlarged to get pretty axes labels.

frameflag=6
: The actual range is computed from the min/max of the `x` and `y` data and enlarged to get pretty axes labels.

frameflag=7
: like `frameflag=1` but the previous plot(s) are redrawn to use the new scale. Used to add the current graph to a previous one.

frameflag=8
: like`frameflag=2` but the previous plot(s) are redrawn to use the new scale. Used to add the current graph to a previous one.

frameflag=9
: like`frameflag=8` but the range is enlarged to get pretty axes labels. This the default value.

The axes boundaries can also be customized throught the axes entity properties (see axes_properties)

axesflag
: This option may be used to specify how the axes are drawn. The associated value should be an integer ranging from 0 to 5 :

axesflag=0
: nothing is drawn around the plot.(axes_visible=["off" "off"];box="off")

axesflag=1
: axes are drawn, the y=axis is displayed on the left (axes_visible=["on" "on"];box="on",y_location="left").

axesflag=2
: the plot is surrounded by a box without tics. (axes_visible=["off" "off"];box="on").

axesflag=3
: axes are drawn, the y=axis is displayed on the right.(axes_visible=["on" "on"];box="off",y_location="right").

axesflag=4
: axes are drawn centred in the middle of the frame box

axesflag=5
: axes are drawn so as to cross at point (0,0). If point (0,0) does not lie inside the frame, axes will not appear on the graph. (axes_visible=["on" "on"];box="on",y_location="middle").

axesflag=9
: axes are drawn, the y=axis is displayed on the left (axes_visible=["on" "on"];box="off",y_location="left"). This is the defualt value

The axes aspect can also be customized throught the axes entity properties (see axes_properties).

nax
: This option may be used to specify the axes labels and tics definition when `axesflag=1` option is used. The associated value should be a real vector with four integer entries `[nx,Nx,ny,Ny]`.

`Nx` gives the number of main tics to be used on the x-axis (**no more taken into account**), `nx` gives the number of subtics to be drawn between two main x-axis tics.

`Ny` and `ny` give similar information for the y-axis.

If `axesflag` option is not set `nax` option supposes that `axesflag` option has been set to 1.

leg

This option may be used to sets the curve captions. It must be a string with the form `"leg1@leg2@...."` where `leg1` , `leg2` , etc. are respectively the captions of the first curve, of the second curve, etc. The default is `" "`.

The curve captions are drawn on below the x-axis. This option is not flexible enough, use the captions or legend functions preferably.

More information

To get more information on the plot2d old syntax , use the plot2d_old_version help document.

By default, successive plots are superposed. To clear the previous plot, use `clf()`.

Enter the command `plot2d()` to see a demo.

Other high level plot2d functions exist:

- plot2d2 same as `plot2d` but the curve is supposed to be piecewise constant.
- plot2d3 same as `plot2d` but the curve is plotted with vertical bars.
- plot2d4 same as `plot2d` but the curve is plotted with vertical arrows.

Examples

```
// x initialisation
x=[0:0.1:2*%pi]';
//simple plot
plot2d(sin(x));
clf();
plot2d(x,sin(x));
//multiple plot
clf();
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
// multiple plot giving the dimensions of the frame
clf();
plot2d(x,[sin(x) sin(2*x) sin(3*x)],rect=[0,0,6,0.5]);
//multiple plot with captions and given tics + style
clf();
plot2d(x,[sin(x) sin(2*x) sin(3*x)],..
  [1,2,3],leg="L1@L2@L3",nax=[2,10,2,10],rect=[0,-2,2*%pi,2]);
// isoview
clf();
plot2d(x,sin(x),1,frameflag= 4);
// scale
clf();
plot2d(x,sin(x),1,frameflag= 6);
// auto scaling with previous plots + style
clf();
plot2d(x,sin(x),-1);
plot2d(x,2*sin(x),12);
plot2d(2*x,cos(x),3);
// axis on the right
clf();
plot2d(x,sin(x),leg="sin(x)");
a=gca(); // Handle on axes entity
a.y_location ="right";
```

```
// axis centered at (0,0)
clf();
plot2d(x-4,sin(x),1,leg="sin(x)");
a=gca(); // Handle on axes entity
a.x_location = "middle";
a.y_location = "middle";
// Some operations on entities created by plot2d ...
a=gca();
a.isoview='on';
a.children // list the children of the axes.
// There are a compound made of two polylines and a legend
poly1= a.children(1).children(1); //store polyline handle into poly1
poly1.foreground = 4; // another way to change the style...
poly1.thickness = 3;  // ...and the tickness of a curve.
poly1.clip_state='off'; // clipping control
leg = a.children(2); // store legend handle into leg
leg.font_style = 9;
leg.line_mode = "on";
a.isoview='off';
```

See Also

plot, plot2d1, plot2d2, plot2d3, plot2d4, clf, xdel, delete

Authors

Name

plot2d1 — 2D plot (logarithmic axes) (obsolete)

```
plot2d1(str,x,y,[style,strf,leg,rect,nax])
```

Parameters

str
: is a string of length three `"abc"`.

  a
  : can have the following values: `e`, `o` or `g`.

    e
    : means "empty". Itspecifies the fact that the value of `x` is not used (the `x` values are supposed to be regularly spaced, ie 1:<number of rows of `y`>). The user must anyway give a value for `x`, 1 for instance: `plot2d1("enn",1,y)`.

    o
    : means "one". If there are many curves, they all have the same x-values: `x` is a column vector of size nl and `y` is a matrix of size (nl,nc). For example : `x=[0:0.1:2*%pi]';plot2d1("onn",x,[sin(x) cos(x)])`.

    g
    : means "general". `x` and `y` must have the same size (nl,nc). Each column of `y` is plotted with respect to the corresponding column of `x`. `nc` curves are plotted using `nl` points.

  b, c
  : can have the values `n` (normal) or `l` (logarithmic).

    b=l
    : a logarithmic axis is used on the x-axis

    c=l
    : a logarithmic axis is used on the y-axis

x,y,[style,strf,leg,rect,nax]
: these arguments have the same meaning as in the `plot2d` function.

opt_args
: these arguments have the same meaning as in the `plot2d` function.

Description

This function is obsolete. USE `plot2d` INSTEAD !!

`plot2d1` plots a set of 2D curves. It is the same as `plot2d` but with one more argument `str` which enables logarithmic axis. Moreover, it allows to specify only one column vector for `x` when it is the same for all the curves.

By default, successive plots are superposed. To clear the previous plot, use `clf`.

Enter the command `plot2d1()` to see a demo.

Examples

```
// multiple plot without giving x
x=[0:0.1:2*%pi]';
plot2d1("enn",1,[sin(x) sin(2*x) sin(3*x)])
// multiple plot using only one x
clf()
plot2d1("onn",x,[sin(x) sin(2*x) sin(3*x)])
// logarithmic plot
x=[0.1:0.1:3]'; clf()
plot2d1("oll",x,[exp(x) exp(x^2) exp(x^3)])
```

See Also

plot2d , plot2d2 , plot2d3 , plot2d4 , clf

Authors

J.Ph.C.

Name

plot2d2 — 2D plot (step function)

```
plot2d2([x],y)
plot2d2([x],y,<opt_args>)
plot2d2([logflag],x,y,[style,strf,leg,rect,nax])
```

Parameters

args
: see `plot2d` for a description of parameters.

Description

`plot2d2` is the same as `plot2d` but the functions given by `(x,y)` are supposed to be piecewise constant.

By default, successive plots are superposed. To clear the previous plot, use `clf()`.

Enter the command `plot2d2()` to see a demo. Note that all the modes proposed by `plot2dxx` (xx = 1 to 4) can be enabled using `plot2d` and setting the `polyline_style` option to the corresponding number.

Examples

```
// plots a step function of value i on the segment [i,i+1]
// the last segment is not drawn
plot2d2([1:4],[1:4],1,"111","step function",[0,0,5,5])
// compare the following with plot2d
x=[0:0.1:2*%pi]';
clf()
plot2d2(x,[sin(x) sin(2*x) sin(3*x)])
  // In New graphics only
clf();
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
e=gce();
e.children(1).polyline_style=2;
e.children(2).polyline_style=2;
e.children(3).polyline_style=2;
```

See Also

plot2d , plot2d3 , plot2d4 , subplot , clf , polyline_properties

Authors

J.Ph.C.

Name

plot2d3 — 2D plot (vertical bars)

```
plot2d3([logflags,] x,y,[style,strf,leg,rect,nax])
plot232(y)
plot2d3(x,y <,opt_args>)
```

Parameters

args
: see `plot2d` for a description of parameters.

Description

`plot2d3` is the same as `plot2d` but curves are plotted using vertical bars.

By default, successive plots are superposed. To clear the previous plot, use `clf()`.

Enter the command `plot2d3()` to see a demo. Note that all the modes proposed by `plot2dxx` (xx = 1 to 4) can be enabled using `plot2d` and setting the `polyline_style` option to the corresponding number.

Examples

```
// compare the following with plot2d1
x=[0:0.1:2*%pi]';
plot2d3(x,[sin(x) sin(2*x) sin(3*x)])
// In New graphics only
clf()
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
e=gce();
e.children(1).polyline_style=3;
e.children(2).polyline_style=3;
e.children(3).polyline_style=3;
```

See Also

plot2d , plot2d2 , plot2d4 , clf , polyline_properties

Authors

J.Ph.C.

Name

plot2d4 — 2D plot (arrows style)

```
plot2d4([logflag,] x,y,[style,strf,leg,rect,nax])
plot2d4(y)
plot2d4(x,y <,opt_args>)
```

Parameters

args
: see `plot2d` for a description of parameters.

Description

`plot2d4` is the same as `plot2d` but curves are plotted using arrows style. This can be useful when plotting solutions of an ODE in a phase space.

By default, successive plots are superposed. To clear the previous plot, use `clf()`.

Enter the command `plot2d4()` to see a demo. Note that all the modes proposed by `plot2dxx` (xx = 1 to 4) can be enabled using `plot2d` and setting the `polyline_style` option to the corresponding number.

Examples

```


// compare the following with plot2d1
x=[0:0.1:2*%pi]';
plot2d4(x,[sin(x) sin(2*x) sin(3*x)])
 // In New graphics only
clf()
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
e=gce();
e.children(1).polyline_style=4;
e.children(2).polyline_style=4;
e.children(3).polyline_style=4;
```

See Also

fchamp , plot2d , plot2d2 , plot2d3 , subplot , clf , polyline_properties

Authors

J.Ph.C.

Name

plot2d_old_version — **The syntaxes described below are obsolete**

```
plot2d([logflag],x,y,[style,strf,leg,rect,nax])
```

Parameters

x,y
: two matrices (or column vectors).

  - in the usual way `x` is a matrix of the same size than `y` (the column j of `y` is plotted with respect to column j of `x`)
  - if all the columns of `x` are equal (ie the abscissae of all the curves are the same), `x` may be simply the (column) vector of these abscissae (`x` is then a column vector of length equal to the row dimension of `y`).
  - when `x` is not given, it is supposed to be the column vector [1; 2; ...; row dimension of y].

style
: is a real row vector of size nc. The style to use for curve `i` is defined by `style(i)`. The default style is `1:nc` (1 for the first curve, 2 for the second, etc.).

  - if `style(i)` is negative or zero, the curve is plotted using the mark with id `abs(style(i))`; use `xset()` to set the mark id and xget('mark') to get the current mark id.
  - if `style(i)` is strictly positive, a plain line with color id `style(i)` or a dashed line with dash id `style(i)` is used; use `xset()` to see the color ids.
  - When only one curve is drawn, `style` could be the row vector of size 2 `[sty,pos]` where `sty` is used to specify the style and `pos` is an integer ranging from 1 to 6 which specifies a position to use for the caption. This was useful when a user wants to draw multiple curves on a plot by calling the function `plot2d` several times and wants to give a caption for each curve. **This option is no more effective with the new graphic mode. Use the captions function to set the captions when all curves are drawn.**

strf
: is a string of length 3 `"xyz"` (by default `strf= "081"`)

  x
  : controls the display of captions.

    x=0
    : no caption.

    x=1
    : captions are displayed. They are given by the optional argument `leg`.

  y
  : controls the computation of the actual coordinate ranges from the minimal requested values. Actual ranges can be larger than minimal requirements.

    y=0
    : no computation, the plot use the previus (or default) scale

    y=1
    : from the rect arg

y=2
: from the min/max of the x, y datas

y=3
: built for an isometric scale from the rect arg

y=4
: built for an isometric plot from the min/max of the x, y datas

y=5
: enlarged for pretty axes from the rect arg

y=6
: enlarged for pretty axes from the min/max of the x, y datas

y=7
: like y=1 but the previus plot(s) are redrawn to use the new scale

y=8
: like y=2 but the previus plot(s) are redrawn to use the new scale

z
: controls the display of information on the frame around the plot. If axes are requested, the number of tics can be specified by the `nax` optional argument.

z=0
: nothing is drawn around the plot.

z=1
: axes are drawn, the y=axis is displayed on the left.

z=2
: the plot is surrounded by a box without tics.

z=3
: axes are drawn, the y=axis is displayed on the right.

z=4
: axes are drawn centred in the middle of the frame box.

z=5
: axes are drawn so as to cross at point `(0,0)`. If point `(0,0)` does not lie inside the frame, axes will not appear on the graph.

leg
: a string. It is used when the first character x of argument `strf` is 1. `leg` has the form `"leg1@leg2@...."` where `leg1`, `leg2`, etc. are respectively the captions of the first curve, of the second curve, etc. The default is `""`.

rect
: This argument is used when the second character y of argument `strf` is 1, 3 or 5. It is a row vector of size 4 and gives the dimension of the frame: `rect=[xmin,ymin,xmax,ymax]`.

nax
: This argument is used when the third character z of argument `strf` is 1. It is a row vector with four entries `[nx,Nx,ny,Ny]` where `nx` (`ny`) is the number of subgraduations on the x (y) axis and `Nx` (`Ny`) is the number of graduations on the x (y) axis.

logflag
: a string formed by to characters h (for horizontal axis) and v (for vertical axis) each of these characters can take the values "n" or "l". "l" stands for logarithmic graduation and "n" for normal graduation. For example "ll"stands for a log-log plot. Default value is "nn".

Description

`plot2d` plots a set of 2D curves. Piecewise linear plotting is used.

By default, successive plots are superposed. To clear the previous plot, use `xbasc()`.

See the meaning of the parameters above for a complete description.

Enter the command `plot2d()` to see a demo.

Other high level plot2d function exists:

- plot2d2: same as `plot2d` but the curve is supposed to be piecewise constant.
- plot2d3: same as `plot2d` but the curve is plotted with vertical bars.
- plot2d4: same as `plot2d` but the curve is plotted with vertical arrows.

Examples

```
//simple plot
x=[0:0.1:2*%pi]';
plot2d(sin(x))
xbasc()
plot2d(x,sin(x))
//multiple plot
xbasc()
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
// multiple plot giving the dimensions of the frame
// old syntax and new syntax
xbasc()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],1:3,"011","",[0,0,6,0.5])
xbasc()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],rect=[0,0,6,0.5])
//multiple plot with captions and given tics // old syntax and new syntax
xbasc()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],..
  [1,2,3],"111","L1@L2@L3",[0,-2,2*%pi,2],[2,10,2,10]);
xbasc()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],..
  [1,2,3],leg="L1@L2@L3",nax=[2,10,2,10],rect=[0,-2,2*%pi,2])
// isoview
xbasc()
plot2d(x,sin(x),1,"041")
// scale
xbasc()
plot2d(x,sin(x),1,"061")
// auto scaling with previous plots
xbasc()
plot2d(x,sin(x),1)
plot2d(x,2*sin(x),2)
plot2d(2*x,cos(x),3)
// axis on the right
xbasc()
plot2d(x,sin(x),1,"183","sin(x)")
// centered axis
```

```
xbasc()
plot2d(x,sin(x),1,"184","sin(x)")
// axis centered at (0,0)
xbasc()
plot2d(x-4,sin(x),1,"185","sin(x)")
```

See Also

plot2d1, plot2d2, plot2d3, plot2d4, xbasc, xset

Authors

J.Ph.C.

Name

plot3d — 3D plot of a surface

```
plot3d(x,y,z,[theta,alpha,leg,flag,ebox])
plot3d(x,y,z,<opt_args>)

plot3d(xf,yf,zf,[theta,alpha,leg,flag,ebox])
plot3d(xf,yf,zf,<opt_args>)

plot3d(xf,yf,list(zf,colors),[theta,alpha,leg,flag,ebox])
plot3d(xf,yf,list(zf,colors),<opt_args>)
```

Parameters

x,y
: row vectors of sizes n1 and n2 (x-axis and y-axis coordinates). These coordinates must be monotone.

z
: matrix of size (n1,n2). `z(i,j)` is the value of the surface at the point (x(i),y(j)).

xf,yf,zf
: matrices of size (nf,n). They define the facets used to draw the surface. There are `n` facets. Each facet `i` is defined by a polygon with `nf` points. The x-axis, y-axis and z-axis coordinates of the points of the ith facet are given respectively by `xf(:,i)`, `yf(:,i)` and `zf(:,i)`.

colors
: a vector of size n giving the color of each facets or a matrix of size (nf,n) giving color near each facet boundary (facet color is interpolated).

<opt_args>
: This represents a sequence of statements `key1=value1, key2=value2`,... where `key1`, `key2,...` can be one of the following: theta, alpha ,leg,flag,ebox (see definition below).

theta, alpha
: real values giving in degree the spherical coordinates of the observation point.

leg
: string defining the labels for each axis with @ as a field separator, for example "X@Y@Z".

flag
: a real vector of size three. `flag=[mode,type,box]`.

  mode
  : an integer (surface color).

    mode>0
    : the surface is painted with color `"mode"` ; the boundary of the facet is drawn with current line style and color.

    mode=0:
    : a mesh of the surface is drawn.

    mode<0:
    : the surface is painted with color `"-mode"` ; the boundary of the facet is not drawn.

      Note that the surface color treatement can be done using `color_mode` and `color_flag` options through the surface entity properties (see surface_properties).

  type
  : an integer (scaling).

type=0:
the plot is made using the current 3D scaling (set by a previous call to `param3d`, `plot3d`, `contour` or `plot3d1`).

type=1:
rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument `ebox`.

type=2:
rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data.

type=3:
3d isometric with box bounds given by optional `ebox`, similarily to `type=1`.

type=4:
3d isometric bounds derived from the data, to similarily`type=2`.

type=5:
3d expanded isometric bounds with box bounds given by optional `ebox`, similarily to `type=1`.

type=6:
3d expanded isometric bounds derived from the data, similarily to `type=2`.

Note that axes boundaries can be customized through the axes entity properties (see axes_properties).

box
an integer (frame around the plot).

box=0:
nothing is drawn around the plot.

box=1:
unimplemented (like box=0).

box=2:
only the axes behind the surface are drawn.

box=3:
a box surrounding the surface is drawn and captions are added.

box=4:
a box surrounding the surface is drawn, captions and axes are added.

Note that axes aspect can also be customized through the axes entity properties (see axes_properties).

ebox
It specifies the boundaries of the plot as the vector `[xmin,xmax,ymin,ymax,zmin,zmax]`. This argument is used together with `type` in `flag` : if it is set to `1`, `3` or `5` (see above to see the corresponding behaviour). If `flag` is missing, `ebox` is not taken into acoount.

Note that, when specified, the `ebox` argument acts on the `data_bounds` field that can also be reset through the axes entity properties (see axes_properties).

Description

`plot3d(x,y,z,[theta,alpha,leg,flag,ebox])` draws the parametric surface `z=f(x,y)`.

`plot3d(xf,yf,zf,[theta,alpha,leg ,flag,ebox])` draws a surface defined by a set of facets. You can draw multiple plots by replacing `xf`, `yf` and `zf` by multiple matrices assembled by rows as `[xf1 xf2 ...]`, `[yf1 yf2 ...]` and `[zf1 zf2 ...]`. Note that data can also be set or get through the surface entity properties (see surface_properties).

You can give a specific color for each facet by using `list(zf,colors)` instead of `zf`, where `colors` is a vector of size `n`. If `colors(i)` is positive it gives the color of facet `i` and the boundary of the facet is drawn with current line style and color. If `colors(i)` is negative, color id `-colors(i)` is used and the boundary of the facet is not drawn.

It is also possible to get interpolated color for facets. For that the color argument must be a matrix of size nfxn giving the color near each boundary of each facets. In this case positive values for colors mean that the boundary are not drawn. Note that colors can also be set through the surface entity properties (via `tlist` affectations) and edited using `color_flag` option (see surface_properties).

The optional arguments `theta, alpha, leg ,flag, ebox`, can be passed by a sequence of statements `key1=value1, key2=value2`, ... In this case, the order has no special meaning. Note that all these optional arguments except `flag` can be customized through the axes entity properties (see axes_properties). As described before, the `flag` option deals with surface entity properties for `mode` (see surface_properties) and axes properties for `type` and `box` (see axes_properties).

You can use the function `genfac3d` to compute four sided facets from the surface `z=f(x,y)`. `eval3dp` can also be used.

Enter the command `plot3d()` to see a demo.

More information

To get more information on the plot3d old syntax , use the plot3d_old_version help document.

Examples

```
// simple plot using z=f(x,y)
t=[0:0.3:2*%pi]';
z=sin(t)*cos(t');
plot3d(t,t,z)
// same plot using facets computed by genfac3d
[xx,yy,zz]=genfac3d(t,t,z);
clf()
plot3d(xx,yy,zz)
// multiple plots
clf()
plot3d([xx xx],[yy yy],[zz 4+zz])
// multiple plots using colors
clf()
plot3d([xx xx],[yy yy],list([zz zz+4],[4*ones(1,400) 5*ones(1,400)]))
// simple plot with viewpoint and captions
clf()
plot3d(1:10,1:20,10*rand(10,20),alpha=35,theta=45,flag=[2,2,3])
// plot of a sphere using facets computed by eval3dp
deff("[x,y,z]=sph(alp,tet)",["x=r*cos(alp).*cos(tet)+orig(1)*ones(tet)";..
 "y=r*cos(alp).*sin(tet)+orig(2)*ones(tet)";..
 "z=r*sin(alp)+orig(3)*ones(tet)"]);
```

```
r=1; orig=[0 0 0];
[xx,yy,zz]=eval3dp(sph,linspace(-%pi/2,%pi/2,40),linspace(0,%pi*2,20));
clf();plot3d(xx,yy,zz)
clf();
f=gcf();
f.color_map = hotcolormap(128);
r=0.3;orig=[1.5 0 0];
[xx1,yy1,zz1]=eval3dp(sph,linspace(-%pi/2,%pi/2,40),linspace(0,%pi*2,20));
cc=(xx+zz+2)*32;cc1=(xx1-orig(1)+zz1/r+2)*32;
clf();plot3d1([xx xx1],[yy yy1],list([zz,zz1],[cc cc1]),theta=70,alpha=80,flag=

//Available operations using only New Graphics...
delete(gcf());
t=[0:0.3:2*%pi]'; z=sin(t)*cos(t');
[xx,yy,zz]=genfac3d(t,t,z);
plot3d([xx xx],[yy yy],list([zz zz+4],[4*ones(1,400) 5*ones(1,400)]))
e=gce();
f=e.data;
TL = tlist(["3d" "x" "y" "z" "color"],f.x,f.y,f.z,6*rand(f.z)); // random color
e.data = TL;
TL2 = tlist(["3d" "x" "y" "z" "color"],f.x,f.y,f.z,4*rand(1,800)); // random co
e.data = TL2;
TL3 = tlist(["3d" "x" "y" "z" "color"],f.x,f.y,f.z,[20*ones(1,400) 6*ones(1,400
e.data = TL3;
TL4 = tlist(["3d" "x" "y" "z"],f.x,f.y,f.z); // no color
e.data = TL4;
e.color_flag=1 // color index proportional to altitude (z coord.)
e.color_flag=2; // back to default mode
e.color_flag= 3; // interpolated shading mode (based on blue default color)
clf()
plot3d([xx xx],[yy yy],list([zz zz+4],[4*ones(1,400) 5*ones(1,400)]))
h=gce(); //get handle on current entity (here the surface)
a=gca(); //get current axes
a.rotation_angles=[40,70];
a.grid=[1 1 1]; //make grids
a.data_bounds=[-6,0,-1;6,6,5];
a.axes_visible="off"; //axes are hidden
a.axes_bounds=[.2 0 1 1];
h.color_flag=1; //color according to z
h.color_mode=-2;  //remove the facets boundary by setting color_mode to white c
h.color_flag=2; //color according to given colors
h.color_mode = -1; // put the facets boundary back by setting color_mode to bla
f=gcf();//get the handle of the parent figure
f.color_map=hotcolormap(512);
c=[1:400,1:400];
TL.color = [c;c+1;c+2;c+3];
h.data = TL;
h.color_flag=3; // interpolated shading mode
```

See Also

eval3dp , genfac3d , geom3d , param3d , plot3d1 , clf , gca , gcf , xdel , delete

Authors

J.Ph.C.

Name

plot3d1 — 3D gray or color level plot of a surface

```
plot3d1(x,y,z,[theta,alpha,leg,flag,ebox])
plot3d1(xf,yf,zf,[theta,alpha,leg,flag,ebox])

plot3d1(x,y,z,<opts_args>)
plot3d1(xf,yf,zf,<opts_args>)
```

Parameters

x,y
: row vectors of sizes n1 and n2 (x-axis and y-axis coordinates). These coordinates must be monotone.

z
: matrix of size (n1,n2). `z(i,j)` is the value of the surface at the point (x(i),y(j)).

xf,yf,zf
: matrices of size (nf,n). They define the facets used to draw the surface. There are `n` facets. Each facet `i` is defined by a polygon with `nf` points. The x-axis, y-axis and z-axis coordinates of the points of the ith facet are given respectively by `xf(:,i)`, `yf(:,i)` and `zf(:,i)`.

<opt_args>
: This represents a sequence of statements `key1=value1, key2=value2`,... where `key1`, `key2,...` can be one of the following: theta, alpha ,leg,flag,ebox (see definition below).

theta, alpha
: real values giving in degree the spherical coordinates of the observation point.

leg
: string defining the labels for each axis with @ as a field separator, for example "X@Y@Z".

flag
: a real vector of size three. `flag=[mode,type,box]`.

  mode
  : an integer (surface color).

    mode>0
    : the surface is painted with color `"mode"` ; the boundary of the facet is drawn with current line style and color.

    mode=0:
    : a mesh of the surface is drawn.

    mode<0:
    : the surface is painted with color `"-mode"` ; the boundary of the facet is not drawn.

      Note that the surface color treatement can be done using `color_mode` and `color_flag` options through the surface entity properties (see surface_properties).

  type
  : an integer (scaling).

    type=0:
    : the plot is made using the current 3D scaling (set by a previous call to `param3d`, `plot3d`, `contour` or `plot3d1`).

type=1:
rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument `ebox`.

type=2:
rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data.

type=3:
3d isometric with box bounds given by optional `ebox`, similarily to `type=1`.

type=4:
3d isometric bounds derived from the data, to similarily`type=2`.

type=5:
3d expanded isometric bounds with box bounds given by optional `ebox`, similarily to `type=1`.

type=6:
3d expanded isometric bounds derived from the data, similarily to `type=2`.

Note that axes boundaries can be customized through the axes entity properties (see axes_properties).

box
an integer (frame around the plot).

box=0:
nothing is drawn around the plot.

box=1:
unimplemented (like box=0).

box=2:
only the axes behind the surface are drawn.

box=3:
a box surrounding the surface is drawn and captions are added.

box=4:
a box surrounding the surface is drawn, captions and axes are added.

Note that axes aspect can also be customized through the axes entity properties (see axes_properties).

ebox
It specifies the boundaries of the plot as the vector `[xmin,xmax,ymin,ymax,zmin,zmax]`. This argument is used together with `type` in `flag` : if it is set to `1`, `3` or `5` (see above to see the corresponding behaviour). If `flag` is missing, `ebox` is not taken into acoount.

Note that, when specified, the `ebox` argument acts on the `data_bounds` field that can also be reset through the axes entity properties (see axes_properties).

Description

`plot3d1` plots a surface with colors depending on the z-level of the surface. This special plot function can also be enabled setting `color_flag=1` after a `plot3d` (see surface_properties)

Enter the command `plot3d1()` to see a demo.

Examples

```
// simple plot using z=f(x,y)
t=[0:0.3:2*%pi]'; z=sin(t)*cos(t');
plot3d1(t,t,z)
// same plot using facets computed by genfac3d
[xx,yy,zz]=genfac3d(t,t,z);
clf();
plot3d1(xx,yy,zz)
// multiple plots
clf();
plot3d1([xx xx],[yy yy],[zz 4+zz])
// simple plot with viewpoint and captions
clf() ;
plot3d1(1:10,1:20,10*rand(10,20),35,45,"X@Y@Z",[2,2,3])
// same plot without grid
clf()
plot3d1(1:10,1:20,10*rand(10,20),35,45,"X@Y@Z",[-2,2,3])
// plot of a sphere using facets computed by eval3dp
deff("[x,y,z]=sph(alp,tet)",["x=r*cos(alp).*cos(tet)+orig(1)*ones(tet)";..
"y=r*cos(alp).*sin(tet)+orig(2)*ones(tet)";..
"z=r*sin(alp)+orig(3)*ones(tet)"]);
r=1; orig=[0 0 0];
[xx,yy,zz]=eval3dp(sph,linspace(-%pi/2,%pi/2,40),linspace(0,%pi*2,20));
clf()

plot3d(xx,yy,zz)
e=gce();
e.color_flag=1;
scf(2);
plot3d1(xx,yy,zz) // the 2 graphics are similar
```

See Also

plot3d , gca , gce , scf , clf

Authors

J.Ph.C.

Name

plot3d2 — plot surface defined by rectangular facets

```
plot3d2(X,Y,Z [,vect,theta,alpha,leg,flag,ebox])
plot3d2(X,Y,Z, <opt_args>)
```

Parameters

X, Y, Z:
: 3 real matrices defining a data structure.

vect
: a real vector.

<opt_args>
: This represents a sequence of statements `key1=value1, key2=value2`,... where `key1`, `key2,...` can be one of the following: theta, alpha ,leg,flag,ebox (see definition below).

theta, alpha
: real values giving in degree the spherical coordinates of the observation point.

leg
: string defining the labels for each axis with @ as a field separator, for example "X@Y@Z".

flag
: a real vector of size three. `flag=[mode,type,box]`.

  mode
  : an integer (surface color).

    mode>0
    : the surface is painted with color `"mode"` ; the boundary of the facet is drawn with current line style and color.

    mode=0:
    : a mesh of the surface is drawn.

    mode<0:
    : the surface is painted with color `"-mode"` ; the boundary of the facet is not drawn.

      Note that the surface color treatement can be done using `color_mode` and `color_flag` options through the surface entity properties (see surface_properties).

  type
  : an integer (scaling).

    type=0:
    : the plot is made using the current 3D scaling (set by a previous call to `param3d`, `plot3d`, `contour` or `plot3d1`).

    type=1:
    : rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument `ebox`.

    type=2:
    : rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data.

    type=3:
    : 3d isometric with box bounds given by optional `ebox`, similarily to `type=1`.

type=4:
3d isometric bounds derived from the data, to similarily`type=2`.

type=5:
3d expanded isometric bounds with box bounds given by optional `ebox`, similarily to `type=1`.

type=6:
3d expanded isometric bounds derived from the data, similarily to `type=2`.

Note that axes boundaries can be customized through the axes entity properties (see axes_properties).

box
an integer (frame around the plot).

box=0:
nothing is drawn around the plot.

box=1:
unimplemented (like box=0).

box=2:
only the axes behind the surface are drawn.

box=3:
a box surrounding the surface is drawn and captions are added.

box=4:
a box surrounding the surface is drawn, captions and axes are added.

Note that axes aspect can also be customized through the axes entity properties (see axes_properties).

ebox
It specifies the boundaries of the plot as the vector `[xmin,xmax,ymin,ymax,zmin,zmax]`. This argument is used together with `type` in `flag` : if it is set to `1`, `3` or `5` (see above to see the corresponding behaviour). If `flag` is missing, `ebox` is not taken into acoount.

Note that, when specified, the `ebox` argument acts on the `data_bounds` field that can also be reset through the axes entity properties (see axes_properties).

Description

`plot3d2` plots a surface defined by rectangular facets. (X,Y,Z) are three matrices which describe a surface. The surface is composed of four sided polygons.

The X-coordinates of a facet are given by X(i,j), X(i+1,j), X(i+1,j+1) and X(i,j+1). Similarly Y and Z matrices contain Y and Z-coordinates.

The `vect` vector is used when multiple surfaces are coded in the same (X,Y,Z) matrices. `vect(j)` gives the line at which the coding of the jth surface begins. Like in `plot3d`, the same properties are editable (see surface_properties and axes_properties).

Examples

```
u = linspace(-%pi/2,%pi/2,40);
v = linspace(0,2*%pi,20);
```

```
X = cos(u)'*cos(v);
Y = cos(u)'*sin(v);
Z = sin(u)'*ones(v);
plot3d2(X,Y,Z);
 // New Graphic mode only
 e=gce();
e.color_mode=4; // change color
f=e.data;
TL = tlist(["3d" "x" "y" "z" "color"],f.x,f.y,f.z,10*(f.z)+1);
e.data=TL;
e.color_flag=2;
```

See Also

plot3d , genfac3d

Name

plot3d3 — mesh plot surface defined by rectangular facets

```
plot3d3(X,Y,Z [,vect,theta,alpha,leg,flag,ebox])
plot3d3(X,Y,Z, <opt_args>)
```

Parameters

X, Y, Z:
: 3 real matrices defining a data structure.

vect
: a real vector.

<opt_args>
: This represents a sequence of statements `key1=value1, key2=value2`,... where `key1`, `key2,...` can be one of the following: theta, alpha ,leg,flag,ebox (see definition below).

theta, alpha
: real values giving in degree the spherical coordinates of the observation point.

leg
: string defining the labels for each axis with @ as a field separator, for example "X@Y@Z".

flag
: a real vector of size four. `flag=[vertical_color, horizontal_color,type,box]`.

  vertical_color
  : an integer (surface color), default is 3.

    Colormap index defining the color used to draw vertical edges.

  horizontal_color
  : an integer (surface color), default is 4.

    Colormap index defining the color used to draw horizontal edges.

  type
  : an integer (scaling) default is 2.

    type=0:
    : the plot is made using the current 3D scaling (set by a previous call to `param3d`, `plot3d`, `contour` or `plot3d1`).

    type=1:
    : rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument `ebox`.

    type=2:
    : rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data.

    type=3:
    : 3d isometric with box bounds given by optional `ebox`, similarily to `type=1`.

    type=4:
    : 3d isometric bounds derived from the data, to similarily`type=2`.

type=5:
3d expanded isometric bounds with box bounds given by optional `ebox`, similarily to `type=1`.

type=6:
3d expanded isometric bounds derived from the data, similarily to `type=2`.

Note that axes boundaries can be customized through the axes entity properties (see axes_properties).

box
an integer (frame around the plot), default is 4.

box=0:
nothing is drawn around the plot.

box=1:
unimplemented (like box=0).

box=2:
only the axes behind the surface are drawn.

box=3:
a box surrounding the surface is drawn and captions are added.

box=4:
a box surrounding the surface is drawn, captions and axes are added.

Note that axes aspect can also be customized through the axes entity properties (see axes_properties).

ebox
It specifies the boundaries of the plot as the vector `[xmin,xmax,ymin,ymax,zmin,zmax]`. This argument is used together with `type` in `flag` : if it is set to `1`, `3` or `5` (see above to see the corresponding behaviour). If `flag` is missing, `ebox` is not taken into acoount. Note that, when specified, the `ebox` argument acts on the `data_bounds` field that can also be reset through the axes entity properties (see axes_properties).

Description

`plot3d3` performs a mesh plot of a surface defined by rectangular facets. (X,Y,Z) are three matrices which describe a surface. The surface is composed of four sided polygons.

The X-coordinates of a facet are given by X(i,j), X(i+1,j), X(i+1,j+1) and X(i,j+1). Similarly Y and Z matrices contain Y and Z-coordinates.

The `vect` vector is used when multiple surfaces are coded in the same (X,Y,Z) matrices. `vect(j)` gives the line at which the coding of the jth surface begins. See `plot3d2` for a full description. As a mesh plot, the only available property you can edit is the `visible` option (see axes_properties).

Examples

```
u = linspace(-%pi/2,%pi/2,40);
v = linspace(0,2*%pi,20);
X = cos(u)'*cos(v);
Y = cos(u)'*sin(v);
Z = sin(u)'*ones(v);
```

```
plot3d3(X,Y,Z);
 // New Graphic mode only
e=gce(); // get the current entity
e.visible='off';
e.visible='on'; // back to the mesh view
```

See Also

plot3d2 , plot3d , param3d

Name

plot3d_old_version — 3D plot of a surface

```
plot3d(x,y,z,[theta,alpha,leg,flag,ebox])
plot3d(x,y,z,<opt_args>)

plot3d(xf,yf,zf,[theta,alpha,leg,flag,ebox])
plot3d(xf,yf,zf,<opt_args>)

plot3d(xf,yf,list(zf,colors),[theta,alpha,leg,flag,ebox])
plot3d(xf,yf,list(zf,colors),<opt_args>)
```

Parameters

x,y
: row vectors of sizes n1 and n2 (x-axis and y-axis coordinates). These coordinates must be monotone.

z
: matrix of size (n1,n2). `z(i,j)` is the value of the surface at the point (x(i),y(j)).

xf,yf,zf
: matrices of size (nf,n). They define the facets used to draw the surface. There are `n` facets. Each facet `i` is defined by a polygon with `nf` points. The x-axis, y-axis and z-axis coordinates of the points of the ith facet are given respectively by `xf(:,i)`, `yf(:,i)` and `zf(:,i)`.

colors
: a vector of size n giving the color of each facets or a matrix of size (nf,n) giving color near each facet boundary (facet color is interpolated)

<opt_args>
: This represents a sequence of statements `key1=value1, key2=value2`,... where `key1`, `key2,...` can be one of the following: theta, alpha ,leg,flag,ebox (see definition below)

theta, alpha
: real values giving in degree the spherical coordinates of the observation point.

leg
: string defining the captions for each axis with @ as a field separator, for example "X@Y@Z".

flag
: a real vector of size three `flag=[mode,type,box]`.

  mode
  : string (treatment of hidden parts).

    mode>0
    : the hidden parts of the surface are removed and the surface is painted with color `mode`.

    mode=0
    : the hidden parts of the surface are drawn.

    mode<0
    : only the backward facing facets are painted with color or pattern id `-mode`. Use `xset()` to see the meaning of the ids.

  type
  : an integer (scaling).

type=0
: the plot is made using the current 3D scaling (set by a previous call to `param3d`, `plot3d`, `contour` or `plot3d1`).

type=1
: rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument `ebox`.

type=2
: rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data.

type=3
: 3d isometric with box bounds given by optional `ebox`, similarily to `type=1`

type=4
: 3d isometric bounds derived from the data, to similarily`type=2`

type=5
: 3d expanded isometric bounds with box bounds given by optional `ebox`, similarily to `type=1`

type=6
: 3d expanded isometric bounds derived from the data, similarily to `type=2`

box
: an integer (frame around the plot).

box=0
: nothing is drawn around the plot.

box=1
: unimplemented (like box=0).

box=2
: only the axes behind the surface are drawn.

box=3
: a box surrounding the surface is drawn and captions are added.

box=4
: a box surrounding the surface is drawn, captions and axes are added.

ebox
: used when `type` in `flag` is 1. It specifies the boundaries of the plot as the vector `[xmin,xmax,ymin,ymax,zmin,zmax]`.

Description

`plot3d(x,y,z,[theta,alpha,leg,flag,ebox])` draws the parametric surface `z=f(x,y)`.

`plot3d(xf,yf,zf,[theta,alpha,leg ,flag,ebox])` draws a surface defined by a set of facets. You can draw multiple plots by replacing `xf`, `yf` and `zf` by multiple matrices assembled by rows as `[xf1 xf2 ...]`, `[yf1 yf2 ...]` and `[zf1 zf2 ...]`.

You can give a specific color for each facet by using `list(zf,colors)` instead of `zf`, where `colors` is a vector of size `n`. If `colors(i)` is positive it gives the color of facet `i` and the boundary of the facet is drawn with current line style and color. If `colors(i)` is negative, color id `-colors(i)` is used and the boundary of the facet is not drawn. Use `xset()` to see the ids of the colors.

It is also possible to get interpolated color for facets. For that the color argument must be a matrix of size nfxn giving the color near each boundary of each facets. In this case positive values for colors mean that the boundary are not drawn.

The optional arguments `theta,alpha,leg ,flag,ebox`, can be passed by a sequence of statements `key1=value1, key2=value2`, ... In this case, the order has no special meaning.

You can use the function `genfac3d` to compute four sided facets from the surface `z=f(x,y)`. `eval3dp` can also be used.

Enter the command `plot3d()` to see a demo.

Examples

```
// simple plot using z=f(x,y)
t=[0:0.3:2*%pi]'; z=sin(t)*cos(t');
plot3d(t,t,z)
// same plot using facets computed by genfac3d
[xx,yy,zz]=genfac3d(t,t,z);
xbasc()
plot3d(xx,yy,zz)
// multiple plots
xbasc()
plot3d([xx xx],[yy yy],[zz 4+zz])
// multiple plots using colors
xbasc()
plot3d([xx xx],[yy yy],list([zz zz+4],[4*ones(1,400) 5*ones(1,400)]))
// simple plot with viewpoint and captions
xbasc()
plot3d(1:10,1:20,10*rand(10,20),35,45,"X@Y@Z",[2,2,3])
// plot of a sphere using facets computed by eval3dp
deff("[x,y,z]=sph(alp,tet)",["x=r*cos(alp).*cos(tet)+orig(1)*ones(tet)";..
  "y=r*cos(alp).*sin(tet)+orig(2)*ones(tet)";..
  "z=r*sin(alp)+orig(3)*ones(tet)"]);
r=1; orig=[0 0 0];
[xx,yy,zz]=eval3dp(sph,linspace(-%pi/2,%pi/2,40),linspace(0,%pi*2,20));
xbasc();plot3d(xx,yy,zz)

xbasc();xset('colormap',hotcolormap(128));
r=0.3;orig=[1.5 0 0];
[xx1,yy1,zz1]=eval3dp(sph,linspace(-%pi/2,%pi/2,40),linspace(0,%pi*2,20));
cc=(xx+zz+2)*32;cc1=(xx1-orig(1)+zz1/r+2)*32;
xbasc();plot3d1([xx xx1],[yy yy1],list([zz,zz1],[cc cc1]),70,80)

xbasc();plot3d1([xx xx1],[yy yy1],list([zz,zz1],[cc cc1]),theta=70,alpha=80,fla
```

See Also

eval3dp , genfac3d , geom3d , param3d , plot3d1 , xset

Authors

J.Ph.C.

Name

plotframe — plot a frame with scaling and grids. **This function is obsolete.**

```
plotframe(rect,tics,[arg_opt1,arg_opt2,arg_opt3])
plotframe(rect,<opts_args>)
```

Parameters

rect
: vector `[xmin,ymin,xmax,ymax]`.

tics
: vector `[nx,mx,ny,my]` where `mx, nx` (resp. `my, ny`) are the number of x-axis (resp. y-axis) intervals and subintervals.

arg_optX
: optional arguments up to three and choosen among.

  flags
  : vector `[wantgrids,findbounds]` where `wantgrids` is a boolean variable (`%t` or `%f`) which indicates gridding. `findbounds` is a boolean variable. If `findbounds` is `%t`, the bounds given in `rect` are allowed to be slightly modified (in fact always increased) in order to have simpler graduations: then `tics(2)` and `tics(4)` are ignored.

  Captions
  : vector of 3 strings `[title,x-leg,y-leg]` corresponding respectively to the title of the plot and the captions on the x-axis and the y-axis. Warning: the upper-case "C" is important.

  subwin
  : a vector of size 4 defining the sub window. The sub window is specified with the parameter `subwin=[x,y,w,h]` (upper-left, width, height). The values in `subwin` are specified using proportion of the width or height of the current graphics window (see `xsetech`).

<opts_args>
: This represents a sequence of statements **key1=value1**, **key2=value2**,... where **key1**, **key2**,... can be one of the following: **tics**, **flags**, **captions** ou **subwin**. These arguments have the same meaning as the ones used in the first form of the routine.

Description

`plotframe` is used with 2D plotting functions plot2d, plot2d1,... to set a graphics frame. It must be used before `plot2d` which should be invoked with the "000" superposition mode.

This function is obsolete.

Examples

```
x=[-0.3:0.8:27.3]';
y=rand(x);
rect=[min(x),min(y),max(x),max(y)];
tics=[4,10,2,5];    //4 x-intervals and 2 y-intervals
plotframe(rect,tics,[%f,%f],["My plot","x","y"],[0,0,0.5,0.5])
plot2d(x,y,2,"000")
plotframe(rect,tics=tics,flags=[%t,%f],Captions=["My plot with grids","x","y"],
plot2d(x,y,3,"000")
```

```
plotframe(rect,tics,[%t,%t],..
["My plot with grids and automatic bounds","x","y"],[0,0.5,0.5,0.5])
plot2d(x,y,4,"000")
plotframe(rect,flags=[%f,%t],tics=tics,..
   Captions=["My plot without grids but with automatic bounds ","x","y"],..
   subwin=[0.5,0.5,0.5,0.5])
plot2d(x,y,5,"000")
```

See Also

plot2d, graduate, xsetech

Name

plzr — pole-zero plot

```
plzr(sl)
```

Parameters

sl
: list (`syslin`)

Description

produces a pole-zero plot of the linear system `sl` (`syslin` list)

Examples

```
s=poly(0,'s');
n=[1+s   2+3*s+4*s^2        5; 0        1-s             s];
d=[1+3*s   5-s^3          s+1;1+s     1+s+s^2      3*s-1];
h=syslin('c',n./d);
plzr(h);
```

See Also

trzeros , roots , syslin

Name

polarplot — Plot polar coordinates

```
polarplot(theta,rho,[style,strf,leg,rect])
polarplot(theta,rho,<opt_args>)
```

Parameters

rho
: a vector, the radius values

theta
: a vector with same size than rho, the angle values.

<opt_args>
: a sequence of statements `key1=value1, key2=value2`, ... where keys may be `style`,`leg`,`rect`,`strf` or `frameflag`

style
: is a real row vector of size nc. The style to use for curve `i` is defined by `style(i)`. The default style is `1:nc` (1 for the first curve, 2 for the second, etc.).

  -
  : if `style(i)` is negative, the curve is plotted using the mark with id `abs(style(i))+1`; use `xset()` to see the mark ids.

  -
  : if `style(i)` is strictly positive, a plain line with color id `style(i)` or a dashed line with dash id `style(i)` is used; use `xset()` to see the color ids.

  -
  : When only one curve is drawn, `style` can be the row vector of size 2 `[sty,pos]` where `sty` is used to specify the style and `pos` is an integer ranging from 1 to 6 which specifies a position to use for the caption. This can be useful when a user wants to draw multiple curves on a plot by calling the function `plot2d` several times and wants to give a caption for each curve.

strf
: is a string of length 3 `"xy0"`.

  default
  : The default is `"030"`.

  x
  : controls the display of captions,

    x=0
    : no captions.

    x=1
    : captions are displayed. They are given by the optional argument `leg`.

  y
  : controls the computation of the frame. same as frameflag

    y=0
    : the current boundaries (set by a previous call to another high level plotting function) are used. Useful when superposing multiple plots.

y=1
: the optional argument `rect` is used to specify the boundaries of the plot.

y=2
: the boundaries of the plot are computed using min and max values of `x` and `y`.

y=3
: like `y=1` but produces isoview scaling.

y=4
: like `y=2` but produces isoview scaling.

y=5
: like `y=1` but `plot2d` can change the boundaries of the plot and the ticks of the axes to produce pretty graduations. When the zoom button is activated, this mode is used.

y=6
: like `y=2` but `plot2d` can change the boundaries of the plot and the ticks of the axes to produce pretty graduations. When the zoom button is activated, this mode is used.

y=7
: like `y=5` but the scale of the new plot is merged with the current scale.

y=8
: like `y=6` but the scale of the new plot is merged with the current scale.

leg
: a string. It is used when the first character x of argument `strf` is 1. `leg` has the form `"leg1@leg2@...."` where `leg1`, `leg2`, etc. are respectively the captions of the first curve, of the second curve, etc. The default is `""`.

rect
: This argument is used when the second character y of argument `strf` is 1, 3 or 5. It is a row vector of size 4 and gives the dimension of the frame: `rect=[xmin,ymin,xmax,ymax]`.

Description

polarplot creates a polar coordinate plot of the angle theta versus the radius rho. theta is the angle from the x-axis to the radius vector specified in radians; rho is the length of the radius vector specified in dataspace units.

Examples

```
t= 0:.01:2*%pi;
clf();polarplot(sin(7*t),cos(8*t))

clf();polarplot([sin(7*t') sin(6*t')],[cos(8*t') cos(8*t')],[1,2])
```

Name

polyline_properties — description of the Polyline entity properties

Description

The Polyline entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for polylines.

parent:
: This field contains the handle of the parent. The parent of the polyline entity should be of the type `"Axes"` or `"Compound"`.

children:
: This property contains a vector with the `children` of the handle. However, polyline handles currently do not have any `children`.

visible:
: This field contains the `visible` property value for the entity . It should be `"on"` or `"off"` . By default, the polyline is visible, the value's property is `"on"` . If `"off"` the polyline is not drawn on the screen.

data:
: This field contains the values for the x and y coordinates. Component Z is to be added in the case of three-dimensional axes. It is a two (three) column matrix `[x,y,[z]]` of points.

closed:
: This field determines wether the polyline is closed or not: its value can be `"on"` or `"off"` (no default value, it depends on the primitive used to create the polyline).

line_mode:
: This field contains the default `line_mode` property value for the polyline. Its value should be `"on"` (line drawn) or `"off"` (no line drawn).

fill_mode:
: If the `polyline_style` field is different of 5, fill the background of the curve with color defined by the `background` property.

line_style:
: The `line_style` property value should be an integer in [1 6]. 1 stands for solid the other value stands for a selection of dashes (see getlinestyle).

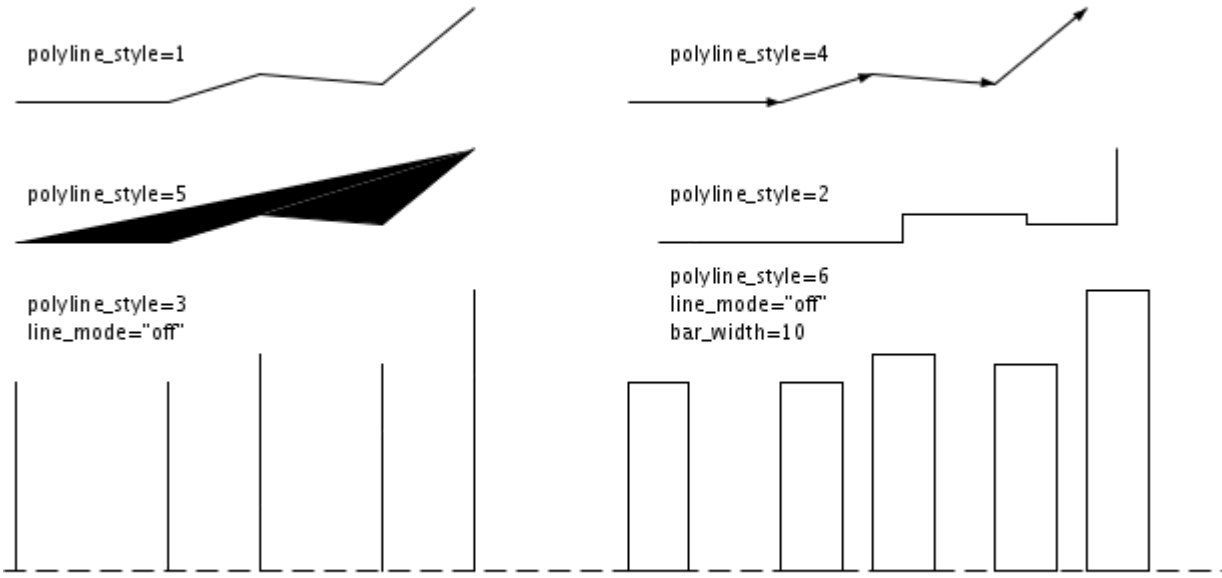
thickness:
: This field contains the line `thickness` property. It's value should be positive integer.

arrow_size_factor:
: This integer allows to set the size of arrows drawn on the polyline. The actual size of arrows is the product of the `thickness` and the the size factor.

polyline_style:
: This property sets several polyline drawing mode:

  - If the value is 0 or 1 lines are drawn between two consecutives points.
  - If the value is 2 the polyline produces a staircase plot. Two consecutives points are linked by an horizontal line followed by a vertical line.
  - If the value is 3 the polyline produces a bar plot. For each given point (x,y) a vertical line is drawn from (x,y) to (x,0).

- If the value is 4 arrows are drawn between two consecutives points.
- If the value is 5 the polyline is filled (patch).
- If the value is 6 the polyline is a Matlab-like bar object. The properties `bar_shift` and `bar_width` command its appearance.



foreground:
This field contains the default `foreground` property used to draw the polyline. Its value should be a color index (relative to the current colormap).

background:
This field contains the color used to fill the background of the polyline. Its value should be a color index (relative to the current colormap).

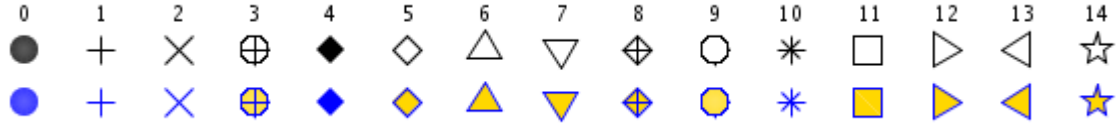
interp_color_vector:
This field contains the vector of color indices used to fill in the polyline when the `interp_color_mode` property is set to `"on"`. It defines the intervals of colormap indices used to fill each segment. For instance, the first segment will be filled by every colors whose index is between the first two elements of the vecor. It is only applicable if the polyline is defined by 3 or 4 points. Therefore, the size of the vector must match this dimension.

interp_color_mode:
This field determines if we are using the interpolated shading mode to fill the polyline : its value can be `"on"` or `"off"`. Note that an `interp_color_vector` must be defined before switching to "on" value (see above).

mark_mode:
This field contains the default `mark_mode` property value for the polyline. Its value should be `"on"` (marks drawn) or `"off"` (no marks drawn).

mark_style:
The `mark_style` property value is used to select the type of mark to use when `mark_mode` property is `"on"`. The value should be an integer in [0 14] which stands for: dot, plus, cross, star, filled diamond, diamond, triangle up, triangle down, diamond plus, circle, asterisk,

square, triangle right, triangle left and pentagram. The figure below shows the aspects of the marks depending on the `mark_style` and the `mark_foreground` and `mark_background` properties.



mark_size_unit:
This field contains the default `mark_size_unit` property value. If `mark_size_unit` is set to `"point"`, then the `mark_size` value is directly given in points. When `mark_size_unit` is set to `"tabulated"`, `mark_size` is computed relative to the font size array: therefore, its value should be an integer in [0 5] which stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt. Note that plot2d and pure scilab functions use `tabulated` mode as default ; when using plot function, the `point` mode is automatically enabled.

mark_size:
The `mark_size` property is used to select the type of size of the marks when `mark_mode` property is `"on"`. Its value should be an integer between 0 and 5 whith stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt.

mark_foreground:
This field contains the `mark_foreground` property value which is the marks' edge color. Its value should be a color index (relative to the current color_map).

mark_background:
This field contains the `mark_background` property value which is the marks' face color. Its value should be a color index (relative to the current color_map).

x_shift:
This field contains the offset computed by a call to the bar function (or re-computed by a call to barhomogenize) and is used to perform a nice vertical bar representation. Note that this offset is also taken into account for all the other `polyline_style`. The unit is expressed in user coordinates.

y_shift:
This field contains the offset computed by a call to the bar function (or re-computed by a call to barhomogenize) and is used to perform a nice horizontal bar representation. Note that this offset is also taken into account for all the other `polyline_style`. The unit is expressed in user coordinates.

z_shift:
This field contains the offset the user may specify. Note that this offset is taken into account for all the `polyline_style`. The unit is expressed in user coordinates.

bar_width:
This field determines the width of the selected polyline when its `polyline_style` is set to bar mode (case 6) : the unit is expressed in user coordinates.

clip_state:
This field contains the `clip_state` property value for the polyline. It should be :

- `"off"` this means that the polyline is not clipped.
- `"clipgrf"` this means that the polyline is clipped outside the Axes box.
- `"on"` this means that the polyline is clipped outside the rectangle given by property clip_box.

clip_box:
This field is to determinate the `clip_box` property. By Default its value should be an empty matrix if clip_state is "off". Other cases the vector `[x,y,w,h]` (upper-left point width height) defines the portions of the polyline to display, however `clip_state` property value will be changed.

user_data:
This field can be use to store any scilab variable in the polyline data structure, and to retrieve it.

Examples

```
a=get("current_axes")//get the handle of the newly created axes
a.data_bounds=[-2,-2;2,2];

xpoly(sin(2*%pi*(0:5)/5),cos(2*%pi*(0:5)/5),"lines",0)
p=get("hdl"); //get handle on current entity (here the polyline entity)
p.foreground=2;
p.thickness=3;
p.mark_style=9;
d=p.data;d(1,:)=[0 0];p.data=d;
a.rotation_angles=[0 45];

p.data=[(-2:0.1:2)' sin((-2:0.1:2)*%pi)']
p.mark_mode="off";
p.polyline_style=3;
p.line_style=4;
```

See Also

set , get , delete , xpoly , xfpoly , xpolys , xfpolys , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

rainbowcolormap — red through orange, yellow, green,blue to violet colormap

```
cmap=rainbowcolormap(n)
```

Parameters

n
: integer >= 3, the colormap size.

cmap
: matrix with 3 columns `[R,G,B]`.

Description

`rainbowcolormap` computes a colormap with `n` colors varying from red through orange, yellow, green,blue to violet.

Examples

```
f = scf();
plot3d1();
f.color_map = rainbowcolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , springcolormap , summercolormap , whitecolormap , wintercolormap

Name

rectangle_properties — description of the Rectangle entity properties

Description

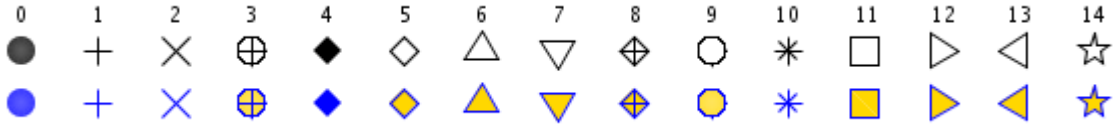
The Rectangle entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for rectangles and filled rectangles.

parent:
: This field contains the handle of the parent. The parent of the rectancle entity should be of the type `"Axes"` or `"Compound"`.

children:
: This property contains a vector with the `children` of the handle. However, `rectangle` handles currently do not have any `children`.

mark_mode:
: This field contains the default `mark_mode` property value for the polyline. Its value should be `"on"` (marks drawn) or `"off"` (no marks drawn).

mark_style:
: The `mark_style` property value is used to select the type of mark to use when `mark_mode` property is `"on"`. The value should be an integer in [0 14] which stands for: dot, plus, cross, star, filled diamond, diamond, triangle up, triangle down, diamond plus, circle, asterisk, square, triangle right, triangle left and pentagram. The figure below shows the aspects of the marks depending on the `mark_style` and the `mark_foreground` and `mark_background` properties.



mark_size_unit:
: This field contains the default `mark_size_unit` property value. If `mark_size_unit` is set to `"point"`, then the `mark_size` value is directly given in points. When `mark_size_unit` is set to `"tabulated"`, `mark_size` is computed relative to the font size array: therefore, its value should be an integer in [0 5] whith stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt. Note that xrect and pure scilab functions use `tabulated` mode as default ; when using plot function, the `point` mode is automatically enabled.

mark_size:
: The `mark_size` property is used to select the type of size of the marks when `mark_mode` property is `"on"`. Its value should be an integer in [0 5] whith stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt.

mark_foreground:
: This field contains the `mark_foreground` property value which is the marks' edge color. Its value should be a color index (relative to the current color_map).

mark_background:
: This field contains the `mark_background` property value which is the marks' face color. Its value should be a color index (relative to the current color_map).

line_mode:
: This field contains the default `line_mode` property value for the rectangle. Its value should be `"on"` (line drawn) or `"off"` (no line drawn).

fill_mode:
If `fill_mode` property value is `"on"` , the rectangle is filled with the foreground color, the `mark_mode` must also have the value `"off"`. if not and the value's property is `"off"` only the shape of the rectangle is drawn using the foreground color.

line_style:
The `line_style` property value should be an integer in [1 6]. 1 stands for solid the other value stands for a selection of dashes.

thickness:
This field contains the line `thickness` property. Its value should be a positif integer.

foreground:
This field contains the color used to draw the outline of the rectangle. Its value should be a color index (relative to the current colormap).

background:
This field contains the color used to fill the rectangle. Its value should be a color index (relative to the current colormap).

data:
This property is to return the coordinates of the upper-left point of the rectangle and its width and height in user coordinates. The result is the matrix `[xleft,yup,[zup],width,height]`

visible:
This field contains the `visible` property value for the entity . It should be `"on"` or `"off"` . By default, the rectangle is visible, the value's property is `"on"`. If `"off"` the rectangle is not drawn on the screen.

clip_state:
This field contains the `clip_state` property value for the rectangle. It should be :

- `"off"` this means that the rectangle is not clipped.
- `"clipgrf"` this means that the rectangle is clipped outside the Axes box.
- `"on"` this means that the rectangle is clipped outside the rectangle given by property clip_box.

clip_box:
This field is to determinate the `clip_box` property. By Default its value should be an empty matrix if clip_state is "off". Other cases the vector `[x,y,w,h]` (upper-left point width height) defines the portions of the rectangle to display, however `clip_state` property value will be changed.

user_data:
This field can be use to store any scilab variable in the rectangle data structure, and to retreive it.

Examples

```
a=get("current_axes");//get the handle of the newly created axes
a.data_bounds=[-2,-2;2,2];

xrect(-1,1,2,2)

r=get("hdl");//get handle on current entity (here the rectangle entity)
r.type
r.parent.type
r.foreground=13;
```

```
r.line_style=2;
r.fill_mode="on";
r.background=color('red');
r.clip_box=[-1 1;1 1];
r.data(:,[3 4])=[1/2 1/2];
r.data(:,[1 2])=[1/2 1/2];
r.clip_state="off"
```

See Also

set , get , delete , xrect , xfrect , xrects , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

relocate_handle — Move handles inside the graphic hierarchy.

```
relocate_handle( movedHandles, parent )
```

Parameters

movedHandles
: Vector of relocated handles.

parent
: New parent of the handles.

Description

The relocate_handle function allows to move handles from their current locations in the graphical hierarchy to another. All the moved entities are relocated under the same parent handle specified with the **parent** parameter.

Since not every handles are compatible with each others, some restrictions exist when relocationg handles. For examples, it is not allowed to move an axes handle under a polyline. More information about compatibility can be found in the graphics_entities page.

This routine is particularly useful to move an object from an axes entity to an other or to move axes from figures handles.

Examples

```
x = 0:10 ;

// plot a first polyline
plot(x,x^2) ;
axes1 = gca() ;
poly1 = gce() ;

// plot a second in an other window
scf() ;
plot( x,x )  ;
axes2 = gca() ;
poly2 = gce()  ;
poly2bis = copy( poly2 ) ; // make a copy of the polyline

// put both polyline in the same window
relocate_handle( poly2bis, axes1 ) ;
```

See Also

graphics_entities , copy , delete , swap_handles

Authors

Jean-Baptiste Silvy

Name

replot — redraw the current graphics window with new boundaries

```
replot(rect,[handle])
```

Parameters

rect
: row vector of size 4.

handle
: optional argument. Graphics handle(s) of type Axes to select one or several given Axes. Only available in new graphics mode.

Description

`replot` is used to redraw the content of the current graphics window with new boundaries defined by `rect=[xmin,ymin,xmax,ymax]`. Under old graphics syntax, it works only with the driver `"Rec"`.

Under new graphics mode, this transformation can be applied to specific axes given by Axes graphics handles via the handle argument. If handle is not specified, the new boundaries are applied to the current axes of the current figure. The transformation changes the `data_bounds` value of those axes. Note that the axes property `tight_limits` must also be set to "on" to strictly select those bounds (see axes_properties).

Examples

```
backupstyle='?'

// First Example
x=[0:0.1:2*%pi]';
plot2d(x,sin(x))
replot([-1,-1,10,2])

// Second Example
xdel(winsid());
plot() // plot demo
f=gcf();
replot([-1,-1,10,2],f.children(1)) // specify axes handle's value
replot([-3,-2,8,4],f.children(2))
```

See Also

xbasr , xbasc , clf

Authors

J.Ph.C.

Name

rgb2name — returns the name of a color

```
names=rgb2name(r,g,b)
names=rgb2name(rgb)
```

Parameters

r,g,b
: RGB integer values of a color.

rgb
: vector of RGB integer values of a color.

names
: names of the color.

Description

`rgb2name` returns the color name corresponding to the RGB values given by its argument. A vector of color names can also be returned if the color has more than 1 name. `r`, `g` and `b` must be integers between 0 and 255 corresponding to colors components red, green and blue. As usual 0 means no intensity and 255 means all the intensity of the color. RGB values can also be given by a vector `[r,g,b]`.

If no color is found `[]` is returned.

The list of all known colors is given by color_list.

Examples

```
rgb2name(255,128,128)
rgb2name([255 215 0])
// get color #10 of the current colormap and find its name
cmap=get(gcf(),"color_map");
rgb2name(cmap(10,:)*255)
```

See Also

color , color_list , name2rgb

Name

rotate — rotation of a set of points

```
xy1=rotate(xy,[theta,orig])
```

Parameters

xy
: matrice of size (2,.).

xy1
: matrice of size (2,.).

theta
: real, angle en radian; default value is 0.

orig
: center of the rotation; default value is [0;0].

Description

`rotate` performs a rotation with angle `theta`:

```
xy1(:,i) = M(theta) *xy (:,i) + orig
```

where `M` stands for the corresponding rotation matrix.

Examples

```
xsetech([0,0,1,1],[-1,-1,1,1])
xy=[(0:0.1:10);sin(0:0.1:10)]/10;
for i=2*%pi*(0:10)/10,
  [xy1]=rotate(xy,i);
  xpoly(xy1(1,:),xy1(2,:),"lines")
end
```

Name

rotate_axes — Interactive rotation of an Axes handle.

```
rotate_axes()
rotate_axes(h)
```

Parameters

h
: Figure or Axes handle. Specify on which Axes the rotation will apply.

Description

`rotate_axes` funtion is used to perform an interactive rotation of an Axes object. When the function is called, the user is requested to click twice on the graphic window. The first click initializes the rotation and the second ends it.

If an Axes handle is specified as input argument, the rotation will apply on it. If a figure handle is specified, the first click determines the Axes objet to rotate. If the function is called with no argument, the rotation apply on the current figure.

Examples

```
clf();
// create two axes in the figure
subplot(2, 1, 1);
plot2d;
subplot(2, 1, 2);
plot3d;

// rotate only the second axes
axes2 = gca();
rotate_axes(axes2);

// rotate the selected one
rotate_axes();
// or
rotate_axes(gcf());
```

See Also

zoom_rect , axes_properties

Authors

Jean-Baptiste Silvy INRIA

Name

rubberbox — Rubberband box for rectangle selection

```
[final_rect,btn]=rubberbox()
[final_rect,btn]=rubberbox(initial_rect)
[final_rect,btn]=rubberbox(edition_mode)
[final_rect,btn]=rubberbox(initial_rect, edition_mode)
```

Parameters

initial_rect
: vector with two or four entries. With four entries it gives the initial rectangle defined by [x_left, y_top, width, height], with two entries width and height are supposed to be 0.

edition_mode
: a boolean, if edition_mode is `%t` button press selects the first corner, release selects the opposite corner. If edition_mode is `%f`, a button press or click selects the first corner, a click is requested to select the opposite corner. The default value is %f.

final_rect
: a rectangle defined by [x_left, y_top, width, height]

btn
: an integer, the number of the mouse button clicked

Description

`rubberbox(initial_rect)` tracks a rubberband box in the current graphic window, following the mouse. When a button is clicked `rubberbox` returns the final rectangles definition in `final_Rect`. If the argument `initial_rect` is not specified, a click is needed to fix the initial corner position.

Examples

```
xsetech(frect=[0,0,100,100])
[x,y]=xclick();r=rubberbox([x;y;30;10])
xrect(r)
r=rubberbox()
```

See Also

xrect , xrects , xclick , xgetmouse , dragrect

Name

sca — set the current axes entity

```
a=sca(a)
```

Parameters

a
: a handle, the handle on the Axes entity

Description

`sca(a)` is used to set the current `Axes` entity (see graphics_entities) to the one pointed to by the handle `a`. The drawing functions generaly use the current axes entity.

Examples

```
clf()
a1=newaxes();
a1.axes_bounds=[0,0,1.0,0.5];
t=0:0.1:20;
plot(t,acosh(t),'r')
a2=newaxes();
a2.axes_bounds=[0,0.5,1.0,0.5];
x=0:0.1:4;
plot(x,sinh(x))


sca(a1); //make first axes current
plot(t,asinh(t),'g')
legend(['acosh','asinh'])
sca(a2); //make secoçnd axes current
legend('sinh')
```

See Also

subplot , gda , newaxes

Authors

S. Steer, INRIA

Name

scaling — affine transformation of a set of points

```
xy1=scaling(xy,factor,[orig])
```

Parameters

xy1
: matrice of size (2,.).

xy
: matrice of size (2,.).

factor
: real scalar, coefficient of the linear transformation.

orig
: shift vector; default value is [0;0].

Description

`scaling` performs an affine transformation on the set of points defined by the coordinates `xy`:

`xy1(:,i) = factor * xy(:,i) + orig`.

Name

scf — set the current graphic figure (window)

```
f = scf()
f = scf(h)
f = scf(num)
```

Parameters

h
: a handle, the figure handle

num
: a number, the figure_id

f
: the handle of the current figure

Description

The current figure is the destination of the graphic drawing. The `scf` function allows to change this current figure or to create it if it does not already exist.

`scf(num)` set the figure with `figure_id==num` as the current figure. If it does not already exist it is created.

`scf(h)` set the figure pointed to by the handle `h` as the current figure. If it does not already exist it is created.

`scf()` is equivalent to `scf(max(winsid())+1)` . It may be used to create a new graphic window.

Examples

```
f4=scf(4); //creates figure with id==4 and make it the current one
f0=scf(0); //creates figure with id==0 and make it the current one
plot2d() //draw in current figure (id=0)
scf(f4); // set first created figure as current one
plot3d() //draw in current figure (id=4)
```

See Also

set , get , gcf , clf , get_figure_handle , graphics_entities

Authors

S. Steer INRIA

Name

sd2sci — gr_menu structure to scilab instruction convertor

```
txt=sd2sci(sd [,sz [,orig]])
```

Parameters

sd
: data structure build by gr_menu.

sz
: vector of number or strings with two components, give the x and y zoom factors

orig
: vector of number or strings with two components, give the origin translation vector

Description

given a `sd` data structure generated by `gr_menu` `sd2sci` forms a vector of scilab instructions corresponding to the graphic edited by `gr_menu`.

The optional parameters `sz` and `orig` allows to zoom and shift the initial graphic.

If `sz` or `orig` are given by strings generated instructions are relative use then as formal expressions.

See Also

execstr

Authors

Serge Steer INRIA 1988

Name

sda — Set default axes.

```
sda()
a = gda(); set(a,"default_values",1)
```

Parameters

a
: handle, the handle of the default axes.

Description

This routine resets the axes' model to default values.

Examples

```
x=[0:0.1:2*%pi]';
f=get("default_figure"); // get the handle of the model figure
a=get("default_axes"); // get the handle of the model axes
  // setting its' properties
f.figure_size=[1200 900];
f.figure_position=[0 0];
a.background=4;
a.box="off";
a. tics_color=5;
a.labels_font_color=25;
a.labels_font_size=4;
a.sub_tics=[7 3];
a.x_location="middle";
a.y_location="middle";
a.tight_limits="on";
a.thickness=2;
a.grid=[-1 24];
subplot(221);
plot2d(x-2,sin(x))
subplot(222);
plot2d(x-6,[2*cos(x)+.7 2*cos(x)+.9 cos(2*x) .2+sin(3*x)],[-1,-2,-3 -4])
sda() // return to the  default values of the axes' model
subplot(223);
plot2d(x-2,sin(x))
subplot(224);
plot2d(x-6,[2*cos(x)+.7 2*cos(x)+.9 cos(2*x) .2+sin(3*x)],[-1,-2,-3 -4])
xdel(0)
plot2d(x-2,sin(x))
```

See Also

sdf , gda , gdf , set , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

sdf — Set default figure.

```
sdf()
f = gdf(); set(f,"default_values",1)
```

Parameters

f
: handle, the handle of the default figure.

Description

This routine resets the figure's model to default values.

Examples

```
x=[0:0.1:2*%pi]';
f=get("default_figure"); // get the handle of the model figure
a=get("default_axes"); // get the handle of the model axes
  // setting its' properties
f.background=4;
f.auto_resize="off";
f.figure_size=[400 300];
f.axes_size=[600 400];
f.figure_position=[0 -1];
a.x_location="top";
a.y_location="left";
for (i=1:6)
  xset("window",i) // create a figure with the identifier i
  plot2d(x,[sin(x) cos(x)],[i -i])
  xclick();
  if i == 4, sdf(); end // return to the  default values of the figure's mod
 end
```

See Also

sda , gdf , gda , set , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

secto3d — 3D surfaces conversion

```
[m[,x]]=secto3d(seclist,npas)
[m]=secto3d(seclist ,x)
```

Parameters

seclist
: a list whose elements are (2,.) matrices

npas
: an integer

m
: a matrix

x
: a vector

Description

Considering a surface given through a list `seclist` of sections in the `(x,z)` plane `[m [,x]]=secto3d(seclist [,npas])` returns a matrix `m` which contains a regular discretization of the surface.

- The i-th row of the matrix m corresponds to the i-th section
- The j-th column of m corresponds to the x(j)

Each section `seclist(i)` is described by a (2,.) matrix which gives respectively the x and z coordinates of points.

`[m]=secto3d(seclist ,x)`in that case the x-vector gives the discretization of the x-axis for all the sections

See Also

plot3d

Authors

Steer S.; ;

Name

segs_properties — description of the Segments entity properties

Description

The Segs entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for a set of colored segments or colored arrows.

parent:
: This property contains the handle of the parent. The parent of the segment entity should be of the type `"Axes"` or `"Compound"`.

children:
: This property contains a vector with the `children` of the handle. However, `segs` handles currently do not have any `children`.

visible:
: This field contains the `visible` property value for the entity . It should be `"on"` or `"off"` . By default, the segments are visibles, the value's property is `"on"`. If `"off"` the segments are not drawn on the screen.

data:
: This field is two column matrix `[x,y,[z]]` which gives the coordinates of the segments boundary. If `xv=matrix(x,2,-1)` and `yv=matrix(y,2,-1)` then `xv(:,k)` and `yv(:,k)` are the boundary coordinates of the segment numbered `k`.

line_mode:
: This field contains the default `line_mode` property value for the segs. Its value should be `"on"` (line drawn) or `"off"` (no line drawn).

line_style:
: The `line_style` property value should be an integer in [0 6]. 0 stands for solid the other value stands for a selection of dashes. This property applies to all segments.

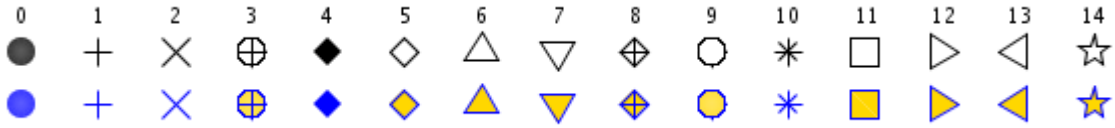
thickness:
: This field contains the `thickness` property for all segments. Its value should be a non negative integer..

arrow_size:
: Factor that specify the size of a arrowheads. With a negative value, the size is also dependent of the arrows length. TO draw segment, the value must be set to 0.

segs_color:
: This field contains the vector of colors to use to draw each segment. Each element is a color index relative to the current colormap.

mark_mode:
: This field contains the default `mark_mode` property value for the polyline. Its value should be `"on"` (marks drawn) or `"off"` (no marks drawn).

mark_style:
: The `mark_style` property value is used to select the type of mark to use when `mark_mode` property is `"on"`. The value should be an integer in [0 14] which stands for: dot, plus, cross, star, filled diamond, diamond, triangle up, triangle down, diamond plus, circle, asterisk, square, triangle right, triangle left and pentagram. The figure below shows the aspects of the marks depending on the `mark_style` and the `mark_foreground` and `mark_background` properties.



mark_size_unit:
This field contains the default `mark_size_unit` property value. If `mark_size_unit` is set to `"point"`, then the `mark_size` value is directly given in points. When `mark_size_unit` is set to `"tabulated"`, `mark_size` is computed relative to the font size array: therefore, its value should be an integer in [0 5] which stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt. Note that plot2d and pure scilab functions use `tabulated` mode as default ; when using plot function, the `point` mode is automatically enabled.

mark_size:
The `mark_size` property is used to select the type of size of the marks when `mark_mode` property is `"on"`. Its value should be an integer between 0 and 5 whith stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt.

mark_foreground:
This field contains the `mark_foreground` property value which is the marks' edge color. Its value should be a color index (relative to the current color_map).

mark_background:
This field contains the `mark_background` property value which is the marks' face color. Its value should be a color index (relative to the current color_map).

clip_state:
This field contains the `clip_state` property value for the segments. It should be :

- `"off"` this means that the segments is not clipped.
- `"clipgrf"` this means that the segments is clipped outside the Axes box.
- `"on"` this means that the segments is clipped outside the rectangle given by the property `clip_box`.

clip_box:
This field contains the `clip_box` property. By default segment are not clipped, clip_state is "off", so the value should be an empty matrix .Other cases the vector `[x,y,w,h]` (upper-left point width height) defines the portions of the segments to display, however `clip_state` property value will be changed.

user_data:
This field can be use to store any scilab variable in the segs data structure, and to retreive it.

Examples

```
a=get("current_axes");//get the handle of the newly created axes
a.data_bounds=[-10,-10;10,10];
x=2*%pi*(0:7)/8;
xv=[2*sin(x);9*sin(x)];
yv=[2*cos(x);9*cos(x)];
xsegs(xv,yv,1:8)

s=a.children
s.arrow_size=1;
```

```
 s.segs_color=15:22;
 for j=1:2
   for i=1:8
     h=s.data(i*2,j);
     s.data(i*2,j)=s.data(i*2-1,j);
     s.data(i*2-1,j)=  h;
    end
 end

s.segs_color=5; //set all the colors to 5

s.clip_box=[-4,4,8,8];
a.thickness=4;
xrect(s.clip_box);
```

See Also

set , get , delete , xsegs , graphics_entities

Authors

Djalel ABDMOUCHE

Name

set — set a property value of a graphic entity object or of a User Interface object.

```
set(prop,val)
set(h,prop)
set(h,prop,val)
h.prop=val
```

Parameters

h
: graphic handle of the entity which to set the named property. `h` can be a vector of handles, in which case set modifies the property for all entities contained in h.

prop
: character string, name of the property to set.

val
: value to give to the property.

Description

This routine can be used to modify the value of a specified property from a graphics entity or a GUI object. In this case it is equivalent to use the dot operator on an handle. For exemple, `set(h,"background",5)` is equivalent to `h.background = 5`.

Property names are character strings. The type of the set values depends on the handle type and property.

To get the list of all existing properties see graphics_entities or uicontrol for User Interface objects.

`set` function can be also called with only a property and a value as argument. In this case, the property must be one of the following:

current_entity or hdl
: `set('current_entity',h)` or `set('hdl',h)` sets a new entity as current. In this case, the value must be a graphic handle.

current_figure
: `set('current_figure',fig)` sets a new graphic figure as current. It is equivalent to scf. In this case, the value must be a Figure handle.

current_axes
: `set('current_axes',axes)` sets a new axes entity as current. It is equivalent to sca. In this case, the value must be an Axes handle.

`set` can also be called with a graphic handle and property as arguments. The handle must be either a handle on the default figure or the default axes entities. The property must be `"default_values"`. In this case, the default entity is reset to the value it had at Scilab startup. `set("default_values",h)` is equivalent to sda or sdf.

Examples

```
clf()
set("auto_clear","off") ;
// Exemple of a Plot 2D
```

```
x=[-.2:0.1:2*%pi]';
plot2d(x-.3,[sin(x-1) cos(2*x)],[1 2] );
a=get("current_axes");
p1=a.children.children(1);
p2=a.children.children(2);
// set the named properties to the specified values on the objects
set(p2,"foreground",13);
set(p2,"polyline_style",2);
set(a,'tight_limits',"on");
set(a,"box","off");
set(a,"sub_tics",[ 7 0 ]);
set(a,"y_location","middle")
set(p2,'thickness',2);
set(p1,'mark_mode',"on");
set(p1,'mark_style',3);
plot2d(x-2,x.^2/20);
p3= a.children(1).children;
set([a p1 p2 p3],"foreground",5)
```

See Also

get , delete , copy , move , graphics_entities , uicontrol

Authors

Djalel ABDEMOUCHE

Name

set_posfig_dim — change defaut transformation for exporting in postscript

```
set_posfig_dim(w,h)
set_posfig_dim(0,0)
```

Parameters

w
: graphic figure width, number of pixels

h
: graphic figure height, number of pixels

Description

`set_posfig_dim(w,h)` set the coordinates transformation to be used when exporting a graphic window in postscript. The graphic figure zoomed with ratios `w/<graphic figure width>` horizontally and `h/<graphic figure height>` vertically.

This function is particularily useful when one wants an export a graphic figure exactly as it is shown on the screen. For that `w` and `h` has to be set respectively to `<graphic figure width>` and `<graphic figure height>`

`set_posfig_dim(0,0)` resets the default values: `w=600`, `h=424`.

Function `set_posfig_dim` is obsolete and will be permanently removed in Scilab 5.2. In current version, set_posfig_dim has no influence on exported files. Their width and height instead match the exported figure `axes_size` property.

Examples

```
// make a figure with a specific shape
f=scf(0);f.figure_size=[800,300];
plot2d();
set_posfig_dim(f.figure_size(1),f.figure_size(2));
filename='foo.ps';
xs2ps(0,filename);
```

See Also

xs2ps

Authors

Serge Steer INRIA

Name

seteventhandler — set an event handler for the current graphic window

```
seteventhandler(sfun_name)
seteventhandler('')
```

Parameters

sfun_name
: a character string. The name of the Scilab function which is intended to handle the events

Description

The function allows the user to set a particular event handler for the current graphic window. `seteventhandler('')` removes the handler.

For more information about event handler functions see the event handler functions help.

Examples

```
function my_eventhandler(win,x,y,ibut)
   if ibut==-1000 then return,end
   [x,y]=xchange(x,y,'i2f')
    xinfo(msprintf('Event code %d at mouse position is (%f,%f)',ibut,x,y))
endfunction
plot2d()
seteventhandler('my_eventhandler')
//now:
//   -  move the mouse over the graphic window
//   -  press and release keys shifted or not with Ctrl pressed or not
//   -  press button, wait a little release
//   -  press and release button
//   -  double-click button


seteventhandler('') //suppress the event handler
```

See Also

addmenu , xgetmouse , xclick , xchange , event handler functions , figure_properties

Name

show_pixmap — send the pixmap buffer to the screen

```
show_pixmap()
```

Description

If a graphic window `pixmap` property is `"on"` the drawings are send to a pixmap memory instead of the screen display.

The `show_pixmap()` instruction send the pixmap to the screen.

The pixmap mode can be used to obtain smooth animations. This property can be found among the figure entity fields (see figure_properties).

Examples

```
f=gcf();f.pixmap='on'; //set the pixmap mode
a=gca();a.data_bounds=[0 0; 10 10];
//construct two rectangles
xrects([0;10;1;1],5);r1=gce();r1=r1.children;
xrects([0;1;1;1],13);r2=gce();r2=r2.children;
//animation loop
for k=1:1000
   //draw the rectangles in the pixmap buffer
   move(r1,[0.01,-0.01]);move(r2,[0.01,0.01])
   //show the pixmap buffer
   show_pixmap()
end
```

See Also

figure_properties , clear_pixmap

Authors

Serge Steer INRIA

Name

show_window — raises a graphics window

```
show_window([figure])
```

Parameters

figure
: number or handle of the figure to show.

Description

With no parameters,`show_window`raises the current graphics window even if it is iconified. Otherwise, raises the specified window by its number or handle. If no window already exists, one is created.

Authors

J.Ph.C.

Name

springcolormap — magenta to yellow colormap

```
cmap=springcolormap(n)
```

Parameters

n
: integer >= 3, the colormap size.

cmap
: matrix with 3 columns `[R,G,B]`.

Description

`springcolormap` computes a colormap with `n` colors varying from magenta to yellow.

Examples

```
f = scf();
plot3d1();
f.color_map = springcolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , summercolormap , whitecolormap , wintercolormap

Name

square — set scales for isometric plot (change the size of the window)

```
square(xmin,ymin,xmax,ymax)
```

Parameters

xmin,xmax,ymin,ymax
: four real values

Description

`square` is used to have isometric scales on the x and y axes. The requested values `xmin, xmax, ymin, ymax` are the boundaries of the graphics frame and `square` changes the graphics window dimensions in order to have an isometric plot. `square` set the current graphics scales and can be used in conjunction with graphics routines which request the current graphics scale (for instance `fstrf="x0z"` in `plot2d`).

Examples

```
t=[0:0.1:2*%pi]';
plot2d(sin(t),cos(t))
xbasc()
square(-1,-1,1,1)
plot2d(sin(t),cos(t))
xset("default")
```

See Also

isoview , xsetech

Authors

Steer S.

Name

stringbox — Compute the bounding rectangle of a text or a label.

```
rect = stringbox( string, x, y, [angle, [fontStyle, [fontSize]]] )
rect = stringbox( Handle )
```

Parameters

rect
: a 2x4 matrix containing the 4 vertex coordinates of the bounding rectangle.

string
: string matrix to be enclosed.

x,y
: real scalars, coordinates of the lower left point of strings.

angle
: Rotation angle of the string clockwise and in degrees around the `(x,y)` point.

fonStyle
: integer specifying the type of the font.

fontSize
: integer specifying the size of the font.

Handle
: a graphic handle of Text or Label type.

Description

`stringbox` returns the bounding rectangle vertices of a text or label object or a string which will be displayed with a certain way. the coordinates are given with the current graphic scale. the first vertex correspond to the text coordinates `(x,y)`, the lower left point without rotation, the following vertex are given clockwise in the resulting matrix.

The result migth not be very accurate with PostScript driver.

Examples

```
// show axes
axes = gca() ;
axes.axes_visible = 'on' ;
axes.data_bounds = [ 1, 1 ; 10, 10 ] ;

// display a labels for axes
xtitle( 'stringbox', 'X', 'Y' )  ;

// get the bounding box of X label
stringbox( axes.x_label )

// draw a string
str = [ "Scilab", "is" , "not", "Skylab" ] ;
xstring( 4, 9, str ) ;

//modify the text
e = gce() ;
```

```
e.font_angle = 90 ;
e.font_size    = 6  ;
e.font_style   = 7 ;
e.box = 'on' ;

// get its bounding box
stringbox( e )
// or
rect = stringbox( str, 4, 9, 90, 7, 6 )

// click and find if the text was hit
hit = xclick()  ;
hit = hit( 2 : 3 ) ;

if hit(1) >= rect(1,1)  & hit(1) <=  rect(1,2) & hit(2) <= rect(2,2) & hit(2) >=
  disp('You hit the text.') ;
else
  disp('You missed it.')
end;
```

See Also

xstring , xstringl , xstringb

Authors

Jean-Baptiste Silvy

Name

subplot — divide a graphics window into a matrix of sub-windows

```
subplot(m,n,p)
subplot(mnp)
```

Parameters

m,n,p
: positive integers

mnp
: an integer with decimal notation `mnp`

Description

subplot(m,n,p) or subplot(mnp) breaks the graphics window into an m-by-n matrix of sub-windows and selects the p-th sub-window for drawing the current plot. The number of a sub-window into the matrices is counted row by row ie the sub-window corresponding to element (i,j) of the matrix has number (i-1)*n + j.

Examples

```
subplot(221)
plot2d()
subplot(222)
plot3d()
subplot(2,2,3)
param3d()
subplot(2,2,4)
hist3d()
```

See Also

plot2d , plot3d , xstring , xtitle

Name

summercolormap — green to yellow colormap

```
cmap=summercolormap(n)
```

Parameters

n
: integer >= 3, the colormap size.

cmap
: matrix with 3 columns `[R,G,B]`.

Description

`summercolormap` computes a colormap with `n` colors varying from green to yellow.

Examples

```
f = scf();
plot3d1();
f.color_map = summercolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , whitecolormap , wintercolormap

Name

surf — 3D surface plot

```
surf(Z,<GlobalProperty>)
surf(Z,color,<GlobalProperty>)
surf(X,Y,Z,<color>,<GlobalProperty>)
surf(<axes_handle>,...)
```

Parameters

Z
: a real matrix defining the surface height. It can not be omitted. The Z data is a mxn matrix.

X,Y
: two real matrices or vectors: always set together, these data defines a new standard grid. This new X and Y components of the grid must match Z dimensions (see description below).

color
: an optional real matrix defining a color value for each (X(j),Y(i)) point of the grid (see description below).

<GlobalProperty>
: This optional argument represents a sequence of couple statements {PropertyName,PropertyValue} that defines global objects' properties applied to all the curves created by this plot. For a complete view of the available properties (see GlobalProperty).

<axes_handle>
: This optional argument forces the plot to appear inside the selected axes given by axes_handle rather than the current axes (see gca).

Description

surf draws a colored parametric surface using a rectangular grid defined by X and Y coordinates (if {X,Y} are not specified, this grid is determined using the dimensions of the Z matrix) ; at each point of this grid, a Z coordinate is given using the Z matrix (only obligatory data). surf has been created to better handle Matlab syntax. To improve graphical compatibility, Matlab users should use surf (rather than plot3d).

Data entry specification :

In this paragraph and to be more clear, we won't mention GlobalProperty optional arguments as they do not interfer with entry data (except for "Xdata", "Ydata" and "Zdata" property, see GlobalProperty). It is assumed that all those optional arguments could be present too.

If Z is the only matrix specified, surf(Z) plots the matrix Z versus the grid defined by 1:size(Z,2) along the x axis and 1:size(Z,1) along the y axis.

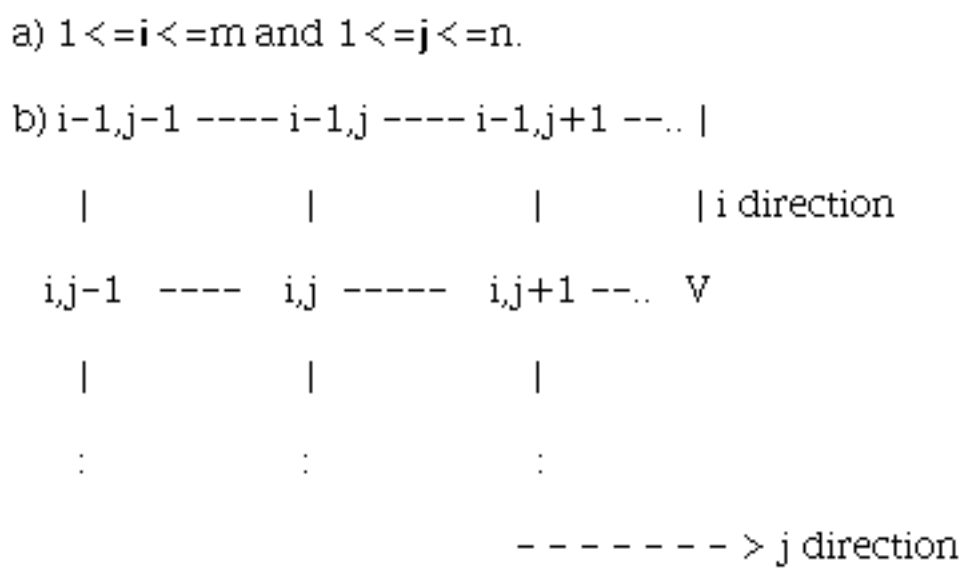
If a {X,Y,Z} triplet is given, Z must be a matrix with size(Z)= [mxn], X or Y can be :

- a) a vector : if X is a vector, length(X)=n. Respectively, if Y is a vector, length(Y)=m.

  b) a matrix : in this case, size(X) (or size(Y)) must equal size(Z).

Color entry specification :

As stated before, the surface is created over a rectangular grid support. Let consider two independant variables i and j such as :



This imaginary rectangular grid is used to build the real surface support onto the XY plane. Indeed, X,Y and Z data have the same size (even if X or Y is vector, see below) and can be considered as 3 functions `x(i,j)`, `y(i,j)` and `z(i,j)` specifying the desired surface. If X or Y are vectors, they are internally treated to produce good matrices matching the Z matrix dimension (and the grid is forcibly a rectangular region).

Considering the 3 functions `x(i,j)`, `y(i,j)` and `z(i,j)`, the portion of surface defining between two consecutive `i` and `j` is called a patch.

By default, when no color matrix is added to a surf call, the color parameter is linked to the Z data. When a `color` matrix is given, it can be applied to the patch in two different ways : at the vertices or at the center of each patch.

That is why, if Z is a [mxn] matrix, the `C color` matrix dimension can be [mxn] (one color defined per vertex) or [`m-1xn-1`] (one color per patch).

Color representation also varies when specifying some GlobalPropery:

The `FaceColor` property sets the shading mode : it can be `'interp'` or `'flat'` (default mode). When `'interp'` is selected, we perform a bilinear color interpolation onto the patch. If size(C) equals size(Z)-1 (i.e. we provided only one color per patch) then the color of the vertices defining the patch is set to the given color of the patch.

When `'flat'` (default mode) is enabled we use a color faceted representation (one color per patch). If size(C) equals size(Z) (i.e. we provided only one color per vertices), the last row and column of C are ignored.

The `GlobalProperty` arguments sould be used to customize the surface. Here is a brief description on how it works:

GlobalProperty
: This option may be used to specify how all the surfaces are drawn. It must always be a couple statement constituted of a string defining the `PropertyName`, and its associated value `PropertyValue` (which can be a string or an integer or... as well depending on the type of the `PropertyName`). Note that you can set multiple properties : the face & edge color, color data, color data mapping, marker color (foreground and background), the visibility, clipping and thickness of the edges of the surface... (see GlobalProperty)

  Note that all these properties can be (re-)set throught the surface entity properties (see surface_properties).

Remarks

By default, successive surface plots are superposed. To clear the previous plot, use `clf()`. To enable `auto_clear` mode as the default mode, edit your default axes doing:

da=gda();

da.auto_clear = 'on'

Enter the command `surf` to see a demo.

Examples

```
// Z initialisation

Z= [   0.0001    0.0013    0.0053   -0.0299   -0.1809   -0.2465   -0.1100   -0.0
    0.0005    0.0089    0.0259   -0.3673   -1.8670   -2.4736   -1.0866   -0.160
    0.0004    0.0214    0.1739   -0.3147   -4.0919   -6.4101   -2.7589   -0.277
   -0.0088   -0.0871    0.0364    1.8559    1.4995   -2.2171   -0.2729    0.836
   -0.0308   -0.4313   -1.7334   -0.1148    3.0731    0.4444    2.6145    2.441
   -0.0336   -0.4990   -2.3552   -2.1722    0.8856   -0.0531    2.6416    2.406
   -0.0137   -0.1967   -0.8083    0.2289    3.3983    3.1955    2.4338    1.212
   -0.0014   -0.0017    0.3189    2.7414    7.1622    7.1361    3.1242    0.663
    0.0002    0.0104    0.1733    1.0852    2.6741    2.6725    1.1119    0.197
    0.0000    0.0012    0.0183    0.1099    0.2684    0.2683    0.1107    0.019

//simple surface
surf(Z); // Note that X and Y are determined by Z dimensions

//same surface with red face color and blue edges
scf(2); // new figure number 2
surf(Z,'facecol','red','edgecol','blu")

// X and Y initialisation
// NB: here, X has the same lines and Y the same columns
X = [ -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.6
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666
   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.666

Y= [   -3.0000   -3.0000   -3.0000   -3.0000   -3.0000   -3.0000   -3.0000   -3
   -2.3333   -2.3333   -2.3333   -2.3333   -2.3333   -2.3333   -2.3333   -2.333
   -1.6667   -1.6667   -1.6667   -1.6667   -1.6667   -1.6667   -1.6667   -1.666
   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.000
   -0.3333   -0.3333   -0.3333   -0.3333   -0.3333   -0.3333   -0.3333   -0.333
    0.3333    0.3333    0.3333    0.3333    0.3333    0.3333    0.3333    0.333
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.000
    1.6667    1.6667    1.6667    1.6667    1.6667    1.6667    1.6667    1.666
    2.3333    2.3333    2.3333    2.3333    2.3333    2.3333    2.3333    2.333
    3.0000    3.0000    3.0000    3.0000    3.0000    3.0000    3.0000    3.000
```

```
// example 1
scf(3)
surf(X,Y,Z)

//example 2
// As you can see, the grid is not necessary rectangular
scf(4)
X(1,4) = -1.5;
Y(1,4) = -3.5;
Z(1,4) = -2;
surf(X,Y,Z)

// example 3
// X and Y are vectors => same behavior as sample 1
// With vectors, the grid is inevitably rectangular
scf(5)// new figure number 5
X=[ -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000    1.66
Y=X;
surf(X,Y,Z)


//LineSpec and GlobalProperty examples:
xdel(winsid()) // destroy all existing figures
surf(Z,Z+5) // color array specified
e=gce();
e.cdata_mapping='direct' // default is 'scaled' relative to the colormap
e.color_flag=3; // interpolated shading mode. The default is 4 ('flat' mode) fo

scf(2)
surf(X,Y,Z,'colorda',ones(10,10),'edgeco','cya','marker','penta','markersiz',20

scf(3)
surf(Z,'cdatamapping','direct')
scf(4)
surf(Z,'facecol','interp') // interpolated shading mode (color_flag == 3)

scf(10)
axfig10=gca();
scf(11);
surf(axfig10,Z,'ydat',[100:109],'marker','d','markerfac','green','markeredg','y

xdel(winsid())
```

See Also

plot2d , clf , xdel , delete , LineSpec , GlobalProperty

Authors

F.Leray

Name

surface_properties — description of the 3D entities properties

Description

The Surface entity is a leaf of the graphics entities hierarchy. Two classes appears under this type of entity : `Plot3d` and `Fac3d` according to the ploting function or the way data is entered. `Fac3d` and `Plo3d` entities are similar but `Fac3d` is more complete and accept more options than `Plot3d`. To always have `Fac3d` entities, simply use `genfac3d` to pre-build matrices before using `plot3d` or use the `surf` command.

Here are the properties contained in a surface entity:

parent:
: This property contains the handle of the parent. The parent of the surface entity should be of type `"Axes"` or `"Compound"`.

children:
: This property contains a vector with the `children` of the handle. However, surface handles currently do not have any `children`.

visible:
: This field contains the `visible` property value for the entity . It should be `"on"` or `"off"` . By default, surfaces are visibles, the value's property is `"on"`. If `"off"` the 3D graphics are not displayed on the screen.

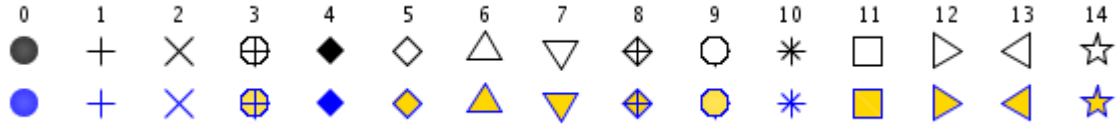
surface_mode:
: This field contains the default `surface_mode` property value for the surface. Its value should be `"on"` (surface drawn) or `"off"` (no surface drawn).

foreground:
: If `color_mode >= 0`, this field contains the color index used to draw the edges. If not, foreground is not used at all. The foreground value should be an integer color index (relative to the current colormap).

thickness:
: This field contains the default `thickness` value of the lines used to draw the facets contours. It should be a positive integer.

mark_mode:
: This field contains the default `mark_mode` property value for the surface. Its value should be `"on"` (marks drawn) or `"off"` (no marks drawn).

mark_style:
: The `mark_style` property value is used to select the type of mark to use when `mark_mode` property is `"on"`. The value should be an integer in [0 14] which stands for: dot, plus, cross, star, filled diamond, diamond, triangle up, triangle down, diamond plus, circle, asterisk, square, triangle right, triangle left and pentagram.The figure below shows the aspects of the marks depending on the `mark_style` and the `mark_foreground` and `mark_background` properties.



mark_size_unit:
: This field contains the default `mark_size_unit` property value. If `mark_size_unit` is set to `"point"`, then the `mark_size` value is directly given in points. When `mark_size_unit`

is set to "tabulated", mark_size is computed relative to the font size array: therefore, its value should be an integer in [0 5] whith stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt. Note that plot3d and pure scilab functions use tabulated mode as default ; when using the surf (or plot for 2D lines) function, the point mode is automatically enabled.

mark_size:
The mark_size property is used to select the type of size of the marks when mark_mode property is "on". Its value should be an integer between 0 and 5 whith stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt.

mark_foreground:
This field contains the mark_foreground property value which is the marks' edge color. Its value should be a color index (relative to the current color_map).

mark_background:
This field contains the mark_background property value which is the marks' face color. Its value should be a color index (relative to the current color_map).

data:
This field defines a tlist data structure of type "3d" composed of a row and column indices of each element as the x-, y- and z-coordinates contained respectivly in data.x,data.y and data.z. The complementary field named data.color is available in case a real color vector or matrix is specified. If none, data.color is not listed. The surface is painted according to color_mode and color_flag properties.

color_mode:
an integer between [-size(colormap) ; size(colormap)] defining the color of the facet when color_flag value is 0. As stated before, if color_mode > 0, edges are drawn using foreground color. If color_mode is set to 0, a mesh of the surface is drawn: front faces have no colors. Finally, when color_mode < 0, front faces are painted with color -color_mode but no edges are displayed.

color_flag:
This field is used to specify the algorithm used to set facets' colors.

Not that the rules on color_mode, foreground and hiddencolor are still applied to this case.

- color_flag == 0
  - All facets are painted using the color index and method defined by color_mode (see above).
- color_flag == 1
  - All facets are painted using one color index per facet proportional to z. The minimum z value is painted using the index 1 color while the maximum z value is painted using highest color index. The edges of the facets can be additionnaly drawn depending on the value of color_mode (see above).
- The 3 remaining cases (color_flag== 2,3 or 4) are only available only with Fac3d entity. Then, the data.color value is used to set colors for facets (indices in the current colormap) if it exists. If not, the current color_mode is used to paint the facets.
- color_flag == 2 ('flat' shading)
  - All facets are painted using the color index given in the data.color property (one color per facet is needed). Two cases are then possible :
  - data.color contains a color vector : if color(i) is positive it gives the color of facet i and the boundary of the facet is drawn with current line style and color. If color(i) is negative, color id -color(i) is used and the boundary of the facet is not drawn.

`data.color` contains a color matrix of size (nf,n) where `n` stands for the number of facets and `nf` for the number of points defining the polygonal facet. For the `nf` vertices defining each facet, the algorithm computes an `average value` of the color index (from the matrix color index) : the `nf` vertices of the same facet will have the same color index value.

- `color_flag == 3 ('interpolated' shading)`
  - Facets painting results of interpolation of vertices colors. The indices of vertices color are given in the `data.color` property (one color per vertex is needed). Two cases are possible :
  - `data.color` contains a `colors` vector : then, there are too few data to complete the interpolated shading mode. Indeed, a color matrix of size (nf,n) (where `n` stands for the number of facets and `nf` for the number of points defining the polygonal facet) is needed to perform this operation. For each facet, the algorithm copies the single color index value of the facet into the `nf` color indexes vertices defining the facet's boundary.

    `data.color` contains a color matrix of size (nf,n) (see upper for `nf` and `n` definitions), the interpolated shading mode can be completed normally using those color indexes.
- `color_flag == 4 (Matlab-like 'flat' shading)`
  - Same as `color_flag==2` with a slight difference when `data.color` is a matrix. All facets are painted using the color index given in the `data.color` property (one color per facet is needed). Two cases are then possible :
  - `data.color` contains a `color` vector : if `color(i)` is positive it gives the color of facet `i` and the boundary of the facet is drawn with current line style and color. If `color(i)` is negative, color id `-color(i)` is used and the boundary of the facet is not drawn.

    `data.color` contains a color matrix of size (nf,n) where `n` stands for the number of facets and `nf` for the number of points defining the polygonal facet. For the `nf` vertices defining each facet, the algorithm takes the `color of the first vertex` defining the patch (facet).

cdata_mapping:
: `Specific to Fac3d handles`. A string with value `'scaled'` or `'direct'`. If a `data.color` is set, each index color data specifies a single value for each vertex. `cdata_mapping` determines wether those indices are scaled to map linearly into the current colormap (`'scaled'` mode) or point directly into this colormap (`'direct'` mode). This property is usefull when `color_flag` equals 2,3 or 4.

hiddencolor:
: This field contains the color index used to draw the backward faces of a surface. Its value should be a positive integer (color index relative to the current colormap). If it is a negative integer,the same color than the "visible" face is applied to the rear face.

clip_state:
: This field contains the `clip_state` property value for the surface. It should be :
  - `"off"` this means that the surface is not clipped.
  - `"clipgrf"` this means that the surface is clipped outside the Axes box.
  - `"on"` this means that the surface is clipped outside the rectangle given by property clip_box.

clip_box:
: This field is to determinate the `clip_box` property. By Default its value should be an empty matrix if clip_state is "off". Other cases the vector `[x,y,w,h]` (upper-left point width height) defines the portions of the surface to display, however `clip_state` property value will be changed.

user_data:
This field can be use to store any scilab variable in the surface data structure, and to retreive it.

Examples

```
//create a figure
t=[0:0.3:2*%pi]'; z=sin(t)*cos(t');
[xx,yy,zz]=genfac3d(t,t,z);
plot3d([xx xx],[yy yy],list([zz zz+4],[4*ones(1,400) 5*ones(1,400)]))
h=get("hdl") //get handle on current entity (here the surface)
a=gca(); //get current axes
a.rotation_angles=[40,70];
a.grid=[1 1 1];
//make grids
a.data_bounds=[-6,0,-1;6,6,5];
a.axes_visible="off";
//axes are hidden a.axes_bounds=[.2 0 1 1];
f=get("current_figure");
//get the handle of the parent figure
f.color_map=hotcolormap(64);
//change the figure colormap
h.color_flag=1;
//color according to z
h.color_mode=-2;
//remove the facets boundary
h.color_flag=2;
//color according to given colors
h.data.color=[1+modulo(1:400,64),1+modulo(1:400,64)];
//shaded
h.color_flag=3;

scf(2); // creates second window and use surf command
subplot(211)
surf(z,'cdata_mapping','direct','facecol','interp')

subplot(212)
surf(t,t,z,'edgeco','b','marker','d','markersiz',9,'markeredg','red','markerfac
e=gce();
e.color_flag=1 // color index proportional to altitude (z coord.)
e.color_flag=2; // back to default mode
e.color_flag= 3; // interpolated shading mode (based on blue default color beca
```

See Also

set , get , delete , plot3d , plot3d1 , plot3d2 , surf , graphics_entities

Authors

Djalel ABDEMOUCHE & F.Leray

Name

swap_handles — Permute two handles in the graphic Hierarchy.

```
swap_handle( handle1, handle2 )
```

Parameters

handle1
: first handle of the permutation.

handle2
: second handle of the permutation.

Description

The **swap_handles** function allows to permute two handles in the graphic hierarchy. The first handle will take the second handle position and vice versa.

Since not every handles are compatible with each others, some restrictions exist when swapping handles. For examples, it is not allowed to swap a polyline with an axes handle, since their would not be compatible with their new parents.. More information about compatibility can be found in the graphics_entities page.

This routine may be used on children of the same parent to change their indices..

Examples

```
//-----------------//
//  First example         //
//-----------------//

// create a rectangle
xrect( 0.5, 0.5,0.5,0.5) ;
rect = gce() ;

// create a circle
xarc( 0.5, 0.5, 0.5, 0.5, 0, 64 * 360 ) ;
circle = gce() ;

// create an arrow
xpoly([0,1],[0,1]) ;
arrow = gce() ;
arrow.polyline_style = 4 ;
arrow.arrow_size_factor = 4 ;

// get the list of children
axes = gca()  ;
axes.children

// change the order
swap_handles( rect, arrow ) ;
swap_handles( arrow, circle ) ;

// get the new order
axes.children
```

```
//-----------------//
//  Second example    //
//-----------------//

// create two windows
plot2d ;
axes1 = gca() ;

scf() ;
fec ;
axes2 = gca() ;

// swap their axes
// note that the color map does not change.
swap_handles( axes1, axes2 ) ;
```

See Also

graphics_entities , copy , delete , relocate_handle

Authors

Jean-Baptiste Silvy

Name

text_properties — description of the Text entity properties

Description

The Text entity is a leaf of the graphics entities hierarchy. This entity defines the parameters for string drawing

parent:
: This property contains the handle of the parent. The parent of the text entity should be of the type `"Axes"` or `"Compound"`.

children:
: This property contains a vector with the `children` of the handle. However, text handles currently do not have any `children`.

visible:
: This field contains the `visible` property value for the entity . It should be `"on"` or `"off"` .By default, the text is visible, the value's property is `"on"`. If `"off"` the text is not displayed on the screen.

text:
: the matrix containing the strings of the object. The rows of the matrix are displayed horizontally and the columns vertically.

alignment:
: Specify how the strings are aligned in their columns. The value must be '**left**', '**center**' or '**right**'.

data:
: This field is the vector `[x,y,[z]]` of the origin of the text in the data units of the axes.

box:
: This field takes the values `"on"` or `"off"`. If `"on"` a box is draw around the text with a line on its edge and a background.

line_mode:
: This boolean property allows to draw or not a line around the box when the `box` property is `"on"`. If `line_mode` is `"off"`, the line of the box is not drawn.

fill_mode:
: This boolean property allows to draw or not the background of the box when the `box` property is `"on"`. If `fill_mode` is `"off"`, the background of the box is not transparent.

text_box:
: A two dimensionnal vector specifying the size of a rectangle in user coordinates. The rectangle is used when the text_box_mode property is set to '**centered**' or '**filled**'.

text_box_mode:
: May have three different value : '**off**', '**centered**' or '**filled**'. If '**off**', the strings are displayed using the given font and the data field specifies the position of the lower-left point of the text. If '**centered**', the text is displayed in the middle of the rectangle whose size is given by text_box. If '**filled**' the font size of the strings will be expanded to fill the rectangle.

  When using '**off**' or '**centered**' modes, text size remains constant upon zooming. They are the best modes to create annotations in a graph. On the contrary, when using the '**filled**' mode, the text size follow the graphic scale. It is then possible to zoom upon text objects.

font_foreground:
: This field contains the color used to display the characters of the text. Its value should be a color index (relative to the current colormap).

foreground:
This field contains the color used to display the line on the edge of the box. Its value should be a color index (relative to the current colormap).

background:
This field contains the color used to fill the box around of the text. Its value should be a color index (relative to the current colormap).

font_size:
It is a scalar specifying the displayed characters size. If `fractional_font` property is `"off"` only the integer part of the value is used. For more information see graphics_fonts.

font_style:
Specifies the font used to display the character strings. This is a positive integer referecing one of the loaded fonts. Its value must be between 0, referecing the first font, and the number of loaded fonts minus one, referencing the last font. For more information see graphics_fonts.

fractional_font:
This property specify whether text is displayed using fractional font sizes. Its value must be either `"on"` or `"off"`. If `"on"` the floating point value of `font_size` is used for display and the font is anti-aliased. If `"off"` only the integer part is used and the font is not smoothed.

font_angle:
This property determines the orientation of the text string. Specify value of rotation in degrees.

clip_state:
This field contains the `clip_state` property value for the text. Its value should be :

- `"off"` this means that the text is not clipped.
- `"cliprf"` this means that the text is clipped outside the Axes box.
- `"on"` this means that the text is clipped outside the rectangle given by the property `clip_box`.

clip_box:
This field contains the `clip_box` property. Its value should be an empty matrix if clip_state is "off" or the vector `[x,y,w,h]` (upper-left point width height).

user_data:
This field can be use to store any scilab variable in the text data structure, and to retreive it.

Examples

```
a=get("current_axes");
a.data_bounds=[0,0;1,1];
a.axes_visible = 'on' ;

xstring(0.5,0.5,"Scilab is not esilaB",0,0)

t=get("hdl")   //get the handle of the newly created object

t.font_foreground=6; // change font properties
t.font_size=5;
t.font_style=5;

t.text=["SCILAB","is";"not","esilaB"] ; // change the text
t.font_angle=90 ; // turn the strings
t.text_box = [0,0] ;
t.text_box_mode = 'centered' ; // the text is now centered on [0.5,0.5].
```

```
t.alignment = 'center' ;
t.box = 'on' ; // draw a box around the text
```

See Also

set , get , delete , xtitle , graphics_entities

Authors

Djalel ABDEMOUCHE, Jean-Baptiste SILVY

Name

title — display a title on a graphic window

```
title(my_title)
title(my_title,<Property>)
title(<axes_handle>,<my_title>,<Property>)
```

Parameters

my_title
: a string, it's the title to display

<Property>
: This optional argument represents a sequence of couple statements `{PropertyName,PropertyValue}` that defines global objects' properties applied to the created title.

<axes_handle>
: This optional argument forces the title to appear inside the selected axes given by `axes_handle` rather than the current axes (see gca).

Description

`title` displays a title on a graphic window.

The `Property` arguments should be used to customize the title. Here is a complete list of the available options.

Property :

`backgroundcolor` : this field contains the color used to fill the box if any. Its value should be a color index (relative to the current colormap).

`color` : this field contains the color used to display the title text. Its value should be a color index (relative to the current colormap).

`edgecolor` : this field contains the color used to display the line around the box if any. Its value should be a color index (relative to the current colormap).

`fontname` : seven differents fonts are available : "Courrier", "Symbol", "Times", "Times Italic", "Times Bold", "User defined". The font_size property is an index in [0 6] which is associated to the previous font names.

`fontsize` : the fontsize property is used to select the type of size of the title. Its value should be an integer in between 0 and 5 which stands for 8pt, 10pt, 12pt, 14pt, 18pt and 24pt.

`position` : this 2d vector allows you to place manually the title on the screen. The position is stored in the data units of the axes.

`rotation` : this scalar allows you to turn the title. The font is turned inverse clockise with the angle given in degrees.

`visible` : this field contains the visible property value for the title. It should be "on" or "off". By default, the label is visible, the value's property is "on" . If "off" the title is not displayed on the screen.

Examples

```
// display a title with several properties
```

```
title('my title');
// change the color for the font
title('my title','color','blue');
// change the color for the around the box
title('my title','edgecolor','red');
// change the position of the title
title('my title','position',[0.3 0.8]);
// change the size of the font
title('my title','fontsize',3);
// a rotation
title('my title','rotation',90);

// We can do all these modifications with just the below instruction:
title('my title','color','blue','edgecolor','red','fontsize',3,'rotation',90,'p
```

See Also

label_properties , titlepage , xtitle

Authors

F.Belahcene

Name

titlepage — add a title in the middle of a graphics window

```
titlepage(str)
```

Parameters

str
: matrix of strings

Description

`titlepage` displays the matrix of strings `str` in the middle of the current graphics window with a font as big as possible.

See Also

xtitle

Authors

S. S.

Name

twinkle — is used to have a graphics entity twinkle

```
twinkle(h,[n])
```

Parameters

h
: handle of a graphics entity.

n
: integer.

Description

`twinkle` let the graphics entity given by its handle `h` twinkle. It can be used to find the graphics object corresponding to a graphics handle in the graphics window. By default the graphics entity twinkles 5 times, but you can change this by using optional argument `n`.

Examples

```
x=linspace(-2*%pi,2*%pi,100)';
plot2d(x,[sin(x),cos(x)]);
e=gce(); p1=e.children(1); p2=e.children(2);
// cos plot twinkle
twinkle(p1)
// sin plot twinkle 10 times
twinkle(p2,10)
// axes twinkle
twinkle(gca())
```

See Also

graphics_entities

Name

unglue — unglue a coumpound object and replace it by individual children.

```
unglue(h)
H=unglue(h)
```

Parameters

h
: a handle on an Compound.

H
: a vector of handle on the resulting entities after unCompound.

Description

Given a handle on an Compound entity, this function destroies the Compound and unpacks the elementary entities to associated them to its parent. `glue` returns a vector of handles on these individual children.

See Also

get , set , copy , glue , graphics_entities

Authors

Djalel ABDEMOUCHE

Name

unzoom — unzoom graphics

```
unzoom()
unzoom(H)
```

Parameters

H
: A vector of Figure or Axes handle.

Description

`unzoom()` is used to remove the zoom effect for all the axes of the current graphic figure:

`unzoom(H)` is used to remove the zoom effect for all the Figures and Axes given by the vector of handles `H`. Removing zoom effect for a Figure is the equivalent of removing zoom effect on all its Axes children.

Examples

```
clf()
x=0:0.01:6*%pi;
plot2d(x,sin(x^2))
zoom_rect([16,-1,18,1])
unzoom()


//subplots accordingly
clf()
x=0:0.01:6*%pi;
subplot(211)
plot2d(x,cos(x))
a1=gca();
subplot(212)
plot2d(x,cos(2*x))
a2=gca();
rect=[3 -2 7 10]; // a rectangle specified in the current axes (last one) c
zoom_rect(rect)
unzoom(a1) // unzoom first plot only
unzoom(a2) // unzoom second plot only
zoom_rect(rect) // zoom again
unzoom(gcf()) // unzoom all the axes, equivalent to unzoom()
```

See Also

zoom_rect , axes_properties

Authors

Serge Steer INRIA

Jean-Baptiste Silvy INRIA

Name

whitecolormap — completely white colormap

```
cmap=whitecolormap(n)
```

Parameters

n
: integer >= 3, the colormap size.

cmap
: matrix with 3 columns `[R,G,B]`.

Description

This colormap is completely white

Examples

```
f = scf();
plot3d1();
f.color_map = whitecolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , wintercolormap

Name

winsid — return the list of graphics windows

```
x=winsid()
```

Parameters

x
: row vector.

Description

`winsid` is used to get the list of graphics windows as a vector of windows numbers.

Name

wintercolormap — blue to green colormap

```
cmap=wintercolormap(n)
```

Parameters

n
: integer >= 3, the colormap size.

cmap
: matrix with 3 columns `[R,G,B]`.

Description

`wintercolormap` computes a colormap with `n` colors varying from blue to green.

Examples

```
f = scf();
plot3d1();
f.color_map = wintercolormap(32);
```

See Also

colormap , autumncolormap , bonecolormap , coolcolormap , coppercolormap , graycolormap , hotcolormap , hsvcolormap , jetcolormap , oceancolormap , pinkcolormap , rainbowcolormap , springcolormap , summercolormap , whitecolormap

Name

xarc — draw a part of an ellipse

```
xarc(x,y,w,h,a1,a2)
```

Parameters

x,y,w,h
: four real values defining a rectangle.

a1,a2
: real values defining a sector.

Description

`xarc` draws a part of an ellipse contained in the rectangle `(x,y,w,h)` (upper-left point, width, height), and in the sector defined by the angle `alpha1` and the angle `alpha1+alpha2`. `alpha1` and `alpha2` are given respectively by `a1/64` degrees and `a2/64` degrees. This function uses the current graphics color and user coordinates.

Examples

```
// isoview scaling
plot2d(0,0,-1,"031"," ",[-2,-2,2,2])
xset("color",3)
xarc(-1,1,2,2,0,90*64)
xarc(-1.5,1.5,3,3,0,360*64)
```

See Also

xarcs , xfarc , xfarcs

Authors

J.Ph.C.

Name

xarcs — draw parts of a set of ellipses

```
xarcs(arcs,[style])
```

Parameters

arcs
: matrix of size (6,n) describing the ellipses.

style
: row vector of size n giving the style to use.

Description

`xarcs` draws parts of a set of ellipses described by `arcs`: `arcs=[x y w h a1 a2;x y w h a1 a2;...]'` where each ellipse is defined by the 6 parameters `(x,y,w,h,a1,a2)` (see `xarc`).

`x, y, w, h` parameters are specified in user coordinates.

`style(i)` gives the color used to draw ellipse number `i`.

Examples

```
plot2d(0,0,-1,"031"," ",[-1,-1,1,1])
arcs=[-1.0 0.0 0.5; // upper left x
       1.0 0.0 0.5; // upper left y
       0.5 1.0 0.5; // width
       0.5 0.5 1.0; // height
       0.0 0.0 0.0; // angle 1
       180*64 360*64 90*64]; // angle 2
xarcs(arcs,[1,2,3])
```

See Also

xarc , xfarc , xfarcs

Authors

J.Ph.C.;

Name

xarrows — draw a set of arrows

```
xarrows(nx,ny,[arsize,style])
```

Parameters

nx,ny
: real vectors or matrices of same size.

arsize
: real scalar, size of the arrow head. The default value can be obtained by setting arsize to -1.

style
: matrix or scalar. If `style` is a positive scalar it gives the color to use for all arrows. If it is a negative scalar then the current color is used. If it is a vector `style(i)` gives the color to use for arrow `i`.

Description

`xarrows` draws a set of arrows given by `nx` and `ny`. If `nx` and `ny` are vectors, the ith arrow is defined by `(nx(i),ny(i))-->(nx(i+1),ny(i+1))`. If `nx` and `ny` are matrices:

```
nx=[xi_1 x1_2 ...; xf_1 xf_2 ...]
ny=[yi_1 y1_2 ...; yf_1 yf_2 ...]
```

the `k` th arrow is defined by `(xi_k,yi_k)-->(xf_k,yf_k)`.

`xarrows` uses the current graphics scale which can be set by calling a high level drawing function such as `plot2d`.

Examples

```
x=2*%pi*(0:9)/8;
x1=[sin(x);9*sin(x)];
y1=[cos(x);9*cos(x)];
plot2d([-10,10],[-10,10],[-1,-1],"022")
xset("clipgrf")
xarrows(x1,y1,1,1:10)
xset("clipoff")
```

Authors

J.Ph.C.

Name

xbasc — clears a graphics window.

```
xbasc([window-id])
```

Parameters

window-id
: integer scalar or vector

Description

Without any argument, this function clears the current graphic figure by deleting all its children. Otherwise it clears the graphic figures whose ids are included in the vector `window-id`. For example `xbasc(1:3)` clears windows 1, 2 and 3. If one of the windows does not exist, then it is automatically created.

`xbasc` function delete every children of specified windows including menus and uicontrols added by user. To prevent menus and uicontrols from being deleted, the `delete(gca())` command might be used instead.

Function `xbasc` is obsolete. To erase a figure, please use instead the `clf` or `delete` functions.

See Also

clf , xclear

Name

xbasr — redraw a graphics window

```
xbasr(win_num)
```

Description

`xbasr` is used to redraw the content of the graphics window with id `win_num`. It works only with the driver `"Rec"`.

See Also

driver , replot

Authors

J.Ph.C.

Name

xchange — transform real to pixel coordinates

```
[x1,y1,rect]=xchange(x,y,dir)
```

Parameters

x,y
: two matrices of size (n1,n2) (coordinates of a set of points).

dir
: parameter used to specify the conversion type (See "Description" for details)

x1,y1
: two matrices of size (n1,n2) (coordinates of the set of points).

rect
: a vector of size 4.

Description

After having used a graphics function, `xchange` computes pixel coordinates from real coordinates and conversely, according to the value of the parameter `dir`: "f2i" (float to int) means real to pixel and "i2f" (int to float) means pixel to real. `x1` and `y1` are the new coordinates of the set of points defined by the old coordinates `x` and `y`.

`rect` is the coordinates in pixel of the rectangle in which the plot was done: [upper-left point, width, height].

Examples

```
t=[0:0.1:2*%pi]';
plot2d(t,sin(t))
[x,y,rect]=xchange(1,1,"f2i")
[x,y,rect]=xchange(0,0,"i2f")
```

Authors

J.Ph.C.

Name

xclear — clears a graphics window

```
xclear([window-id])
```

Parameters

window-id
: integer scalar or vector

Description

Without any argument, this function clears the current graphic figure by turning its visibility to 'off'. Otherwise it clears the graphics figures whose numbers are included in the vector `window-id`. For example `xclear(1:3)` clears windows 1, 2 and 3. If one of the windows does not exist, then it is automatically created.

Function `xclear` is obsolete. To clear a figure, please use instead the `clf` function or the `visible` property.

See Also

xbasc

Authors

J.Ph.C.

Name

xclick — Wait for a mouse click.

```
[ibutton,xcoord,yxcoord,iwin,cbmenu]=xclick([flag])
```

Parameters

ibutton
: Real scalar (integer value): mouse button number, key code... (See description below).

xcoord
: Real scalar: x-ccordinate of the mouse pointer when the clic occured, in current graphic scale.

ycoord
: Real scalar: x-ccordinate of the mouse pointer when the clic occured, in current graphic scale.

iwin
: Real scalar (integer value): number of the window where the action occured.

cbmenu
: String: callback associated to a menu if `xclick` returns due to a click on a menu. In this case, `ibutton`, `xcoord`, `ycoord`, and `iwin` take arbitrary values.

flag
: Real scalar (integer value): If present, the click event queue is not cleared when entering xclick.

Description

`xclick` waits for a mouse click in the graphics window.

If it is called with 3 left hand side arguments, it waits for a mouse click in the current graphics window.

If it is called with 4 or 5 left hand side arguments, it waits for a mouse click in any graphics window.

The values of `ibutton` are described below.

ibutton==0
: Left mouse button has been pressed.

ibutton==1
: Middle mouse button has been pressed.

ibutton==2
: Right mouse button has been pressed.

ibutton==3
: Left mouse button has been clicked.

ibutton==4
: Middle mouse button has been clicked.

ibutton==5
: Right mouse button has been clicked.

ibutton==10
: Left mouse button has been double-clicked.

ibutton==11
Middle mouse button has been double-clicked.

ibutton==12
Right mouse button has been double-clicked.

ibutton >=32
key with ASCII code `ibutton` has been pressed.

ibutton <=32
key with ASCII code `-ibutton` has been released.

ibutton >=1000+32
key with ASCII code `ibutton-1000` has been pressed while CTRL key pressed.

ibutton==-1000
graphic window has been closed.

WARNING: `ibutton` was equal to -100 for graphic window closure up to Scilab 4.1.2, but this code has been changed (in Scilab 5.0) because it was also the code returned for `d` key release.

ibutton==-2
A dynamic menu has been selected and its callback is returned in `cbmenu`.

See Also

locate, xgetmouse, seteventhandler

Authors

J.Ph.C.

V.C.

Name

xclip — (obsolete) set a clipping zone

```
xclip([x,y,w,h])
xclip(rect)
xclip("clipgrf")
```

Parameters

x,y,w,h
: real values.

rect
: row vector of size 4.

Description

`xclip` set a clipping zone given by the coordinates, in the current graphics scale, of the rectangle `x,y,w,h` (upper-left point, width, height). If only one argument is given, it stands for a rectangle specification `rect=[x,y,w,h]`.

`xclip("clipgrf")` is used to clip the usual rectangle boundaries.

To unclip a region use the command `xclip()`.

Function `xclip` is obsolete and will be permanently removed in Scilab 5.2. To set a clipping zone, please use instead the `clip_state` and `clip_box` properties of graphic entities.

Examples

```
x=0:0.2:2*%pi;
x1=[sin(x);100*sin(x)];
y1=[cos(x);100*cos(x)];
y1=y1+20*ones(y1);

// set the frame
clf();a=gca();a.data_bounds=[-100 -100;500 600];

// No clipping
xsegs(10*x1+200*ones(x1),10*y1+200*ones(y1))
e=gce(); //handle on the Segs entity

// draw rectangle clipping zone
xrect(150,460,100,150)
// set clip_box for  Segs entity
e.clip_box=[150,460,100,150];

// Set usual rectangle boundaries clipping zone
e.clip_state='clipgrf';
xclip("clipgrf")
// remove clipping
e.clip_state='off';
```

See Also

axes_properties

Authors

J.Ph.C.

Name

xdel — delete a graphics window

```
xdel([win-nums])
```

Parameters

win-nums
: integer or integer vector

Description

`xdel` deletes the graphics windows `win-nums` or the current graphics window if no argument is given.

Authors

J.Ph.C.

Name

xfarc — fill a part of an ellipse

```
xfarc(x,y,w,h,a1,a2)
```

Parameters

x,y,w,h
: four real values defining a rectangle.

a1,a2
: real values defining a sector.

Description

`xfarc` fills a part of an ellipse contained in the rectangle `(x,y,w,h)` (upper-left point, width, height), and in the sector defined by the angle `alpha1` and the angle `alpha1+alpha2`. `alpha1` and `alpha2` are given respectively by `a1/64` degrees and `a2/64` degrees. This function uses the current color and user coordinates.

Examples

```
// isoview scaling
plot2d(0,0,-1,"031"," ",[-2,-2,2,2])
xfarc(-0.5,0.5,1,1,0,90*64)
xset("color",2)
xfarc(0.5,0.5,1,1,0,360*64)
```

See Also

xarc , xarcs , xfarcs

Authors

J.Ph.C.;

Name

xfarcs — fill parts of a set of ellipses

```
xfarcs(arcs,[style])
```

Parameters

arcs
: matrix of size (6,n) describing the ellipses.

style
: row vector of size n giving the colors to use.

Description

`xarcs` fills parts of a set of ellipses described by `arcs`: `arcs=[x y w h a1 a2;x y w h a1 a2;...]'` where each ellipse is defined by the 6 parameters `(x,y,w,h,a1,a2)` (see `xfarc`).

`x, y, w, h` parameters are specified in user coordinates.

`style(i)` gives the color number for filling ellipse number `i`.

Examples

```
plot2d(0,0,-1,"031"," ",[-1,-1,1,1])
arcs=[-1.0 0.0 0.5; // upper left x
       1.0 0.0 0.5; // upper left y
       0.5 1.0 0.5; // width
       0.5 0.5 1.0; // height
       0.0 0.0 0.0; // angle 1
       180*64 360*64 90*64]; // angle 2
xfarcs(arcs,[1,2,3])
```

See Also

xarc , xfarc , xfarc

Authors

J.Ph.C.

Name

xfpoly — fill a polygon

```
xfpoly(xv,yv,[close])
```

Parameters

xv,yv
: two vectors of same size (the points of the polygon).

close
: integer. If close=1, the polyline is closed; default value is 0.

Description

`xfpoly` fills a polygon with the current color. If `close` is equal to 1 a point is added to the polyline `xv,yv` to define a polygon.

Examples

```
x=sin(2*%pi*(0:4)/5);
y=cos(2*%pi*(0:4)/5);
plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("color",5)
xfpoly(x,y)

// News graphics only
e=gce(); // get the current entity (the last created: here the polyline)
e.fill_mode='off';
e.closed = 'off' // the polyline is now open

xset("default")
```

See Also

xfpolys , xpoly , xpolys

Authors

J.Ph.C.

Name

xfpolys — fill a set of polygons

```
xfpolys(xpols,ypols,[fill])
```

Parameters

xpols,ypols
: matrices of the same size (p,n) (points of the polygons).

fill
: vector of size n or of size (p,n)

Description

`xfpolys` fills a set of polygons of the same size defined by the two matrices `xpols` and `ypols`. The coordinates of each polygon are stored in a column of `xpols` and `ypols`.

The polygons may be filled with a given color (flat) or painted with interpolated (shaded) colors.

flat color painting
: In this case `fill` should be a vector of size `n`. The pattern for filling polygon number i is given by `fill(i)`:

  - if `fill(i)<0`, the polygon is filled with pattern id `-fill(i)`.
  - if `fill(i)=0`, the polygon is drawn with the current dash style (or current color) and not filled.
  - if `fill(i)>0`, the polygon is filled with pattern id `fill(i)`. Then its contour is drawn with the current dash (or color) and closed if necessary.

interpolated color painting
: In this case `fill` should be a matrix with same sizes as `xpols` and `ypols`. Note that `p` must be equal to 3 or 4.

  `fill(k,i)` gives the color at the `k`th edge of polygon `i`.

Examples

```
a=gca();a.data_bounds=[0,-10;210,40];a.foreground=color('red');
x1=[0,10,20,30,20,10,0]';
y1=[15,30,30,15,0,0,15]';
xpols=[x1 x1 x1 x1]; xpols=xpols+[0,60,120,180].*.ones(x1);
ypols=[y1 y1 y1 y1];
xfpolys(xpols,ypols,[-1,0,1,2])

// interpolated colors
clf()
f=gcf();
a=gca();a.data_bounds=[0,-10;40,30];a.isoview='on';
x1=[0,10,20,10]';
y1=[10,0,10,20]';
c=linspace(2,100,4)';
```

```
xpols=[x1 x1+20 x1+10 x1+10];
ypols=[y1 y1    y1+10 y1-10];
cols= [c c($:-1:1) c([3 4 1 2]) c]
f.color_map=jetcolormap(max(cols));
xfpolys(xpols,ypols,cols)

// interpolated colors
clf()
f=gcf();
x11=[0;20;20;0];y11=[10;10;30;30];c11=[10;10;30;30];
x12=x11;y12=y11+20;c12=[20;20;1;1];c12=[30;30;10;10];
x21=[0;30;30;0]+22;y21=[20;20;30;30];c21=[20;20;30;30];
x22=x21;y22=y21+10;c22=[30;30;20;20];
x31=[0;40;40;0]+55;y31=[0;0;30;30];c31=[0;0;30;30];
x32=x31;y32=y31+30;c32=[30;30;0;0];
X=[x11 x12 x21 x22 x31 x32];Y=[y11 y12 y21 y22 y31 y32];C=([c11 c12 c21 c22
a=gca();a.isoview='on';
a.data_bounds=[min(X),min(Y);max(X),max(Y)];
f=gcf();f.color_map=graycolormap(max(C));
xfpolys(X,Y,C)
```

See Also

xfpoly , xpoly , xpolys

Authors

J.Ph.C.

Name

xfrect — fill a rectangle

```
xfrect(x,y,w,h)
xfrect(rect) // rect =[x,y,w,h]
```

Parameters

x,y,w,h
: four real values defining the rectangle.

Description

`xrect` fills a rectangle defined by `[x,y,w,h]` (upper-left point, width, height) in user coordinates.

Examples

```
plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("color",5)
xfrect(-1,1,2,2)
xset("default")
```

See Also

xrect , xrects

Authors

J.Ph.C.

Name

xget — get current values of the graphics context. **This function is obsolete**.

```
[x1]=xget(str,[flag])
xget()
```

Parameters

str
: string.

flag
: optional. Set to 1 gives a verbose mode.

Description

Warning this function is obsolete. Use the Scilab graphic objetcs representation instead (see the set and get functions as well as the graphics_entities help page).

This function is used to get values from the graphics context on the topic specified by the string `str`. When called with no argument, a choice menu is created showing the current values and changes can be performed through toggle buttons.

number=xget("alufunction")
: Get the logical function number used for drawing. See `xset`.

str=xget("auto clear")
: Get the auto clear status (`"on"` or `"off"`).

color=xget("background")
: Get the background color of the current Axes object. The result is a colormap index corresponding to the color.

rect=xget("clipping")
: Get the clipping zone as a rectangle rect=[x,y,w,h] (Upper-Left point Width Height).

c=xget("color")
: Get the default color for filling, line or text drawing functions. `c` is an integer projected in the interval [0,whiteid]. 0 stands for black filling and whiteid for white. The value of whiteid can be obtained with `xget("white")`.

cmap=xget("colormap")
: Get the colormap used for the current graphics window as a m x 3 RGB matrix.

dash=xget("dashes")
: Get the dash style dash=[dash_number] where dash_number is the id of the dash. This keyword is obsolete, please use `xget("color")` or `xget("line style")` instead.

font=xget("font")
: Get font=[fontid,fontsize], the default font and the default size for fonts. size.

fontsize=xget("font size")
: Get the default size for fonts size.

color=xget("foreground")
: Get the foreground color of the current Axes object. The result is a colormap index corresponding to the color.

str=xget("fpf")
Get the floating point format for number display in contour functions. Note that `str` is `""` when default format is used.

color=xget("hidden3d")
Get the color number for hidden faces in plot3d.

pat=xget("lastpattern")
Get the id of the last available pattern or color, with the current colormap of the current window. In fact `pat+1` and `pat+2` are also available and stand respectively for black and white pattern.

type=xget("line mode")
Get the line drawing mode. type=1 is absolute mode and type=0 is relative mode. (Warning: the mode type=0 is has bugs)

xget("line style")
Get the default line style (1: solid, >1 for dashed lines).

mark=xget("mark")
Get the default mark id and the default marks size. `mark=[markid,marksize]`.

marksize=xget("mark size")
Get the default marks size.

pat=xget("pattern")
Get the current pattern or the current color. `pat` is an integer in the range `[1,last]`. When one uses black and white, 0 is used for black filling and `last` for white. The value of `last` can be obtained with `xget("lastpattern")`.

value=xget("thickness")
Get the thickness of lines in pixel (0 and 1 have the same meaning: 1 pixel thick).

flag=xget("use color")
Get the flag 0 (use black and white) or 1 (use colors). See `xset`.

[x,y]=xget("viewport")
Get the current postion of the visible part of graphics in the panner.

dim=xget("wdim")
Get the width and the height of the current graphics window dim=[width,height].

win=xget("window")
Get the current window number `win`.

pos=xget("wpos");
Get the position of the upper left point of the graphics window pos=[x,y].

See Also

xset, getcolor, getsymbol, ged, set, graphics_entities

Authors

J.Ph.C.

Name

xgetech — get the current graphics scale

```
[wrect,frect,logflag,arect]=xgetech()
```

Parameters

wrect,frect
: real vectors.

logflag
: string of size 2 "xy".

Description

`xgetech` returns the current graphics scale (of the current window). The rectangle [xmin,ymin,xmax,ymax] given by `frect` is the size of the whole graphics window. The plotting will be made in the region of the current graphics window specified by `wrect`.

`wrect=[x,y,w,h]` (upper-left point, width, height) describes a region inside the graphics window. The values in `wrect` are specified using proportion of the width and height of the graphics window:

`wrect=[0 0 1 1]` means that the whole graphics window is used.

`wrect=[0.5 0 0.5 1]` means that the graphics region is the right half of the graphics window.

`logflag` is a string of size 2 "xy", where x and y can be "n" or "l". "n" stands for normal (linear) scale and "l" stands for logscale. x stands for the x-axis and y stands for the y-axis.

`arect=[x_left, x_right,y_up,y_down]` gives the frame size inside the subwindow. The graphic frame is specified (like `wrect`) using proportion of the width or height of the current graphic subwindow. Default value is `1/8*[1,1,1,1]`. If `arect` is not given, current value remains unchanged.

Examples

```
// first subwindow
xsetech([0,0,1.0,0.5])
plot2d()
// then xsetech is used to set the second sub window
xsetech([0,0.5,1.0,0.5])
grayplot()
// get the graphic scales of first subwindow
xsetech([0,0,1.0,0.5])
[wrect,frect,logflag,arect]=xgetech();
// get the graphic scales of second subwindow
xsetech([0,0.5,1.0,0.5])
[wrect,frect,logflag,arect]=xgetech();
xbasc();
xset('default')
```

See Also

xsetech

Authors

J.Ph.C.;

Name

xgetmouse — get the mouse events and current position

```
[rep [,win]]=xgetmouse([sel])
```

Parameters

sel
: boolean vector `[getmotion, getrelease]`. default value is `[%t, %f]`

rep
: vector of size 3, [x,y,ibutton].

win
: number of the figure where the event occurred.

Description

If the mouse pointer is located in the current graphics window, `xgetmouse` returns in `rep` the current pointer position (x,y) and the value `ibutton`. The `ibutton` value indicates the event type:

ibutton==0
: Left mouse button has been pressed

ibutton==1
: Middle mouse button has been pressed

ibutton==2
: Right mouse button has been pressed

ibutton==3
: Left mouse button has been clicked

ibutton==4
: Middle mouse button has been clicked

ibutton==5
: Right mouse button has been clicked

ibutton==10
: Left mouse button has been double-clicked

ibutton==11
: Middle mouse button has been double-clicked

ibutton==12
: Right mouse button has been double-clicked

ibutton==-5
: Left mouse button has been released

ibutton==-4
: Middle mouse button has been released

ibutton==-3
: Right mouse button has been released

ibutton==-1
pointer has moved

ibutton > =32
key with ascii code ascii(ibutton) has been pressed

ibutton < =-32
key with ascii code ascii(-ibutton) has been released

ibutton > =1000+32
key with ascii code ascii(ibutton-1000) has been pressed while CTRL key pressed

ibutton==-1000
graphic window has been closed

WARNING: In previous versions of Scilab (<5.0), the user could give a flag to precise if the mouse click event queue had to be cleared when entering xgetmouse. This option is now obsolete and will be removed in Scilab 5.1.

Examples

```
//  rectangle selection
clf();  // erase/create window
a=gca();a.data_bounds=[0 0;100 100];//set user coordinates
xtitle(" drawing a rectangle ") //add a title
xselect(); //put the window on the top

[b,xc,yc]=xclick(); //get a point
xrect(xc,yc,0,0) //draw a rectangle entity
r=gce();// the handle of the rectangle
rep=[xc,yc,-1];first=%f;

while rep(3)==-1 do // mouse just moving ...
  rep=xgetmouse();
  xc1=rep(1);yc1=rep(2);
  ox=mini(xc,xc1);
  oy=maxi(yc,yc1);
  w=abs(xc-xc1);h=abs(yc-yc1);
  r.data=[ox,oy,w,h]; //change the retangle origin, width an height
  first=%f;
end
```

See Also

locate , xclick , seteventhandler

Authors

S. Steer

Name

xgraduate — axis graduation

```
[xi,xa,np1,np2,kMinr,kMaxr,ar]=xgraduate(xmi,xma)
```

Parameters

xmi,xma
: real scalars

xi, xa, kMinr, kMaxr, ar
: :real scalars

np1,np2
: integer

Description

`xgraduate` returns the axis graduations which are used by the plot routines (with pretty print flag enabled). It returns an interval `[xi,xa]` which contains the given interval `[xmi,xma]` and such that

`xi= kMinr*10^ar`, `xa=kMaxr*10^ar` and the interval can be divided into `np2` intervals and each interval is divided in `np1` sub-intervals.

Examples

```
[x1,xa,np1,np2,kMinr,kMaxr,ar]=xgraduate(-0.3,0.2)
```

See Also

graduate , plot2d

Authors

J.P.C ; ;

Name

xgrid — add a grid on a 2D plot

```
xgrid([style])
```

Parameters

style
: integer

Description

`xgrid` adds a grid on a 2D plot. `style` is the dash id or the color id to use for the grid plotting. Use `xset()` for the meaning of id.

Examples

```
x=[0:0.1:2*%pi]';
plot2d(sin(x))
xgrid(2)
```

See Also

xset , plot2d

Authors

J.Ph.C.

Name

xinfo — draw an info string in the message subwindow

```
xinfo(info)
```

Parameters

info
: string

Description

`xinfo` draws the string `info` in the message subwindow of the current graphics window.

Name

xlfont — load a font in the graphic context or query loaded font

```
xlfont(font-name)
xlfont(font-filename)
xlfont('reset')
xlfont(font-name,font-id)
xlfont(font-filename,font-id)
xlfont(font-name,font-id,bold)
xlfont(font-name,font-id,bold,italic)
fonts=xlfont('AVAILABLE_FONTS')
fonts=xlfont()
```

Parameters

font-name
: string, name of the font family.

font-filename
: string, filename of a true type font.

font-id
: integer >= 0 .

fonts
: a column vector of font names.

bold
: a boolean %t if bold

italic
: a boolean %t if italic

Description

Without any argument, `xlfont()` returns the list of currently loaded fonts.

xlfont('AVAILABLE_FONTS') returns list of fonts available on your system.

xlfont('reset') reset to initial index list of fonts.

With arguments, `xlfont` is used to load a new font at different sizes in the graphics context.

Default family fonts are "Monospaced" (0), "Symbol" (1), "Serif" (2), "Serif Italic" (3), "Serif Bold" (4), "Serif Bold Italic" (5), "SansSerif" (6), "SansSerif Italic" (7), "SansSerif Bold" (8), "SansSerif Bold Italic" (9). These default fonts are automaticaly loaded when needed and so `xlfont` is not really required for them. In fact `xlfont` is essentially useful to load a new font.

Examples

```
xlfont('reset');
xlfont()

// Caution : this example may not work if your system have not Monospaced font.
xlfont("Monospaced",10,%t,%t);
xstring(1,0,'A title');
```

```
figure_entity = gcf();
axes_entity = figure_entity.children;
title_entity = axes_entity.children;
title_entity.font_style = 10;

xlfont()

xlfont(SCI+'/thirdparty/fonts/scilabsymbols.ttf')
title_entity.font_style = 11; // use scilabsymbols.ttf font
title_entity.font_size  = 4; // size scilabsymbols.ttf font

xlfont()

xlfont('reset');
```

See Also

getfont

Authors

Allan CORNET

Name

xload — load a saved graphics

```
xload(file_name,[win_num])
```

Parameters

file_name
: string, name of the file.

win_num
: integer, the graphics window number. If not given, the current graphics window is used.

Description

`xload` reloads the graphics contained in the file `file_name` in the graphics window `win_num`.

Since Scilab 5.0, all uimenu or uicontrol handles are also loaded.

For files containing new graphics, the `load` function can be used instead of `xload`. `xload` does not restore the window number, the window size nor the window dimensions.

Examples

```
//new style
t=0:0.01:10;
subplot(211),plot2d(t,sin(t))
subplot(212),plot2d(t,sin(3*t))
save(TMPDIR+'/foo.scg',gcf())
clf()
load(TMPDIR+'/foo.scg')

a=gca();
curve=a.children.children; //handle on the curve
save(TMPDIR+'/foo.scg',curve)
delete(curve)
load(TMPDIR+'/foo.scg')
```

See Also

xsave , load , save

Authors

J.Ph.C.

Name

xname — change the name of the current graphics window

```
xname(name)
```

Parameters

name
: string, new name of the graphics window.

Description

`xname` changes the name of the current graphics window.

Authors

J.Ph.C.

Name

xnumb — draw numbers

```
xnumb(x,y,nums,[box,angle])
```

Parameters

x,y,nums
: vectors of same size.

box
: integer value.

angle
: optional vector of same size as `x`

Description

`xnumb` draws the value of `nums(i)` at position `x(i),y(i)` in the current scale. If `box` is 1, a box is drawn around the numbers. If `angle` is given, it gives the direction for string drawing.

Examples

```
plot2d([-100,500],[-100,600],[-1,-1],"022")
x=0:100:200;
xnumb(x,500*ones(x),[10,20,35],1)
```

See Also

xstring

Authors

J.Ph.C.

Name

xpause — suspend Scilab

```
xpause(microsecs)
```

Description

`xpause` suspends the current process for the number of microseconds specified by the argument. The actual suspension time may be longer because of other activities in the system, or because of the time spent in processing the call.

Authors

J.Ph.C.

Name

xpoly — draw a polyline or a polygon

```
xpoly(xv,yv [,dtype [,close]])
```

Parameters

xv,yv
: matrices of the same size (points of the polyline).

dtype
: string (drawing style). default value is "lines".

close
: integer. If close=1, the polyline is closed; default value is 0.

Description

`xpoly` draws a single polyline described by the vectors of coordinates `xv` and `yv`. If `xv` and `yv` are matrices they are considered as vectors by concatenating their columns. `dtype` can be `"lines"` for using the current line style or `"marks"` for using the current mark to draw the polyline.

Examples

```
x=sin(2*%pi*(0:4)/5);
y=cos(2*%pi*(0:4)/5);

plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("color",5)
xpoly(x,y,"lines",1) // by default closed

// News graphics only
e=gce(); // get the current entity (the last created: here the polyline)
e.closed = 'off' // the polyline is now open
```

See Also

xfpoly , xfpolys , xpolys

Authors

J.Ph.C.

Name

xpolys — draw a set of polylines or polygons

```
xpolys(xpols,ypols,[draw])
```

Parameters

xpols,ypols
: matrices of the same size (p,n) (points of the polylines).

draw
: vector of size n.

Description

`xpolys` draws a set of polylinse using marks or dashed lines. The coordinates of each polyline are stored in a column of `xpols` and `ypols`.

The style of polyline i is given by `draw(i)`:

- If `draw(i)` is negative, the mark with id `-draw(i)` is used to draw polyline i (marks are drawn using the current pattern). Use `xset()` to see the meaning of the ids.
- If `draw(i)` is strictly positive, the line style (or color) with id `draw(i)` is used to draw polyline i. Use `xset()` to see the meaning of the ids.

Examples

```
plot2d(0,0,-1,"012"," ",[0,0,1,1])
rand("uniform")
xset("color",3)
xpolys(rand(3,5),rand(3,5),[-1,-2,0,1,2])
xset("default")
```

See Also

xfpoly , xfpolys , xpoly

Authors

J.Ph.C.

Name

xrect — draw a rectangle

```
xrect(x,y,w,h)
xrect(rect) // rect =[x,y,w,h]
```

Parameters

x,y,w,h
: four real values defining the rectangle.

Description

`xrect` draws a rectangle defined by `[x,y,w,h]` (upper-left point, width, height) in user coordinates.

WARNING: please note that height is positive downwards.

Examples

```
plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("color",5)
xrect(-1,1,2,2)
xset("default")
```

See Also

xfrect , xrects

Authors

J.Ph.C.

Name

xrects — draw or fill a set of rectangles

```
xrects(rects,[fill])
```

Parameters

rects
: matrix of size (4,n).

fill
: vector of size n.

Description

`xrects` draws or fills a set of rectangles. Each column of `rects` describes a rectangle (upper-left point, width, height) in user coordinates: `rects=[x1 y1 w1 h1;x2 y2 w2 h2;...]'`.

`fill(i)` gives the pattern to use for filling or drawing rectangle i:

if `fill(i)<0`, rectangle `i` is drawn using the line style (or color) `-fill(i)`

if `fill(i)>0`, rectangle `i` is filled using the pattern (or color) `fill(i)`

if `fill(i)=0`, rectangle `i` is drawn using the current line style (or color).

WARNING: please note that height is positive downwards.

Examples

```
plot2d([-100,500],[-50,50],[-1,-1],"022")
cols=[-34,-33,-32,-20:5:20,32,33,34];
x=400*(0:14)/14; step=20;
rects=[x;10*ones(x);step*ones(x);30*ones(x)];
xrects(rects,cols)
xnumb(x,15*ones(x),cols)
```

See Also

xfrect , xrect

Authors

J.Ph.C.

Name

xrpoly — draw a regular polygon

```
xrpoly(orig,n,r,[theta])
```

Parameters

orig
: vector of size 2.

n
: integer, number of sides.

r
: real scalar.

theta
: real, angle in radian; 0 is the default value.

Description

`xrpoly` draws a regular polygon with `n` sides contained in the circle of diameter `r` and with the origin of the circle set at point `orig`. `theta` specifies a rotation angle in radian. This function uses the current graphics scales.

Examples

```
plot2d(0,0,-1,"012"," ",[0,0,10,10])
xrpoly([5,5],5,5)
```

See Also

xrect

Name

xsave — save graphics into a file

```
xsave(filename,[win_num])
```

Parameters

file_name
: string, name of the file.

win_num
: integer, the graphics window number. If not given, the current graphics window is used.

Description

`xsave` saves the graphics contained in the graphics window `win_num` in the binary file `file_name`. and can be reloaded with `xload`.

Since Scilab 5.0, all uimenu or uicontrol handles are also saved.

For new graphics `xsave(file_name,win_num)` use preferably `save(file_name,scf(win_num))`.

Examples

```
//new style
t=0:0.01:10;
subplot(211),plot2d(t,sin(t))
subplot(212),plot2d(t,sin(3*t))
save(TMPDIR+'/foo.scg',gcf())
clf()
load(TMPDIR+'/foo.scg')

a=gca();
curve=a.children.children; //handle on the curve
save(TMPDIR+'/foo.scg',curve)
delete(curve)
load(TMPDIR+'/foo.scg')
```

See Also

xload , save , load

Authors

J.Ph.C.

Name

xsegs — draw unconnected segments

```
xsegs(xv,yv,[style])
```

Parameters

xv,yv
: matrices of the same size.

style
: vector or scalar. If `style` is a positive scalar, it gives the color to use for all segments. If `style` is a negative scalar, then current color is used. If `style` is a vector, then `style(i)` gives the color to use for segment `i`.

Description

`xsegs` draws a set of unconnected segments given by `xv` and `yv`. If `xv` and `yv` are matrices they are considered as vectors by concatenating their columns. The coordinates of the two points defining a segment are given by two consecutive values of `xv` and `yv`:

`(xv(i),yv(i))-->(xv(i+1),yv(i+1))`.

For instance, using matrices of size (2,n), the segments can be defined by:

```
xv=[xi_1 xi_2 ...; xf_1 xf_2 ...] yv=[yi_1 yi_2 ...; yf_1 yf_2
    ...]
```

and the segments are `(xi_k,yi_k)-->(xf_k,yf_k)`.

Examples

```
x=2*%pi*(0:9)/10;
xv=[sin(x);9*sin(x)];
yv=[cos(x);9*cos(x)];
plot2d([-10,10],[-10,10],[-1,-1],"022")
xsegs(xv,yv,1:10)
```

Authors

J.Ph.C.

Name

xselect — raise the current graphics window

```
xselect()
```

Description

`xselect`raises the current graphics window. It creates the window if none exists.

Warning: This function is obsolete and will be removed in Scilab 5.1. It has been replaced by the show_window function.

See Also

show_window

Authors

J.Ph.C.

Jean-Baptiste Silvy

Name

xset — set values of the graphics context. This function is obsolete.

```
xset(choice-name,x1,x2,x3,x4,x5)
xset()
```

Parameters

choice-name
: string

x1,...,x5
: depending on `choice-name`

Description

Warning this function is obsolete. Use the Scilab graphic objetcs representation instead (see the set and get functions as well as the graphics_entities help page).

`xset` is used to set default values of the current window graphic context.

When called no argument, a choice menu is created showing the current values and changes can be performed through toggle buttons.

Use `xset()` to display or set the current color, mark and fonts used.

xset("alufunction",number)
: Used to set the logical function for drawing. The logical function used is set by `x1`. Usual values are: 3 for copying (default), 6 for animation and 0 for clearing. See alufunctions for more details.

xset("auto clear","on"|"off")
: Switch "on" or "off" the auto clear mode for graphics. When the auto clear mode is "on", successive plots are not superposed, ie an `xbasc()` operation (the graphics window is cleared and the associated recorded graphics is erased) is performed before each high level graphics function. Default value is "off".

xset("background",color)
: Set the background color of the current Axes object. The `color` argument is the colormap index of the color to use.

xset("clipping",x,y,w,h)
: Set the clipping zone (the zone of the graphics window where plots can be drawn) to the rectangle (x,y,w,h) (Upper-Left point Width Height). This function uses the current coordinates of the plot.

xset("color",value)
: Set the default color for filling, line or text drawing functions. `value` is an integer projected in the interval [0,whiteid]. 0 is used for black filling and whiteid for white. The value of whiteid can be obtained with `xget("white")`.

xset("colormap",cmap)
: Set the colormap as a m x 3 matrix. m is the number of colors. Color number i is given as a 3-uple cmap(i,1), cmap(i,2), cmap(i,3) corresponding respectively to red, green and blue intensity between 0 and 1.

xset("dashes",i)
: In black and white mode (`xset("use color",0)`), set the dash style to style i (0 for solid line). In color mode (`xset("use color",1 )`) this is used to set line, mark and text color.

This keyword is obsolete, please use `xset('color',i)` or `xset('line style',i)` instead.

xset("default")
Reset the graphics context to default values.

xset("font",fontid,fontsize) : Set the current font and its current size. Note that `fontsize` applies to all fonts not only `fontid` .

xset("font size",fontsize)
Set the fonts size.

xset("foreground",color)
Set the foreground color of the current Axes object. The `color` argument is the colormap index of the color to use.

xset("fpf",string)
Set the floating point format for number display in contour functions. `string` is a string giving the format in C format syntax (for example `string="%.3f"`). Use `string=""` to switch back to default format.

xset("hidden3d",colorid) : Set the color number for backward facing faces in `plot3d`. `colorid=0` zero suppress the drawing of backward facing faces of 3d objects. This is technically called 'culling' and speeds up the rendering of closed surfaces.

xset("line mode",type)
This function is used to set the line drawing mode. Absolute mode is set with type=1 and relative mode with type=0. (Warning: the mode type=0 has bugs)

xset("line style",value)
Set the current line style (1: solid, >1 for dashed lines).

xset("mark",markid,marksize)
Set the current mark and the current mark size. Use `xset()` to see the marks. Note that `marksize` applies to all marks not only `markid` .

xset("mark size",marksize)
Set the marks size.

xset("pattern",value)
Set the current pattern for filling functions. value is an integer projected in the interval [0,whiteid]. 0 is used for black filling and whiteid for white. The value of whiteid can be obtained with `xget("white")`. "pattern" is equivalent to "color".

xset("pixmap",flag)
If flag=0 the graphics are directly displayed on the screen. If flag=1 the graphics are done on a pixmap and are sent to the graphics window with the command `xset("wshow")`. The pixmap is cleared with the command `xset("wwpc")`. Note that the usual command `xbasc()` also clears the pixmap.

xset("thickness",value)
Set the thickness of lines in pixel (0 and 1 have the same meaning: 1 pixel thick).

xset("use color",flag)
If flag=1 then `xset("pattern",.)` or `xset("dashes",.)` will be used so as to change the default color for drawing or for filling patterns. If flag=0 then we switch back to the gray and dashes mode.

xset("viewport",x,y)
Set the position of the panner.

xset("wdim",width,height)
Set the width and the height of the current graphics window. This option is not used by the postscript driver.

xset("wpdim",width,height)
Sets the width and the height of the current physical graphic window (which can be different from the actual size in mode wresize 1). This option is not used by the postscript driver.

xset("window",window-number)
Set the current window to the window `window-number` and creates the window if it does not exist.

xset("wpos",x,y)
Set the position of the upper left point of the graphics window.

xset("wresize",flag)
If flag=1 then the graphic is automatically resized to fill the graphics window.

```
xdel();xset("wresize",1);plot2d();xset("wdim",1000,500)
```

If flag=0 the scale of the graphic is left unchanged when the graphics window is resized. Top left panner or keyboard arrows may be used to scroll over the graphic.

```
xdel();plot2d();xset("wresize",0);xset("wdim",1000,500)
```

xset("wshow")
See `xset("pixmap",1)` above.

xset("wwpc")
See `xset("pixmap",1)` above.

See Also

xget , getcolor, getsymbol, ged, set, graphics_entities

Authors

J.Ph.C.

Name

xsetech — set the sub-window of a graphics window for plotting

```
xsetech(wrect,[frect,logflag])
xsetech(wrect=[...],frect=[..],logflag="..", arect=[...])
xsetech()
```

Parameters

wrect
: vector of size 4, defining the sub-window to use.

frect
: vector of size 4.

logflag
: string of size 2 "xy", where x and y can be "n" or "l". "n" stands for normal and "l" stands for logscale. x stands for the x-axis and y stands for the y-axis.

arect
: vector of size 4.

Description

`xsetech` is mainly used to set the sub-window of the graphics window which will be used for plotting. The sub-window is specified with the parameter `wrect=[x,y,w,h]` (upper-left point, width, height). The values in `wrect` are specified using proportion of the width or height of the current graphic window. For instance `wrect=[0,0,1,1]` means that the whole graphics window will be used, and `wrect=[0.5,0,0.5,1]` means that the graphics region will be the right half of the graphics window.

`xsetech` also set the current graphics scales for 2D plotting and can be used in conjunction with graphics routines which request the current graphics scale (for instance `strf="x0z"` or`frameflag=0` in `plot2d`).

`frect=[xmin,ymin,xmax,ymax]` is used to set the graphics scale and is just like the `rect` argument of `plot2d`. If `frect` is not given the current value of the graphic scale remains unchanged. the default value of `rect` is `[0,0,1,1]` (at window creation, when switching back to default value with `xset('default')` or when clearing graphic recorded events `xbasc()`).

`arect=[x_left,  x_right,y_up,y_down]` is used to set the graphic frame inside the subwindow. The graphic frame is specified (like `wrect`) using proportion of the width or height of the current graphic subwindow. Default value is `1/8*[1,1,1,1]`. If `arect` is not given, current value remains unchanged.

Examples

```
// To get a graphical explanation of xsetech parameters enter:
exec('SCI/modules/graphics/demos/xsetechfig.sce');


// Here xsetech is used to split the graphics window in two parts
// first xsetech is used to set the first sub-window
// and the graphics scale
xsetech([0,0,1.0,0.5],[-5,-3,5,3])
```

```
// we call plot2d with the "001" option to use the graphics scale
// set by xsetech
plot2d([1:10]',[1:10]',1,"001"," ")
// then xsetech is used to set the second sub-window
xsetech([0,0.5,1.0,0.5])
// the graphics scale is set by xsetech to [0,0,1,1] by default
// and we change it with the use of the rect argument in plot2d
plot2d([1:10]',[1:10]',1,"011"," ",[-6,-6,6,6])
// Four plots on a single graphics window
xbasc()
xset("font",2,0)
xsetech([0,0,0.5,0.5]); plot3d()
xsetech([0.5,0,0.5,0.5]); plot2d()
xsetech([0.5,0.5,0.5,0.5]); grayplot()
xsetech([0,0.5,0.5,0.5]); histplot()
// back to default values for the sub-window
xsetech([0,0,1,1])
// One plot with changed arect
xbasc()
xset("default")
xsetech(arect=[0,0,0,0])
x=1:0.1:10;plot2d(x',sin(x)')
xbasc()
xsetech(arect=[1/8,1/8,1/16,1/4])
x=1:0.1:10;plot2d(x',sin(x)')
xbasc()
xset("default")
```

See Also

xgetech , subplot , isoview , square

Authors

J.Ph.C.

Name

xsetm — dialog to set values of the graphics context. **Obsolete function.**

```
xsetm()
```

Description

This function as well as the xset one was strongly linked with the old graphic mode which is no more available. The current graphic is much more flexible with respect to parameter setting (see the set and get functions as well as the graphics_entities help page). It is possible to start a more convenient property editor using ged.

See Also

xset, ged, set, graphics_entities

Authors

J.Ph.C. ENPC

Name

xstring — draw strings

```
xstring(x,y,str,[angle,box])
```

Parameters

x,y
: real scalars, coordinates of the lower-left point of the strings.

str
: matrix of strings.

angle
: real, clockwise angle in degree; default is 0.

box
: integer, default is 0.

Description

`xstring` draws the matrix of strings `str` at location `x,y` (lower-left point) in the current graphic scale: each row of the matrix stands for a line of text and row elements stand for words separated by a white space. If `angle` is given, it gives the slope in degree used for drawing the strings. If `box` is 1 and `angle` is 0, a box is drawn around the strings.

Examples

```
plot2d([0;1],[0;1],0)
xstring(0.5,0.5,["Scilab" "is"; "not" "esilaB"])
//Other example
alphabet=["a" "b" "c" "d" "e" "f" "g" ..
          "h" "i" "j" "k" "l" "m" "n" ..
          "o" "p" "q" "r" "s" "t" "u" ..
          "v" "w" "x" "y" "z"];
xbasc()
plot2d([0;1],[0;2],0)
xstring(0.1,1.8,alphabet)     // alphabet
xstring(0.1,1.6,alphabet,0,1) // alphabet in a box
xstring(0.1,1.4,alphabet,20)  // angle
xset("font",1,1)              // use symbol fonts
xstring(0.1,0.1,alphabet)
xset("font",1,3)              // change size font
xstring(0.1,0.3,alphabet)
xset("font",1,24); xstring(0.1,0.6,"a") //big alpha
xset("default")
```

See Also

titlepage , xnumb , xstringb , xstringl , xtitle

Authors

J.Ph.C.

Name

xstringb — draw strings into a box

```
xstringb(x,y,str,w,h,[option])
```

Parameters

x,y,w,h
: vector of 4 real scalars defining the box.

str
: matrix of strings.

option
: string.

Description

`xstringb` draws the matrix of strings `str` centered inside the rectangle `rect=[x,y,w,h]` (lower-left point, width, height) in user coordinates.

If `option` is given with the value `"fill"`, the character size is computed so as to fill as much as possible in the rectangle.

Enter the command `xstringb()` to see a demo.

Examples

```
str=["Scilab" "is";"not" "elisaB"];
plot2d(0,0,[-1,1],"010"," ",[0,0,1,1]);
r=[0,0,1,0.5];
xstringb(r(1),r(2),str,r(3),r(4),"fill");
xrect(r(1),r(2)+r(4),r(3),r(4))
r=[r(1),r(2)+r(4)+0.01,r(3),r(4)/2];
xrect(r(1),r(2)+r(4),r(3),r(4))
xstringb(r(1),r(2),str,r(3),r(4),"fill");
r=[r(1),r(2)+r(4)+0.01,r(3),r(4)/2];
xrect(r(1),r(2)+r(4),r(3),r(4))
xstringb(r(1),r(2),str,r(3),r(4),"fill");
```

See Also

titlepage , xstring , xstringl , xtitle

Authors

J.Ph.C.

Name

xstringl — compute a box which surrounds strings

```
rect=xstringl(x,y,str,[fontId,fontSize])
```

Parameters

rect
: vector of 4 real scalars defining the box.

x,y
: real scalars, coordinates of the lower-left point of the strings.

str
: matrix of strings.

fontId
: an integer specifying the font type.

fontSize
: an integer specifying the font size.

Description

`xstringl` returns in `rect=[x,y,w,h]` (upper-left point, width, height) the size of a rectangle in the current graphic scale which would surround the strings `str` drawn at location `x,y` (lower-left point).

The result can be approximative when using the Postscript driver.

Examples

```
plot2d([0;1],[0;1],0)
str=["Scilab" "is";"not" "elisaB"];
r=xstringl(0.5,0.5,str)
xrects([r(1) r(2)+r(4) r(3) r(4)]')
xstring(r(1),r(2),str)


plot2d([0;1],[0;1],0)
str=["Scilab" "n''est ";"pas" "Matlab"];
r2 = xstringl(0.5,0.5,str,2,5)
xrects([r2(1) r2(2)+r2(4) r2(3) r2(4)]')
xstring(r2(1),r2(2),str)

txt2=gce();
txt2.font_size = 5;
txt2.font_style = 2;
```

See Also

titlepage , xstring , xstringl , xtitle , stringbox

Authors

J.Ph.C.

Name

xtitle — add titles on a graphics window

```
xtitle(title,[x_label,[y_label,[z_label]]],<opts_args>)
```

Parameters

title,x_label,y_label, z_label
: matrices of strings.

<opt_args>
: a sequence of statements `key1=value1, key2=value2`, ... where keys may be `boxed` (see below). In this case, the order has no special meaning.

boxed
: an integer value. If it is 1, a box is drawn around each title.

Description

`xtitle` add titles on a 2D or 3D plot. `title` is the general title and **x_label**, **y_label** and **z_label** are the titles on the three axis. If the arguments are matrices, each line of the matrices is displayed on a different line.

Enter the command `xtitle()` to see a demo.

Examples

```
// draw a surface
plot3d() ;
// puts the titles
xtitle( 'My surface is blue', 'X axis', 'Y axis', 'Z axis' ) ;
// draw a box around the titles
xtitle( 'My surface is blue', 'X axis', 'Y axis', 'Z axis', boxed = 1 ) ;
```

See Also

titlepage label_properties

Authors

J.Ph.C.

Name

zoom_rect — zoom a selection of the current graphic figure

```
zoom_rect()
zoom_rect(rect)
zooom_rect(h)
zoom_rect(h,rect)
```

Parameters

rect
: Vector of size 4 `[xmin,ymin,xmax,ymax]` give the rectangle to be zoomed.

h
: Graphic handle of type Figure or Axes. Specify on which Axes the zoom will apply.

Description

`zoom_rect` funtion is used to perform a zoom inside a set of Axes Objects.

The `h` input argument specifies on which Axes the zoom will apply. If `h` is a Figure handle then the zoom will apply on its Axes children. If `h` is a Axes handle then the zoom will only apply to this handle. If `h` is not specified, then the zoom is performed on the current Figure.

If `rect` input argument is specified then the zoomed Axes `zoom_box` property is modified by the argument (see axes_properties). Its bounds along X and Y axis are replaced by `rect`. If `rect` is not specified zoom_rect is an interactive zoom. User is required to select a rectangle using the mouse. The new `zoom_box` property of zoomed axes are then computed by finding the intersections of the rectangle with their axes boxe.

Examples

```
clf()
x=0:0.01:6*%pi;
plot2d(x,sin(x^2))
zoom_rect([16,-1,18,1])
//more zoom
zoom_rect([16,0,16.2,1])
//back to the original
unzoom()
// zooming using axes_properties
a=gca();
a.zoom_box=[16,0,16.2,1];
a.zoom_box=[];

//zooming subplots accordingly
clf()
x=0:0.01:6*%pi;
subplot(211)
plot2d(x,cos(x))
subplot(212)
plot2d(x,cos(2*x))
rect=[3 -2 7 10]; //a rectangle specified in the current axes (last one) co
zoom_rect(rect)
```

```
unzoom()
//set the global underlying axes as current
f=gcf();set('current_axes',f.children($))
rect=[0.4 0 0.6 1] //a rectangle specified in ratio of the window size
zoom_rect(rect)
rect=[0.4 0.2 0.6 0.8]; //a rectangle specified in ratio of the window size
zoom_rect(rect)

// interactive zoom on current figure
zoom_rect();
// or
zoom_rect(gcf());
```

See Also

unzoom , axes_properties

Authors

Serge Steer INRIA

Jean-Baptiste Silvy INRIA

History manager

Name

addhistory — add lines to current history.

```
addhistory(string)
addhistory(string_matrix)
```

Parameters

string
: a string

string_matrix
: a string matrix

Description

add lines to current history.

Examples

```
addhistory('hello')
addhistory(['hello','Scilab'])
```

Authors

A.C

Name

displayhistory — displays current scilab history

```
displayhistory()
```

Description

displays current scilab history.

See Also

gethistory

Authors

A.C

Name

gethistory — returns current scilab history in a string matrix

```
matstr=gethistory()
line=gethistory(N)
```

Parameters

matstr
: a string matrix

N
: Nth line in scilab's history

line
: a string

Description

returns current scilab history in a string matrix.

See Also

savehistory , loadhistory , resethistory

Authors

A.C

Name

gethistoryfile — get filename used for scilab's history

```
filename = gethistoryfile()
```

Parameters

filename
: file name used for history

Description

get filename for scilab's history

Examples

```
gethistoryfile()
 gethistoryfile()
```

Authors

A.C

Name

historymanager — enable or disable history manager

```
state1=historymanager(state2)
state1=historymanager()
```

Parameters

state1
: returns history manager state 'on' or 'off'

state2
: 'on' or 'off' set history manager's state

Description

enable or disable history manager.

Examples

```
displayhistory()
backupstate=historymanager()
historymanager('off')
displayhistory()
historymanager('on')
loadhistory()
displayhistory()
historymanager(backupstate)
```

Authors

A.C

Name

historysize — get number of lines in history

```
nb=historysize()
```

Parameters

nb
: number of lines in history.

Description

get number of lines in history.

Examples

```
historysize()
```

Authors

A.C

Name

loadhistory — load a history file

```
loadhistory()
loadhistory(f)
```

Parameters

f
: file pathname

Description

load a history file

by default, history filename is SCIHOME+'/.history.scilab'

Examples

```
loadhistory(SCI+'/session.scilab')
```

See Also

savehistory , resethistory , gethistory

Authors

A.C

Name

removelinehistory — remove the Nth line in history.

```
removelinehistory(N)
```

Parameters

N
: a line number

Description

remove the Nth line in history.

Examples

```
displayhistory()
removelinehistory(historysize()-2)
displayhistory()
```

Authors

A.C

Name

resethistory — Deletes all entries in the scilab history.

```
resethistory()
```

Description

Deletes all entries in the current scilab history.

See Also

savehistory , loadhistory

Authors

A.C

Name

saveafterncommands — Save the history file after n statements are added to the file.

```
saveafterncommands(n)
v = saveafterncommands()
```

Parameters

n
: a integer, n statements

v
: current value

Description

Save the history file after n statements are added to the file.

For example, when you select the option and set n to 5, after every 5 statements are added, the history file is automatically saved.

Use this option if you don't want to risk losing entries to the saved history because of an abnormal termination, such as a power failure.

saveafterncommands() returns current value.

0 is default value.

Examples

```
saveafterncommands(3)
```

Authors

A.C

Name

saveconsecutivecommands — Save consecutive duplicate commands.

```
saveconsecutivecommands(boolean_in)
boolean_out = saveconsecutivecommands()
```

Parameters

boolean_in
: a boolean (%t or %f)

boolean_out
: current value

Description

Save consecutive duplicate commands.

saveconsecutivecommands(%t) if you want consecutive executions of the same statement to be saved to the history file.

Examples

```
saveconsecutivecommands()
saveconsecutivecommands(%t)
1
1
2
saveconsecutivecommands(%f)
1
1
2
```

Authors

A.C

Name

savehistory — save the current history in a file

```
savehistory()
savehistory(f)
```

Parameters

f
: file pathname

Description

save the current history in a file.

by default, history filename is SCIHOME+'/.history.scilab'

Examples

```
savehistory(SCI+'/session.scilab')
```

See Also

loadhistory , resethistory , gethistory

Authors

A.C

Name

sethistoryfile — set filename for scilab history

```
sethistoryfile(filename)
sethistoryfile()
```

Parameters

filename
: filename for history

Description

set filename for scilab history.

sethistoryfile() without parameters will use the default filename (SCIHOME/history.scilab)

Examples

```
gethistoryfile()
 sethistoryfile(gethistoryfile())
```

Authors

A.C

Input/Output functions

Name

deff — on-line definition of function

```
deff('[s1,s2,...]=newfunction(e1,e2,....)',text [,opt])
```

Parameters

e1,e2,...,
: input variables.

s1,s2,...,
: output variables.

text
: matrix of character strings

opt
: optional character string

  'c'
  : function is "compiled" to be more efficient (default)

  'p'
  : function is "compiled" and prepared for profiling (see profile)

  'n'
  : function is not "compiled"

Description

`deff` can be used to define functions from sequences of instructions written in text strings. The resulting function object has the same proprerties of any other function defined in a text file and loaded with `getf` or `exec`.

Quotes in the instructions (delimiting strings or meaning matrix transposition) have to be doubled to be interpreted correctly (see help quote). This can make writing up a little awkward. An option in such cases is to define functions in files as usual, to load them into Scilab by `getf` (with the `'n'` option) and to use `sci2exp` to get a printout of corresponding `deff` instructions.

Examples

```
deff('[x]=myplus(y,z)','x=y+z')
//
deff('[x]=mymacro(y,z)',['a=3*y+1'; 'x=a*z+y'])
```

See Also

getf , comp , exec , function , profile

Name

diary — diary of session

```
diary(f)
diary(0)
```

Parameters

f
: a character string, give the file name path.

Description

`diary(f)` opens the file which path is given in `f` and register all subsequent Scilab console inputs and outputs.

The diary is terminated and the correspondig file is closed either by a call to `diary(0)` or by opening a new diary file.

Name

disp — displays variables

```
disp(x1,[x2,...xn])
```

Description

displays `xi` with the current format. `xi`'s are arbitrary objects (matrices of constants, strings, functions, lists, ...)

Display of objects defined by `tlist` may be overloaded by the definition of a function. This function must have no output argument a single input argument ant it's name is formed as follow `%<tlist_type>_p` where `%<tlist_type>` stands for the first entry of the tlist type component.

The lines function may be used to control the ouput.

Examples

```
disp([1 2],3)
deff('[]=%t_p(l)','disp(l(3),l(2))')
disp(tlist('t',1,2))
```

See Also

lines , write , read , print , string , tlist

Name

execstr — execute Scilab code in strings

```
execstr(instr)
ierr=execstr(instr,'errcatch' [,msg])
```

Parameters

instr
: vector of character strings, Scilab instruction to be executed.

ierr
: integer, 0 or error number.

msg
: character string with values `'m'` or `'n'`. Default value is `'n'`.

Description

Executes the Scilab instructions given in argument `instr`.

Note that instr should not make use of continuation marks (..)

If the 'errcatch' flag is not present, error handling works as usual.

If the 'errcatch' flag is set, and an error is encountered while executing the instructions defined in `instr`, `execstr` issues no error message, but aborts execution of the `instr` instructions (at the point where the error occurred), and resumes with `ierr` equal to the error number. In this case the display of the error message is controlled by the `msg` option:

"m"
: error message is displayed and recorded.

"n"
: no error message is displayed, but the error message is recorded (see `lasterror`). This is the default.

ierr=execstr(instr,'errcatch') can handle syntactical errors. This is useful for evalution of instruction obtained by a query to the user.

Examples

```
execstr('a=1') // sets a=1.
execstr('1+1') // does nothing (while evstr('1+1') returns 2)

execstr(['if %t then';
         '  a=1';
         '  b=a+1';
         'else'
         ' b=0'
         'end'])

execstr('a=zzzzzzz','errcatch')
execstr('a=zzzzzzz','errcatch','m')
```

```
//syntax errors
execstr('a=1?02','errcatch')
lasterror(%t)

execstr('a=[1 2 3)','errcatch')
lasterror(%t)

// variable1 does not exist
if execstr('variable1;','errcatch')<>0 then disp("Trigger an error"),end

// variable2 exists ... no error is triggered by execstr
variable2=[2,3];
if execstr('variable2;','errcatch')<>0 then
   disp("Trigger an error");
else
   disp("execstr is happy");
end
```

See Also

evstr , lasterror , error , try

Name

file — file management

```
[unit [,err]]=file('open', file-name [,status] [,access [,recl]] [,format])
file(action,unit)
[units [,typ [,nams [,mod [,swap]]]]] = file([unit])
```

Parameters

file-name
: string, file name of the file to be opened

status
: string, The status of the file to be opened

  "new"
  : file must not exist new file (default)

  "old"
  : file must already exists.

  "unknown"
  : unknown status

  "scratch"
  : file is to be deleted at end of session

access
: string, The type of access to the file

  "sequential"
  : sequential access (default)

  "direct"
  : direct access.

format
: string,

  "formatted"
  : for a formatted file (default)

  "unformatted"
  : binary record.

recl
: integer,is the size of records in bytes when `access="direct"`

unit
: integer, logical unit descriptor of the opened file

units
: integer vector, logical unit descriptor of the opened files. Units 5 and 6 are reserved by the system for input and output devices.

typs
: Character string vector, type (C or Fortran) of opened files.

nams
: Character string vector, pathnames of opened files.

mod
: file opening mode. Formed by three digits abc

  Fortran files

  a
  : 0 stands for formatted and 1 for unformatted (binary)

  b
  : 0 stands for sequential acces and 1 for direct access

  c
  : 0 stands for "new", 1 for "old", 2 for "scratch" and 3 for "unknown"

  C files

  a
  : is 1 if file has been opened with a "b" (binary) mode

  b
  : is 1 if file has been opened with a "+" (updating) mode

  c
  : 1 stands for "r" (read), 2 stands for "w" (write) and 3 for "a" (append)

swap
: automatic swap switch. swap=1 if automatic swap is on. swap is always 0 for Fortran files.

err
: integer, error message number (see error), if open fails. If err is omitted an error message is issued.

action
: is one of the following strings:

  "close"
  : closes the file(s) given by the logical unit descriptors given in `units`

  "rewind"
  : puts the pointer at beginning of file

  "backspace"
  : puts the pointer at beginning of last record.

  "last"
  : puts the pointer after last record.

Description

selects a logical unit `unit` and manages the file `file-name`.

`[unit [,err]]=file('open', file-name [,status] [,access [,recl]] [,format])` allows to open a file with specified properties and to get the associated unit number `unit`. This unit number may be used for further actions on this file or as file descriptor in `read`, `write`, `readb`, `writb`,`save`, `load` function calls.

`file(action,unit)` allows to close the file , or move the current file pointer .

`file()` returns the logical unit descriptors of the opened files. So `file('close',file() )` closes all user opened files (C or Fortran type).

Examples

```
u=file('open',TMPDIR+'/foo','unknown')
for k=1:4
  a=rand(1,4)
  write(u,a)
end
file('rewind',u)
x=read(u,2,4)
file('close',u)
//
u1=file('open',TMPDIR+'/foo','unknown')
u2=mopen(TMPDIR+'/foo1','wb')
[units,typs,nams]=file()
file('close',u1);
mclose(u2);
```

See Also

save , load , write , read , writb , readb , uigetfile , mopen , mclose

Name

fileinfo — Provides information about a file

```
[x,ierr]=fileinfo(file)
```

Parameters

file
: a character string, the file pathname

x
: an integer vector of size 13 containing information or an empty matrix if file does not exist.

ierr
: error indicator, 0, if no error has occured

Description

x=fileinfo(file) returns

x(1)
: The file size

x(2)
: The file mode (decimal value).

x(3)
: The user id

x(4)
: The group id

x(5)
: The device number

x(6)
: The date of last modification

x(7)
: The date of last change

x(8)
: The date of last access

x(9)
: The device type (if inode device)

x(10)
: The blocksize for filesystem I/O (always 0 on Windows)

x(11)
: The number of blocks allocated (always 0 on Windows)

x(12)
: The inode

x(13)
: The number of hard links

Reference

This function is an interface to the C function stat.

Permissions are typically specified as octal numbers : dec2oct(x(2)) to convert

Numeric mode is from one to four octal digits (0-7), derived by adding up the bits with values 4, 2, and 1. Any omitted digits are assumed to be leading zeros. The first digit selects the set user ID (4) and set group ID (2) and sticky (1) attributes. The second digit selects permissions for the user who owns the file: read (4), write (2), and execute (1); the third selects permissions for other users in the file's group, with the same values; and the fourth for other users not in the file's group, with the same values.

Examples

```
w = fileinfo(SCI+'/etc/scilab.start')
// file permission
dec2oct(w(2))
// file date
getdate(w(6))

// Checks write permission on a file
w = fileinfo(SCI+'/etc/scilab.start')

S_IWRITE = 128; // mask write permission
S_IEXEC = 64; // mask exec permission
S_IREAD = 256; // mask read permission
S_IFCHR = 8192; // mask directory permission

if ( bitand( w(2), S_IWRITE ) <> 0) then
 disp('WRITE PERMISSION on this file.');
else
 disp('NO WRITE PERMISSION on this file.');
end
```

See Also

getdate, file, dispfiles, newest, isdir

Authors

S. Steer INRIA

A.C

Name

get_absolute_file_path — Given an absolute pathname of a file opened in scilab.

```
pathname = get_absolute_file_path(filename)
```

Parameters

filename
: A character string : filename

pathname
: A character string : the absolute pathname

Description

Given the absolute pathname of a file already opened in scilab.

get_absolute_file_path searchs, in scilab's internal list of files currently opened, filename and returns its path.

"get_absolute_file_path" seek, in the internal list of the files of scilab currently opened, " filename" and it gives his path.

if file not opened , it will return a error.

WARNING : in previous version (scilab 5.0.x) current directory was returned if file was not found.

This function can be used to find from where (path) is executed a scilab script.

Examples

```
// exec this script

a=mopen(TMPDIR+'test.sce','wt');
disp(get_absolute_file_path('test.sce'));
mclose(a);
```

See Also

getshortpathname, getlongpathname, getcwd

Authors

Allan CORNET

Name

getenv — get the value of an environment variable

```
env=getenv(str [, rep] )
```

Parameters

str
: character string specifying environment variable name rep : an optional character string. When this optional value is used, the function `getenv` returns the value `rep` when the environment variable `str` is not found.

env
: character string which contain the environment variable value

Description

Return the value of an environment variable if it exists.

Examples

```
getenv('SCI')
getenv('FOO','foo')
```

Name

getf — defining a function from a file

```
getf(file-name [,opt])
```

Parameters

filename
: Scilab string.

opt
: optional character string

  "c"
  : loaded functions are "compiled" to be more efficient (default)

  "n"
  : loaded functions are not "compiled"

  "p"
  : loaded functions are "compiled" and prepared for profiling (see profile)

Description

loads one or several functions (see `functions`) defined in the file `'file-name'`. The string `opt='n'` means that the functions are not compiled (pre-interpreted) when loaded. This can be useful for some debugging purpose (see `comp`). By default, functions are compiled when loaded (i.e. `opt='c'` is used).

In the file a function must begin by a "syntax definition" line as follows:

```
function [y1,...,yn]=foo(x1,...,xm)
```

The following lines contain a sequence of scilab instructions.

The "syntax definition" line gives the "full" calling syntax of this function. The `yi` are output variables calculated as functions of input variables `xi` and variables existing in Scilab when the function is executed. Shorter input or output argument list may be used.

Many functions may be written in the same file. A function is terminated by an `endfunction` keyword. For compatibility with previous versions a function may also be terminated by the following `function` keyword or the `EOF` mark. For that reason it is not possible to load function containing nested function definition using the `getf` function.

`getf` is an obsolete way for loading functions into scilab from a file. It is replaced by the function exec. Note that functions in a file should be terminated by an `endfunction` keyword. The `exec` function supposes `opt=='c'`.

To prepare a function for profiling please use the add_profiling function.

Examples

```
```

```
getf('SCI/modules/graphics/macros/plot.sci')

getf SCI/modules/graphics/macros/plot.sci
```

See Also

functions, function, genlib, getd, exec, edit, comp, add_profiling

Name

getio — get Scilab input/output logical units

```
ios=getio()
```

Parameters

ios
: a vector [rio rte wio wte]

  rio
  : current logical unit for reading instructions

  rte
  : logical unit assigned for input in main scilab window

  wio
  : logical unit relative to the diary file if any. wio=0 stands for no diary file opened

  wte
  : logical unit assigned for output in main scilab window

Description

`getio` returns logical units assigned for main scilab input and output

See Also

file , exec

Name

getpid — get Scilab process identificator

```
id=getpid()
```

Description

Return an the scilab process identificator integer

Examples

```
d='SD_'+string(getpid())+'_'
```

Name

getrelativefilename — Given an absolute directory and an absolute filename, returns a relative file name.

```
rel_file = getrelativefilename(abs_dir,abs_file)
```

Parameters

abs_dir
: A character string : the absolute directory

abs_file
: A character string : the absolute filename

rel_file
: A character string : relative filename

Description

Given an absolute directory and an absolute filename, returns a relative file name.

For example, if the current directory is C:\scilab\bin and the filename C:\scilab\modules\helptools \readme.txt is given, getrelativefilename will return ..\modules\helptools\readme.txt.

Examples

```
if MSDOS then
 getrelativefilename('C:\program file\scilab-4.0\bin','C:\program file\scilab-4
 getrelativefilename('C:\program file\scilab-4.0\bin\','C:\program file\scilab-4
 getrelativefilename(SCI+'\bin',SCI+'\modules\helptools\help.dtd')
 getrelativefilename(WSCI+'\bin',WSCI+'\modules\helptools\help.dtd')
 getrelativefilename(getcwd(),WSCI+'\bin\Wscilex')
else
 getrelativefilename('/usr/local/scilab-4.0/bin','/usr/local/scilab-4.0/modules
 getrelativefilename('/usr/local/scilab-4.0/bin/','/usr/local/scilab-4.0/module
 getrelativefilename(SCI+'/bin',SCI+'/modules/helptools/help.dtd')
 getrelativefilename(getcwd(),SCI+'/bin/scilex')
end
```

See Also

getshortpathname , getlongpathname , getcwd

Authors

Pierre MARECHAL

Name

getscilabkeywords — returns a list with all scilab keywords.

```
list_keywords=getscilabkeywords()
```

Parameters

list_keywords
: a list

Description

list_keywords(1) : primitives

list_keywords(2) : commands

list_keywords(3) : predef variables

list_keywords(4) :scilab functions

list_keywords(5) :scicos functions

Authors

A.C, adapted from Enrico Segre's code in Scipad

Name

halt — stop execution

```
halt()
halt('a message')
```

Description

stops execution until something is entered in the keyboard.

Examples

```
halt('Press a key')

halt()
```

See Also

pause , return , exec

Name

host — Unix or DOS command execution

```
stat=host(command-name)
```

Parameters

command-name
: A character string containing Unix sh instruction

stat
: An integer flag

Description

Sends a string `command-name` to Unix for execution by the command interpreter (sh under Unix, or command.com under DOS). Standard output and standard errors of the shell command are written in the calling shell. `stat` gives -1 if host can't be called (Not enough system memory available) or the command interpreter return code.

Examples

```
//create a getdir function based on host
function wd=getdir()
  if MSDOS then
    host('cd>'+TMPDIR+'\path');
  else
    host('pwd>'+TMPDIR+'/path');
  end
  wd=read(TMPDIR+'/path',1,1,'(a)')
endfunction
//call it
wd=getdir()
```

See Also

edit , manedit , unix_g , unix_s , unix_w , unix_x

Name

input — prompt for user input

```
x = input(message [, "string"])
```

Parameters

message
: character string

"string"
: the character string `"string"` (may be abbreviated to `"s"`)

x
: real number (or character string if `"string"` is in the calling sequence)

Description

`input(message)` gives the user the prompt in the text string and then waits for input from the keyboard. The input can be expression which is evaluated by evstr. If nothing but a carriage return is entered at the prompt `input(message)` returns an empty matrix

Invoked with two arguments, the output is a character string which is the expression entered at keyboard. If nothing but a carriage return is entered at the prompt `input(message)` returns a single white space `" "`.

Examples

```
//x=input("How many iterations?")
//x=input("What is your name?","string")
```

See Also

evstr , x_dialog , x_mdialog

Name

lib — library definition

```
xlib = lib('lib-dir')
```

Parameters

lib-dir
: character string

Description

`lib-dir` is a character string defining a directory that contains compiled Scilab function (`.bin`) files.

In addition to these files `lib-dir` must have a file called `names`, that contains the names of the functions defined in `lib-dir`. On success, all functions in `lib-dir` are available from within Scilab. They are loaded on demand when called for the first time.

Binary files can be created from within Scilab with the command `save`.

Scilab's standard libraries are defined using `lib` on the `SCIDIR/macros/*` subdirectories.

A library variable usually is saved for later loading, either on-line or from the user-specific startup file (see startup).

Restrictions

Scilab tacitly assumes that each xxxx.bin file defines a variable named `xxxx`.

Examples

```
//define some variables
function z = myplus(x, y), z = x + y,endfunction
function z = yourplus(x, y), x = x - y,endfunction
A=1:10;

//create the *.bin files in libdir
libdir=TMPDIR
save(libdir + '/myplus.bin', myplus);
save(libdir + '/yourplus.bin', yourplus);
save(libdir + '/A.bin', A);

//create the name file
mputl(['myplus';'yourplus';'A'],TMPDIR+'/names');

//build the library containing myplus and yourplus
xlib = lib(libdir+'/')

//erase the variables
clear myplus yourplus A

//Automatic loading and execution
myplus(1,2)
```

```
A
```

See Also

library , genlib , save , deff , getf , whereis

Name

load — load saved variable

```
load(filename [,x1,...,xn])
load(fd [,x1,...,xn])
```

Parameters

filename
: character string containing the path of the file

fd
: a file descriptor given by a call to mopen

xi
: arbitrary Scilab variable name(s) given as strings.

Description

The `load` command can be used to reload in the Scilab session variables previously saved in a file with the `save` command. If the file contains graphic handle variables, the corresponding graphics_entities are drawn.

Since Scilab 5.0, all uimenu or uicontrol handles are also drawn.

`load(filename)` loads the variables saved in file given by its path `filename`.

`load(fd)` loads the variables saved in file given by its descriptor `fd`.

`load(filename,'x','y')` or `load(fd,'x','y')` loads only variables `x,y`.

Even if the binary file format has changed with 2.5 version, `load(filename,...)` is able to read old format files. Previous file format can be accessed for a while using function oldsave ans oldload.

Examples

```
a=eye(2,2);b=ones(a);
save('vals.dat',a,b);
clear a
clear b
load('vals.dat','a','b');
```

See Also

save , listvarinfile , save_format , getf , mopen

Name

newest — returns newest file of a set of files

```
k=newest(paths)
k=newest(path1,path2,...,pathn)
```

Parameters

k
: the index of the newest file

paths
: a character string vector, paths(i) is the pathname of ith file

pathi
: a character string, the pathname of ith file

Description

Given a set of pathnames newest returns the index of the newest one. Non existant files are supposed to be the oldest.

Examples

```
newest('SCI/modules/graphics/macros/bode.sci','SCI/modules/graphics/macros/bode
newest(['SCI/modules/graphics/macros/bode.sci','SCI/modules/graphics/macros/bod
newest('SCI/modules/graphics/macros/bode.'+['sci','bin'])
```

See Also

fileinfo

Name

oldload — load saved variable in 2.4.1 and previous formats

```
oldload('file-name' [,x1,...,xn])
```

Parameters

file-name
: character string

xi
: arbitrary Scilab variable name(s) given as strings.

Description

The `oldload` function is obsolete and is retained only for compatibility purpose.

The `oldload` command can be used to reload in the Scilab session variables previously saved in a file with the `save` command.

`oldload('file-name')` loads the variables saved in file `'file-name'`.

`oldload('file-name','x','y',...,'z')` loads only variables `x,y,...,z` stored in file `'file-name'`.

Examples

```
a=eye(2,2);b=ones(a);
oldsave(TMPDIR+'/vals.dat',a,b);
clear a
clear b
oldload(TMPDIR+'/vals.dat','a','b');
```

See Also

save , getf

Name

oldsave — saving variables in 2.4.1 and previous format

```
oldsave(filename [,x1,x2,...,xn])
```

Parameters

filename
: character string or a logical unit returned by file('open',...)

xi
: arbitrary Scilab variable(s)

Description

The `oldsave` function is obsolete and is retained only for compatibility purpose.

The `oldsave` command can be used to save Scilab current variables in binary form in a file.

`oldsave(filename)` saves all current variables in the file defined by `filename`.

`oldsave(file-name,x,y)` saves only named variables `x` and `y`.

Saved variables can be reloaded by the `load` or oldload command.

Examples

```
a=eye(2,2);b=ones(a);
oldsave('TMPDIR/val.dat',a,b);
clear a
clear b
oldload('TMPDIR/val.dat','a','b');
```

See Also

load , file

Name

print — prints variables in a file

```
print('file-name',x1,[x2,...xn])
```

Description

prints `xi` on file `'file-name'` with the current format, i.e. the format used by scilab to display the variables. All types of variables may be "`print`"ed

Note : `xi` must be a named variable, with expressions variable name part of the display is unpredictable.

`print(%io(2),...)` prints on Scilab's window. this syntax may be used to display variables within a macro.

Examples

```
a=rand(3,3);p=poly([1,2,3],'s');l=list(1,'asdf',[1 2 3]);
print(%io(2),a,p,l)
write(%io(2),a)
```

See Also

write , read , format , printf , disp

Name

printf — Emulator of C language printf function

```
printf(format,value_1,..,value_n)
```

Parameters

format
: a Scilab string. Specifies a character string combining literal characters with conversion specifications.

value_i
: Specifies the data to be converted according to the format parameter.

str
: column vector of character strings

file
: a Scilab string specifying a file name or a logical unit number (see `file`)

Description

The `printf` function converts, formats, and writes its `value` parameters, under control of the `format` parameter, to the standard output.

The `format` parameter is a character string that contains two types of objects:

Literal characters
: which are copied to the output stream.

Conversion specifications
: each of which causes zero or more items to be fetched from the `value` parameter list. see printf_conversion for details

If any `value`s remain after the entire `format` has been processed, they are ignored.

Examples

```
printf('Result is:\nalpha=%f',0.535)
```

See Also

string , print , write , format , disp , file , fprintf , sprintf , printf_conversion

Name

printf_conversion — printf, sprintf, fprintf conversion specifications

Description

Each conversion specification in the `printf` , `sprintf` , `fprintf` `format` parameter has the following syntax:

- A % (percent) sign.
- Zero or more `options`, which modify the meaning of the conversion specification. The following list contains the `option` characters and their meanings:
- Left align, within the field, the result of the conversion.
- Begin the result of a signed conversion with a sign (+ or -).
- Prefix a space character to the result if the first character of a signed conversion is not a sign. If both the (space) and + options appear, the (space) option is ignored.
- Convert the value to an alternate form. For `c`, `d`, `i`, `s`, and `u` conversions, the # option has no effect. For `o` conversion, # increases the precision to force the first digit of the result to be a 0 (zero). For `x` and `X` conversions, a nonzero result has 0x or 0X prefixed to it. For `e, E, f, g,` and `G` conversions, the result always contains a decimal point, even if no digits follow it. For `g` and `G` conversions, trailing zeros are not removed from the result.
- Pad to the field width, using leading zeros (following any indication of sign or base) for `d`, `i`, `o`, `u`, `x`, `X`, `e`, `E`, `f`, `g`, and `G` conversions; no space padding is performed. If the `0` and `\-` (dash) flags both appear, the `0` flag is ignored. For `d`, `i`, `o` `u`, `x`, and `X` conversions, if a precision is specified, the `0` flag is also ignored.

An optional decimal digit string that specifies the minimum field width. If the converted value has fewer characters than the field width, the field is padded on the left to the length specified by the field width. If the left-adjustment option is specified, the field is padded on the right.

An optional precision. The precision is a `.` (dot) followed by a decimal digit string. If no precision is given, the parameter is treated as 0 (zero). The precision specifies:

- The minimum number of digits to appear for `d`, `u`, `o`, `x`, or `X` conversions
- The number of digits to appear after the decimal point for `e`, `E`, and `f` conversions
- The maximum number of significant digits for `g` and `G` conversions
- The maximum number of characters to be printed from a string in an `s` conversion
- A character that indicates the type of conversion to be applied:
- Performs no conversion. Displays %.
- :Accepts an integer `value` and converts it to signed decimal notation. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string. Specifying a field width with a zero as a leading character causes the field width value to be padded with leading zeros.
- :Accepts an integer `value` and converts it to unsigned decimal notation. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string. Specifying a field width with a zero as the leading character causes the field width value to be padded with leading zeros.

- :Accepts an integer `value` and converts it to unsigned octal notation. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string. Specifying a field width with a zero as the leading character causes the field width value to be padded with leading zeros. An octal value for field width is not implied.
- :Accepts an integer `value` and converts it to unsigned hexadecimal notation. The letters ``abcdef'' are used for the `x` conversion; the letters ``ABCDEF'' are used for the `X` conversion. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string. Specifying a field width with a zero as the leading character causes the field width value to be padded with leading zeros.
- Accepts a float or double `value` and converts it to decimal notation in the format %[\-]`ddd.ddd`. The number of digits after the decimal point is equal to the precision specification.
- If no precision is specified, six digits are output.
- If the precision is zero, no decimal point appears and the system outputs a number rounded to the integer nearest to `value`.
- If a decimal point is output, at least one digit is output before it.
- :Accepts a real and converts it to the exponential form %[\-]`d.ddde`+/\-`dd`. There is one digit before the decimal point, and the number of digits after the decimal point is equal to the precision specification.
- If no precision is specified, , six digits are output.
- If the precision is zero, , no decimal point appears.
- The `E` conversion character produces a number with E instead of e before the exponent. The exponent always contains at least two digits. If the value is zero, the exponent is zero.
- Accepts a real and converts it in the style of the `e`, `E`, or `f` conversion characters, with the precision specifying the number of significant digits. Trailing zeros are removed from the result. A decimal point appears only if it is followed by a digit. The style used depends on the value converted. Style `e` (`E`, if `G` is the flag used) results only if the exponent resulting from the conversion is less than -4, or if it is greater or equal to the precision.
- :Accepts and displays an integer value converted to a character.
- :Accepts a string `value` and displays characters from the string to the end or the number of characters indicated by the precision is reached. If no precision is specified, all characters up to the end are displayed.

A field width or precision can be indicated by an `*` (asterisk) instead of a digit string. In this case, an integer `value` parameter supplies the field width or precision. The `value` parameter converted for output is not fetched until the conversion letter is reached, so the parameters specifying field width or precision must appear before the value to be converted (if any).

If the result of a conversion is wider than the field width, the field is expanded to contain the converted result.

The representation of the plus sign depends on whether the + or (space) formatting option is specified.

See Also

printf , fprintf , sprintf

Name

read — matrices read

```
[x]=read(file-desc,m,n,[format])
[x]=read(file-desc,m,n,k,format)
```

Parameters

file-desc
: character string specifying the file name or integer value specifying logical unit (see file).

m, n
: integers (dimensions of the matrix x). Set m=-1 if you do not know the numbers of rows, so the whole file is read.

format : character string, specifies a "Fortran" format. This
: character string must begin with a right parenthesis and end with a left parenthesis. Formats cannot mix floating point or character edition modes.

k
: integer or vector of integer

Description

reads row after row the `mxn` matrix `x` (`n=1` for character chain) in the file `file-desc` (string or integer). Each row of the matrix `x` begin in a new line of `file-desc` file. Depending on `format`, a given row of the `x` matrix may be read from more than one line of `file-desc` file.

The type of the result will depend on the specified format. If format contains only `(d,e,f,g)` descriptors the function tries to read numerical data (the result is matrix of real numbers).

If format contains only `a` descriptors the function tries to read character strings (the result is a character string column vector). In this case n must be equal to 1. Warning: The character strings are truncated when they are longuer than 4093.

Examples for `format`:

```
(1x,e10.3,5x,3(f3.0))
(10x,a20)
```

When format is omitted datas are read using numerical free format: blank, comma and slash may be used as data separators, n*v may be use to represent n occurrences of value n.

A direct access file can be used if using the parameter `k` which is is the vector of record numbers to be read (one record per row), thus `m` must be `m=prod(size(k))`.

To read on the keyboard use `read(%io(1),...)`.

Remark

Last line of data files must be terminated by a newline to be taken into account.

Examples

```
if MSDOS then unix('del foo');
else unix('rm -f foo'); end
A=rand(3,5); write('foo',A);
B=read('foo',3,5)
B=read('foo',-1,5)
read(%io(1),1,1,'(a)')  // waits for user's input
```

See Also

file , readb , write , x_dialog , mscanf , mfscanf , msscanf , fscanfMat

Name

read4b — fortran file binary read

```
x=read4b(file-name,m,n [,rec])
```

Parameters

file-name
: string or integer

m, n
: integers (dimensions of the matrix x). Set m=-1 if you do not know the numbers of rows, so all the file is read

rec
: vector of positive integers. the selected records for direct access. This vector size must be equal to the number of rows of desired `x`.

Description

binary read of the matrix `x` in the file `file-name`. Matrix entries are supposed to have been stored on 4 byte words.

For direct record access, file must have been previously opened using `file` function to set the record_length. `file-name` must be the result of the `file` function.

See Also

file , write , writb , mget , write4b

Name

readb — fortran file binary read

```
x=readb(file-name,m,n [,rec])
```

Parameters

file-name
: string or integer

m, n
: integers (dimensions of the matrix x). Set m=-1 if you do not know the numbers of rows, so all the file is read

rec
: vector of positive integers. the selected records for direct access. This vector size must be equal to the number of rows of desired `x`.

Description

binary read of the matrix `x` in the file `file-name`. Matrix entries are supposed to have been stored on 8 byte words.

For direct record access, file must have been previously opened using `file` function to set the record_length. `file-name` must be the result of the `file` function.

See Also

file , write , writb , mget , read4b

Name

readc_ — read a character string

```
[c]=readc_(unit)
[c]=readc_()
```

Description

`readc_` reads a character string. This function allows one to interrupt an exec file without `pause`; the exec file stops until carriage return is made.

See Also

read

Name

save — saving variables in binary files

```
save(filename [,x1,x2,...,xn])
save(fd [,x1,x2,...,xn])
```

Parameters

filename
: character string containing the path of the file

fd
: a file descriptor given by a call to mopen

xi
: arbitrary Scilab variable(s)

Description

The `save` command can be used to save Scilab current variables in a binary file. If a variable is a graphic handle, the `save` function saves all the corresponding graphics_entities definition.

Since Scilab 5.0, all uimenu or uicontrol handles are also saved by this function.

The file can be given either by its paths or by its descriptor previously given by `mopen`.

`save(filename)` saves all current variables in the file defined by `filename`.

`save(fd)` saves all current variables in the file defined by the descriptor `fd`.

`save(filename,x,y)` or `save(fd,x,y)` saves only named variables `x` and `y`.

Saved variables can be reloaded by the `load` command.

Examples

```
a=eye(2,2);b=ones(a);
save('val.dat',a,b);
clear a
clear b
load('val.dat','a','b');

// sequential save into a file
fd=mopen('TMPDIR/foo','wb')
for k=1:4, x=k^2;save(fd,x,k),end
mclose(fd)
fd=mopen('TMPDIR/foo','rb')
for i=1:4, load(fd,'x','k');x,k,end
mclose(fd)

// appending variables to an old save file
fd=mopen('TMPDIR/foo','r+')
mseek(0,fd,'end')
lst=list(1,2,3)
save(fd,lst)
```

```
mclose(fd)
```

See Also

load, save_format, mopen

Name

setenv — set the value of an environment variable

```
rep=setenv(name, value )
```

Parameters

name
: Points to the name of an environment variable . (name is a string)

value
: Points to the value to be assigned to the environment variable. (value is a string)

rep
: Returns %T if it is ok else %F.

Description

set the value of an environnment variable.

Examples

```
setenv('toto','example')
getenv('toto')
```

See Also

getenv

Authors

Allan CORNET

Name

sprintf — Emulator of C language sprintf function

```
str=sprintf(format,value_1,..,value_n)
```

Parameters

format
: a Scilab string. Specifies a character string combining literal characters with conversion specifications.

value_i
: Specifies the data to be converted according to the format parameter.

str
: column vector of character strings

Description

The `sprintf` function converts, formats, and stores its `value` parameters, under control of the `format` parameter.

The `format` parameter is a character string that contains two types of objects:

Literal characters
: which are copied to the output stream.

Conversion specifications
: each of which causes zero or more items to be fetched from the `value` parameter list. see printf_conversion for details

If there are not enough items for `format` in the `value` parameter list, `sprintf` generate an error. If any `value`s remain after the entire `format` has been processed, they are ignored.

Note: `sprintf` is obsolete, use `msprintf` instead.

Examples

```
fahr=120
sprintf('%3d Fahrenheit = %6.1f Celsius',fahr,(5/9)*(fahr-32))
```

See Also

string , print , write , format , disp , file , printf , fprintf , msprintf , printf_conversion

Name

sscanf — Converts formatted input given by a string

```
[v_1,...v_n]=sscanf (string,format)
```

Parameters

format
:Specifies the format conversion.

string
:Specifies input to be read.

Description

This function is obsolete, use preferably the `msscanf` function which is more efficient and is more compatible with the C `sscanf` procedure.

The sscanf functions interpret character string according to a format, and returns the converted results.

The format parameter contains conversion specifications used to interpret the input.

The format parameter can contain white-space characters (blanks, tabs, newline, or formfeed) that, except in the following two cases, read the input up to the next nonwhite-space character. Unless there is a match in the control string, trailing white space (including a newline character) is not read.

- Any character except % (percent sign), which must match the next character of the input stream.
- A conversion specification that directs the conversion of the next input field. see scanf_conversion for details.

See Also

mprintf , msscanf , mfscanf , scanf_conversion

Name

unix — shell (sh) command execution

```
stat=unix(command-name)
```

Parameters

command-name
: A character string containing Unix sh instruction

stat
: An integer flag

Description

Sends a string `command-name` to Unix for execution by the sh shell. Standard output and standard errors of the shell command are written in the calling shell. `stat` gives -1 if unix can't be called (Not enough system memory available) or the sh return code.

Examples

```
if ~MSDOS then
  unix("ls $SCI/demos");
end

function wd=directory()
  if MSDOS then
    unix('cd>'+TMPDIR+'\path');
  else
    unix('pwd>'+TMPDIR+'/path');
  end
  wd=read(TMPDIR+'/path',1,1,'(a)');
endfunction

wd=directory()
```

See Also

edit , manedit , unix_g , unix_s , unix_w , unix_x , host

Name

unix_g — shell (sh) command execution, output redirected to a variable

```
rep=unix_g(cmd)
[rep,stat]=unix_g(cmd)
[rep,stat,stderr]=unix_g(cmd)
```

Parameters

cmd
: a character string

rep
: a column vector of character strings (standard output)

stat
: a integer, the error status. stat=0 if no error occured

err
: a column vector of character strings (standard error)

Description

Sends a string `cmd` to Unix for execution by the sh shell. The standard output is redirected to scilab variable `rep`. The standard error is redirected to scilab variable err or displays if you had only 2 ouput arguments. Unix execution errors are trapped; *NOTE* that only the last shell command error is reported when a list of command separated by ";" is sent: this is not recommended.

Examples

```
function d=DIR(path)
  path=pathconvert(path,%t,%t)
  if MSDOS then
    d=unix_g('dir '+path)
  else
    d=unix_g('ls '+path)
  end
endfunction

DIR('SCI/etc')
```

See Also

unix_s, unix_w, unix_x, unix

Name

unix_s — shell (sh) command execution, no output

```
unix_s(cmd)
```

Parameters

cmd
: a character string

Description

Sends a string `cmd` to Unix for execution by the sh shell. The standard output is redirected to /dev/null. Unix execution errors are trapped; *NOTE* that only the last shell command error is reported when a list of command separated by ";" is sent: this is not recommended.

Examples

```
if MSDOS then
  unix_s("del foo");
else
  unix_s("rm -f foo");
end
```

See Also

edit , manedit , unix_g , unix_w , unix_x , unix

Name

unix_w — shell (sh) command execution, output redirected to scilab window

```
unix_w(cmd)
```

Parameters

cmd
: a character string

Description

Sends a string `cmd` to Unix for execution by the sh shell. The standard output is redirected to scilab window. Unix execution errors are trapped; *NOTE* that only the last shell command error is reported when a list of command separated by ";" is sent: this is not recommended.

Examples

```
if MSDOS then
  unix_w("dir "+'""'+WSCI+"\modules"+'""');
else
  unix_w("ls $SCI/modules");
end
```

See Also

edit , manedit , unix_g , unix_s , unix_x , unix

Name

unix_x — shell (sh) command execution, output redirected to a window

```
unix_x(cmd)
```

Parameters

cmd
: a character string

Description

Sends a string `cmd` to Unix for execution by the sh shell. The standard output is redirected to a window. Unix execution errors are trapped; *NOTE* that only the last shell command error is reported when a list of command separated by ";" is sent: this is not recommended.

Examples

```
if MSDOS then
  unix_x("dir "+""""+WSCI+"modules\graphics\demos"+"""");
else
  unix_x("ls $SCI/modules/graphics/demos");
end
```

See Also

edit , manedit , unix_g , unix_s , unix_w , unix

Name

writb — fortran file binary write

```
writb(file-name,a [,rec])
```

Parameters

file-name
: string or integer

rec
: vector of positive integers. the selected records for direct access. This vector size must be equal to the number of rows of `a`

Description

writes in binary format the matrix `a` in the file `'filename'`.. Matrix entries are stored on 4 byte words

For direct record access, file must have been previously opened using `file` function to set the record_length. `file-name` must be the result of the `file` function.

See Also

file , readb , write , mput , write4b

Name

write — write in a formatted file

```
write(file-desc,a,[format])
write(file-desc,a,k,format)
```

Parameters

file-desc
: character string specifying the file name or integer value specifying logical unit (see file).

a
: real matrix or column vector of character strings.

format
: character string, specifies a "Fortran" format. This character string must begin with a right parenthesis and end with a left parenthesis. Formats cannot mix floating point , integer or character edition modes

k
: integer vector

Description

writes row-by-row a real matrix or a column vector of character strings in a formatted file. Each row of the `a` argument begin in a new line of `file-desc` file. Depending on `format` a given row of the `a` argument may be written in more than one line of `file-desc` file.

Format examples : `(1x,e10.3,5x,3(f3.0))` , `(10x,a20)` ;

See a Fortran book for more precision.

Direct access files : `x=write(file_desc,a,k,format)`. Here `k` is the vector of records (one record by row, i.e. `m=prod(size(k))`)

`write(%io(2),....)` writes on Scilab's window. Note that in this case `format` should produce one output line per matrix row. If this contraint is not verified unpredictable behavior could happen.

Examples

```
if MSDOS then unix('del asave');
else unix('rm -f asave'); end
A=rand(5,3); write('asave',A); A=read('asave',5,3);
write(%io(2),A,'('' | '',3(f10.3,'' | ''))')
write(%io(2),string(1:10))
write(%io(2),strcat(string(1:10),','))
write(%io(2),1:10,'(10(i2,3x))')

if MSDOS then unix('del foo');
else unix('rm -f foo'); end
write('foo',A)
```

See Also

file , fileinfo , writb , read , print , string , mfprintf , mprintf , msprintf , fprintfMat

Name

write4b — fortran file binary write

```
write4b(file-name,a [,rec])
```

Parameters

file-name
: string or integer

rec
: vector of positive integers. the selected records for direct access. This vector size must be equal to the number of rows of `a`

Description

writes in binary format the matrix `a` in the file `'filename'`. Matrix entries are stored on 8 byte words

For direct record access, file must have been previously opened using `file` function to set the record_length. `file-name` must be the result of the `file` function.

See Also

file , readb , write , mput , read4b

Integers

Name

iconvert — conversion to 1 or 4 byte integer representation

```
y=iconvert(X,itype)
```

Parameters

X
: matrix of floats or integers

y
: matrix of integers coded on one, two or four bytes.

Description

converts and stores data two one, two or four bytes integers.

itype=0
: return floating point numbers

itype=1
: return int8 numbers in the range [-128,127]

itype=11
: return uint8 numbers in the range [0,255]

itype=2
: return int16 numbers in the range [-32768,32767]

itype=12
: return uint16 numbers in the range [0, 65535]

itype=4
: return int32 numbers in the range [-2147483648,2147483647]

itype=14
: return uint32 numbers in the range [0, 4294967295]

Examples

```
b=int32([1 -120 127 312])
y=iconvert(b,1)
```

See Also

double , inttype

Name

int8 — conversion to one byte integer representation
int16 — conversion to 2 bytes integer representation
int32 — conversion to 4 bytes integer representation
uint8 — conversion to one byte unsigned integer representation
uint16 — conversion to 2 bytes unsigned integer representation
uint32 — conversion to 4 bytes unsigned integer representation

```
y=int8(X)
y=int16(X)
y=int32(X)
y=uint8(X)
y=uint16(X)
y=uint32(X)
```

Parameters

X
: matrix of floats or integers

y
: matrix of integers coded on one, two or four bytes.

Description

converts and stores data two one, two or four bytes integers. These data types are specialy useful to store big objects such as images, long signals,...

y=int8(X)
: return numbers in the range [-128,127]

y=uint8(X)
: return numbers in the range [0,255]

y=int16(X)
: return numbers in the range [-32768,32767]

y=uint16(X)
: return numbers in the range [0, 65535]

y=int32(X)
: return numbers in the range [-2147483648,2147483647]

y=uint32(X)
: return numbers in the range [0, 4294967295]

Examples

```
int8([1 -120 127 312])
uint8([1 -120 127 312])

x=int32(-200:100:400)
int8(x)
```

See Also

double , inttype , iconvert

Name

inttype — type integers used in integer data types

```
[i]=inttype(x)
```

Parameters

x
: an matrix of integers (see int8,..)

i
: integer

Description

`inttype(x)` returns an integer which is the type of the entries of `x` as following :

1 : one byte integer representation

2 : two bytes integer representation

4 : four bytes integer representation

11 : one byte unsigned integer representation

12 : two bytes unsigned integer representation

14 : four bytes unsigned integer representation

Examples

```
x=uint16(1:10);
inttype(x)
```

See Also

int8

Interpolation

Name

bsplin3val — 3d spline arbitrary derivative evaluation function

```
[dfp]=bsplin3val(xp,yp,zp,tl,der)
```

Parameters

xp, yp, zp
: real vectors or matrices of same size

tl
: tlist of type "splin3d", defining a 3d tensor spline (called `s` in the following)

der
: vector with 3 components `[ox,oy,oz]` defining which derivative of `s` to compute.

dfp
: vector or matrix of same format than `xp`, `yp` and `zp`, elementwise evaluation of the specified derivative of `s` on these points.

Description

While the function interp3d may compute only the spline `s` and its first derivatives, `bsplin3val` may compute any derivative of `s`. The derivative to compute is specified by the argument `der=[ox,oy,oz]` :

$$dfp(i) = \frac{\partial^{ox+oy+oz}}{\partial^{ox}x\,\partial^{oy}y\,\partial^{oz}z}\, s(xp(i),yp(i),zp(i))$$

So `der=[0 0 0]` corresponds to *s*, `der=[1 0 0]` to *ds/dx*, `der=[0 1 0]` to *ds/dy*, `der=[1 1 0]` to *d2s/dxdy*, etc...

For a point with coordinates *(xp(i),yp(i),zp(i))* outside the grid, the function returns 0.

Examples

```
deff("v=f(x,y,z)","v=cos(x).*sin(y).*cos(z)");
deff("v=fx(x,y,z)","v=-sin(x).*sin(y).*cos(z)");
deff("v=fxy(x,y,z)","v=-sin(x).*cos(y).*cos(z)");
deff("v=fxyz(x,y,z)","v=sin(x).*cos(y).*sin(z)");
deff("v=fxxyz(x,y,z)","v=cos(x).*cos(y).*sin(z)");
n = 20;  // n x n x n  interpolation points
x = linspace(0,2*%pi,n); y=x; z=x; // interpolation grid
[X,Y,Z] = ndgrid(x,y,z); V = f(X,Y,Z);
tl = splin3d(x,y,z,V,[5 5 5]);

// compute f and some derivates on a point
// and compare with the spline interpolant
xp = grand(1,1,"unf",0,2*%pi);
yp = grand(1,1,"unf",0,2*%pi);
zp = grand(1,1,"unf",0,2*%pi);

f_e = f(xp,yp,zp)
f_i = bsplin3val(xp,yp,zp,tl,[0 0 0])
```

```
fx_e = fx(xp,yp,zp)
fx_i = bsplin3val(xp,yp,zp,tl,[1 0 0])

fxy_e = fxy(xp,yp,zp)
fxy_i = bsplin3val(xp,yp,zp,tl,[1 1 0])

fxyz_e = fxyz(xp,yp,zp)
fxyz_i = bsplin3val(xp,yp,zp,tl,[1 1 1])

fxxyz_e = fxxyz(xp,yp,zp)
fxxyz_i = bsplin3val(xp,yp,zp,tl,[2 1 1])
```

See Also

splin3d, interp3d

Authors

R.F. Boisvert, C. De Boor (code from the CMLIB fortran lib)
B. Pincon (scilab interface)

Name

cshep2d — bidimensional cubic shepard (scattered) interpolation

```
tl_coef = cshep2d(xyz)
```

Parameters

xyz
: a n x 3 matrix of the (no gridded) interpolation points (the i th row given the (x,y) coordinates then the altitude z of the i th interpolation point)

tl_coef
: a tlist scilab structure (of type cshep2d)

Description

This function is useful to define a 2d interpolation function when the interpolation points are not on a grid (you may use it in this case but splin2d is better for that purpose). The interpolant is a cubic shepard one and is a C2 (twice continuously differentiable) bivariate function *s(x,y)* such that : *s(xi,yi)=zi* for all *i=1,..,n* (*(xi,yi,zi)* being the i th row of `xyz`).

The evaluation of *s* at some points must be done by the eval_cshep2d function.

Remark

The function works if **n >= 10**, if the nodes are not all colinears (i.e. the *(x,y)* coordinates of the interpolation points are not on the same straight line), and if there is no duplicate nodes (i.e. 2 or more interpolation points with the same *(x,y)* coordinates). An error is issued if these conditions are not respected.

Examples

```
// interpolation of cos(x)cos(y) with randomly choosen interpolation points
n = 150; // nb of interpolation points
xy = grand(n,2,"unf",0,2*%pi);
z = cos(xy(:,1)).*cos(xy(:,2));
xyz = [xy z];
tl_coef = cshep2d(xyz);

// evaluation on a grid
m = 30;
xx = linspace(0,2*%pi,m);
[X,Y] = ndgrid(xx,xx);
Z = eval_cshep2d(X,Y, tl_coef);
xbasc()
plot3d(xx,xx,Z,flag=[2 6 4])
param3d1(xy(:,1),xy(:,2),list(z,-9), flag=[0 0])
xtitle("Cubic Shepard Interpolation of cos(x)cos(y) with randomly choosen interp
legends("interpolation points",-9,1)
xselect()
```

See Also

splin2d, eval_cshep2d

Authors

Robert J. Renka
B. Pincon (scilab interface)

Name

eval_cshep2d — bidimensional cubic shepard interpolation evaluation

```
[zp [,dzpdx, dzpdy [,d2zpdxx,d2zpdxy,d2zpdyy]]] = eval_cshep2d(xp, yp, tl_coef)
```

Parameters

xp, yp
: two real vectors (or matrices) of the same size

tl_coef
: a tlist scilab structure (of type cshep2d) defining a cubic Shepard interpolation function (named S in the following)

zp
: vector (or matrix) of the same size than xp and yp, evaluation of the interpolant S at these points

dzpdx,dzpdy
: vectors (or matrices) of the same size than xp and yp, evaluation of the first derivatives of S at these points

d2zpdxx,d2zpdxy,d2zpdyy
: vectors (or matrices) of the same size than xp and yp, evaluation of the second derivatives of S at these points

Description

This is the evaluation routine for cubic Shepard interpolation function computed with cshep2d, that is :

$$\mathrm{zp}(i) = S(\mathrm{xp}(i),\mathrm{yp}(i))$$

$$\mathrm{dzpdx}(i) = \frac{\partial S}{\partial x}(\mathrm{xp}(i),\mathrm{yp}(i))$$

$$\mathrm{dzpdy}(i) = \frac{\partial S}{\partial y}(\mathrm{xp}(i),\mathrm{yp}(i))$$

$$\mathrm{d2zpdxx}(i) = \frac{\partial^2 S}{\partial x^2}(\mathrm{xp}(i),\mathrm{yp}(i))$$

$$\mathrm{d2zpdxy}(i) = \frac{\partial^2 S}{\partial x \partial y}(\mathrm{xp}(i),\mathrm{yp}(i))$$

$$\mathrm{d2zpdyy}(i) = \frac{\partial^2 S}{\partial y^2}(\mathrm{xp}(i),\mathrm{yp}(i))$$

Remark

The interpolant *S* is C2 (twice continuously differentiable) but is also extended by zero for *(x,y)* far enough the interpolation points. This leads to a discontinuity in a region far outside the interpolation points, and so, is not cumbersome in practice (in a general manner, evaluation outside interpolation points (i.e. extrapolation) leads to very inacurate results).

Examples

```
// see example section of cshep2d

// this example shows the behavior far from the interpolation points ...
```

```
deff("z=f(x,y)","z = 1+ 50*(x.*(1-x).*y.*(1-y)).^2")
x = linspace(0,1,10);
[X,Y] = ndgrid(x,x);
X = X(:); Y = Y(:); Z = f(X,Y);
S = cshep2d([X Y Z]);
// evaluation inside and outside the square [0,1]x[0,1]
m = 40;
xx = linspace(-1.5,0.5,m);
[xp,yp] = ndgrid(xx,xx);
zp = eval_cshep2d(xp,yp,S);
// compute facet (to draw one color for extrapolation region
// and another one for the interpolation region)
[xf,yf,zf] = genfac3d(xx,xx,zp);
color = 2*ones(1,size(zf,2));
// indices corresponding to facet in the interpolation region
ind=find( mean(xf,"r")>0 & mean(xf,"r")<1 & mean(yf,"r")>0 & mean(yf,"r")<1 );
color(ind)=3;
xbasc();
plot3d(xf,yf,list(zf,color), flag=[2 6 4])
legends(["extrapolation region","interpolation region"],[2 3],1)
xselect()
```

See Also

cshep2d

Authors

Robert J. Renka
B. Pincon (scilab interface)

Name

interp — cubic spline evaluation function

```
[yp [,yp1 [,yp2 [,yp3]]]]=interp(xp, x, y, d [, out_mode])
```

Parameters

xp
: real vector or matrix

x,y,d
: real vectors of the same size defining a cubic spline or sub-spline function (called `s` in the following)

out_mode
: (optional) string defining the evaluation of `s` outside the [x1,xn] interval

yp
: vector or matrix of same size than `xp`, elementwise evaluation of `s` on `xp` (yp(i)=s(xp(i) or yp(i,j)=s(xp(i,j))

yp1, yp2, yp3
: vectors (or matrices) of same size than `xp`, elementwise evaluation of the successive derivatives of `s` on `xp`

Description

Given three vectors `(x,y,d)` defining a spline or sub-spline function (see splin) with `yi=s(xi), di = s'(xi)` this function evaluates `s` (and `s', s'', s'''` if needed) at `xp(i)` :

$$yp(i) = s(xp(i)) \text{ or } yp(i,j) = s(xp(i,j))$$
$$yp1(i) = s'(xp(i)) \text{ or } yp1(i,j) = s'(xp(i,j))$$
$$yp2(i) = s''(xp(i)) \text{ or } yp2(i,j) = s''(xp(i,j))$$
$$yp3(i) = s'''(xp(i)) \text{ or } yp3(i,j) = s'''(xp(i,j))$$

The `out_mode` parameter set the evaluation rule for extrapolation, i.e. for `xp(i)` not in [x1,xn] :

"by_zero"
: an extrapolation by zero is done

"by_nan"
: extrapolation by Nan

"C0"
: the extrapolation is defined as follows :

$$s(x) = y_1 \text{ for } x < x_1$$
$$s(x) = y_n \text{ for } x > x_n$$

"natural"
: the extrapolation is defined as follows (*p_i* being the polynomial defining `s` on [x_i,x_{i+1}]) :

$$s(x) = p_1(x) \text{for} x < x_1$$
$$s(x) = p_{n-1}(x) \text{for} x > x_n$$

"linear"
the extrapolation is defined as follows :

$$s(x) = y_1 + s'(x_1) \cdot (x - x_1) \text{for} x < x_1$$
$$s(x) = y_n + s'(x_n) \cdot (x - x_n) \text{for} x > x_n$$

"periodic"
: s is extended by periodicity.

Examples

```
// see the examples of splin and lsq_splin

// an example showing C2 and C1 continuity of spline and subspline
a = -8; b = 8;
x = linspace(a,b,20)';
y = sinc(x);
dk = splin(x,y);  // not_a_knot
df = splin(x,y, "fast");
xx = linspace(a,b,800)';
[yyk, yy1k, yy2k] = interp(xx, x, y, dk);
[yyf, yy1f, yy2f] = interp(xx, x, y, df);
xbasc()
subplot(3,1,1)
plot2d(xx, [yyk yyf])
plot2d(x, y, style=-9)
legends(["not_a_knot spline","fast sub-spline","interpolation points"],...
        [1 2 -9], "ur",%f)
xtitle("spline interpolation")
subplot(3,1,2)
plot2d(xx, [yy1k yy1f])
legends(["not_a_knot spline","fast sub-spline"], [1 2], "ur",%f)
xtitle("spline interpolation (derivatives)")
subplot(3,1,3)
plot2d(xx, [yy2k yy2f])
legends(["not_a_knot spline","fast sub-spline"], [1 2], "lr",%f)
xtitle("spline interpolation (second derivatives)")

// here is an example showing the different extrapolation possibilities
x = linspace(0,1,11)';
y = cosh(x-0.5);
d = splin(x,y);
xx = linspace(-0.5,1.5,401)';
yy0 = interp(xx,x,y,d,"C0");
yy1 = interp(xx,x,y,d,"linear");
yy2 = interp(xx,x,y,d,"natural");
yy3 = interp(xx,x,y,d,"periodic");
xbasc()
plot2d(xx,[yy0 yy1 yy2 yy3],style=2:5,frameflag=2,leg="C0@linear@natural@period
xtitle(" different way to evaluate a spline outside its domain")
```

See Also

splin, lsq_splin

Authors

B. Pincon

Name

interp3d — 3d spline evaluation function

```
[fp[,dfpdx,dfpdy,dfpdz]]=interp3d(xp,yp,zp,tl,out_mode)
```

Parameters

xp, yp, zp
: real vectors or matrices of same size

tl
: tlist of type "splin3d", defining a 3d tensor spline (called `s` in the following)

out_mode
: (optional) string defining the evaluation of `s` outside the grid ([xmin,xmax]x[ymin,ymax]x[zmin,zmax])

fp
: vector or matrix of same format than `xp`, `yp` and `zp`, elementwise evaluation of `s` on these points.

dfpdx, dfpdy, dfpdz
: vectors (or matrices) of same format than `xp`, `yp` and `zp`, elementwise evaluation of the first derivatives of `s` on these points.

Description

Given a tlist `tl` defining a 3d spline function (see splin3d) this function evaluates *s* (and *ds/dx*, *ds/dy*, *ds/dz* if needed) at *(xp(i),yp(i),zp(i))* :

$$\mathrm{zp}(i) = s(\mathrm{xp}(i),\mathrm{yp}(i))$$
$$\mathrm{dzpdx}(i) = \frac{ds}{dx}(\mathrm{xp}(i),\mathrm{yp}(i),\mathrm{zp}(i))$$
$$\mathrm{dzpdy}(i) = \frac{ds}{dy}(\mathrm{xp}(i),\mathrm{yp}(i),\mathrm{zp}(i))$$
$$\mathrm{dzpdz}(i) = \frac{ds}{dz}(\mathrm{xp}(i),\mathrm{yp}(i),\mathrm{zp}(i))$$

The `out_mode` parameter defines the evaluation rule for extrapolation, i.e. for *(xp(i),yp(i),zp(i)) not in [xmin,xmax]x[ymin,ymax]x[zmin,zmax]*:

"by_zero"
: an extrapolation by zero is done

"by_nan"
: extrapolation by Nan

"C0"
: the extrapolation is defined as follows :

```
s(x,y) = s(proj(x,y)) where proj(x,y) is nearest point
                      of [x(1),x(nx)]x[y(1),y(ny)] from (x,y)
```

"periodic"
: `s` is extended by periodicity.

Examples

```
// see the examples of the splin3d help page
```

See Also

splin3d, bsplin3val

Authors

R.F. Boisvert, C. De Boor (code from the CMLIB fortran lib)
B. Pincon (scilab interface)

Name

interpln — linear interpolation

```
[y]=interpln(xyd,x)
```

Parameters

xyd
: 2 row matrix (xy coordinates of points)

x
: vector (abscissae)

y
: vector (y-axis values)

Description

given `xyd` a set of points in the xy-plane which increasing abscissae and `x` a set of abscissae, this function computes `y` the corresponding y-axis values by linear interpolation.

Examples

```
x=[1 10 20 30 40];
y=[1 30 -10 20 40];
plot2d(x',y',[-3],"011"," ",[-10,-40,50,50]);
yi=interpln([x;y],-4:45);
plot2d((-4:45)',yi',[3],"000");
```

See Also

splin, interp, smooth

Name

intsplin — integration of experimental data by spline interpolation

```
v = intsplin([x,] s)
```

Parameters

x
: vector of increasing x coordinate data. Default value is `1:size(y,'*')`

s
: vector of y coordinate data

v
: value of the integral

Description

computes :

Where f is a function described by a set of experimental value:

`s(i)=f(x(i))` and `x0=x(1)`, `x1=x(n)`

Between mesh points function is interpolated using spline's.

Examples

```
t=0:0.1:%pi
intsplin(t,sin(t))
```

See Also

intg, integrate, inttrap, splin

Name

linear_interpn — n dimensional linear interpolation

```
vp = linear_interpn(xp1,xp2,..,xpn, x1, ..., xn, v [,out_mode])
```

Parameters

xp1, xp2, .., xpn
: real vectors (or matrices) of same size

x1 ,x2, ..., xn
: strictly increasing row vectors (with at least 2 components) defining the n dimensional interpolation grid

v
: vector (case n=1), matrix (case n=2) or hypermatrix (case n > 2) with the values of the underlying interpolated function at the grid points.

out_mode
: (optional) string defining the evaluation outside the grid (extrapolation)

vp
: vector or matrix of same size than `xp1, ..., xpn`

Description

Given a n dimensional grid defined by the n vectors `x1 ,x2, ..., xn` and the values `v` of a function (says *f*) at the grid points :

$$v(i1,i2,\cdots,in) = f(x1(i1),x2(i2),\cdots,xn(in))$$

this function computes the linear interpolant of *f* from the grid (called *s* in the following) at the points which coordinates are defined by the vectors (or matrices) `xp1, xp2, ..., xpn`:

$$vp(i) = s(xp1(i),xp2(i),\cdots,xpn(i))$$

$$\text{or } vp(i,j) = s(xp1(i,j),xp2(i,j),\cdots,xpn(i,j)) \text{ in case the xpk are matrices}$$

The `out_mode` parameter set the evaluation rule for extrapolation: if we note *Pi=(xp1(i),xp2(i),...,xpn(i))* then `out_mode` defines the evaluation rule when:

$$P(i) \notin [x1(1),x1(\$)] \times [x2(1),x2(\$)] \times \cdots \times [xn(1),xn(\$)]$$

The different choices are:

"by_zero"
: an extrapolation by zero is done

"by_nan"
: extrapolation by Nan

"C0"
: the extrapolation is defined as follows:

```
s(P) = s(proj(P)) where proj(P) is nearest point from P
                  located on the grid boundary.
```

"natural"
: the extrapolation is done by using the nearest n-linear patch from the point.

"periodic"
: s is extended by periodicity.

Examples

```
// example 1 : 1d linear interpolation
x = linspace(0,2*%pi,11);
y = sin(x);
xx = linspace(-2*%pi,4*%pi,400)';
yy = linear_interpn(xx, x, y, "periodic");
xbasc()
plot2d(xx,yy,style=2)
plot2d(x,y,style=-9, strf="000")
xtitle("linear interpolation of sin(x) with 11 interpolation points")

// example 2 : bilinear interpolation
n = 8;
x = linspace(0,2*%pi,n); y = x;
z = 2*sin(x')*sin(y);
xx = linspace(0,2*%pi, 40);
[xp,yp] = ndgrid(xx,xx);
zp = linear_interpn(xp,yp, x, y, z);
xbasc()
plot3d(xx, xx, zp, flag=[2 6 4])
[xg,yg] = ndgrid(x,x);
param3d1(xg,yg, list(z,-9*ones(1,n)), flag=[0 0])
xtitle("Bilinear interpolation of 2sin(x)sin(y)")
legends("interpolation points",-9,1)
xselect()

// example 3 : bilinear interpolation and experimentation
//             with all the outmode features
nx = 20; ny = 30;
x = linspace(0,1,nx);
y = linspace(0,2, ny);
[X,Y] = ndgrid(x,y);
z = 0.4*cos(2*%pi*X).*cos(%pi*Y);
nxp = 60 ; nyp = 120;
xp = linspace(-0.5,1.5, nxp);
yp = linspace(-0.5,2.5, nyp);
[XP,YP] = ndgrid(xp,yp);
zp1 = linear_interpn(XP, YP, x, y, z, "natural");
zp2 = linear_interpn(XP, YP, x, y, z, "periodic");
zp3 = linear_interpn(XP, YP, x, y, z, "C0");
zp4 = linear_interpn(XP, YP, x, y, z, "by_zero");
zp5 = linear_interpn(XP, YP, x, y, z, "by_nan");
xbasc()
subplot(2,3,1)
   plot3d(x, y, z, leg="x@y@z", flag = [2 4 4])
   xtitle("initial function 0.4 cos(2 pi x) cos(pi y)")
subplot(2,3,2)
   plot3d(xp, yp, zp1, leg="x@y@z", flag = [2 4 4])
   xtitle("Natural")
subplot(2,3,3)
   plot3d(xp, yp, zp2, leg="x@y@z", flag = [2 4 4])
   xtitle("Periodic")
subplot(2,3,4)
```

```
   plot3d(xp, yp, zp3, leg="x@y@z", flag = [2 4 4])
   xtitle("C0")
subplot(2,3,5)
   plot3d(xp, yp, zp4, leg="x@y@z", flag = [2 4 4])
   xtitle("by_zero")
subplot(2,3,6)
   plot3d(xp, yp, zp5, leg="x@y@z", flag = [2 4 4])
   xtitle("by_nan")
xselect()


// example 4 : trilinear interpolation (see splin3d help
//             page which have the same example with
//             tricubic spline interpolation)
getf("SCI/demos/interp/interp_demo.sci")
func =  "v=(x-0.5).^2 + (y-0.5).^3 + (z-0.5).^2";
deff("v=f(x,y,z)",func);
n = 5;
x = linspace(0,1,n); y=x; z=x;
[X,Y,Z] = ndgrid(x,y,z);
V = f(X,Y,Z);
// compute (and display) the linear interpolant on some slices
m = 41;
dir = ["z="  "z="  "z="  "x="  "y="];
val = [ 0.1   0.5   0.9   0.5   0.5];
ebox = [0 1 0 1 0 1];

XF=[]; YF=[]; ZF=[]; VF=[];
for i = 1:length(val)
   [Xm,Xp,Ym,Yp,Zm,Zp] = slice_parallelepiped(dir(i), val(i), ebox, m, m, m);
   Vm = linear_interpn(Xm,Ym,Zm, x, y, z, V);
   [xf,yf,zf,vf] = nf3dq(Xm,Ym,Zm,Vm,1);
   XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf]; VF = [VF vf];
   Vp =  linear_interpn(Xp,Yp,Zp, x, y, z, V);
   [xf,yf,zf,vf] = nf3dq(Xp,Yp,Zp,Vp,1);
   XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf]; VF = [VF vf];
end
nb_col = 128;
vmin = min(VF); vmax = max(VF);
color = dsearch(VF,linspace(vmin,vmax,nb_col+1));
xset("colormap",jetcolormap(nb_col));
xbasc()
xset("hidden3d",xget("background"))
colorbar(vmin,vmax)
plot3d(XF, YF, list(ZF,color), flag=[-1 6 4])
xtitle("tri-linear interpolation of "+func)
xselect()
```

See Also

interpln, splin, splin2d, splin3d

Authors

B. Pincon

Name

lsq_splin — weighted least squares cubic spline fitting

```
[y, d] = lsq_splin(xd, yd [, wd], x)
```

Parameters

xd, yd
: vectors of the same size, datas to be fitted by a cubic spline

wd
: (optional) a vector of same format than `xd` and `yd`, weights of the least square fit.

x
: a strictly increasing (row or column) vector, breakpoints of the cubic spline

y, d
: vectors of same format than `x`, the triplet (x,y,d) defines the approximated cubic spline.

Description

This function computes an approximated cubic spline *s* for the datas `xd, yd, wd` (in the following *m* is supposed to be the length of these vectors) and from a choice of the spline breakpoints `x` (for instance if you want n breakpoints uniformly choosen you may use `x=linspace(min(xd),max(xd),n)`). If *S* is the space of all cubic splines functions with breakpoints *x1 < x2 < ... < xn* then the resulting spline *s* is such that:

$$\sum_{k=1}^{m} wd(k)(s(xd(k)) - yd(k))^2 = \min_{f \in S} \sum_{k=1}^{m} wd(k)(f(xd(k)) - yd(k))^2$$

for all *f in S*, i.e. realizes the minimum of the sum of the squared errors over all functions of *S*.

The spline *s* is completly defined by the triplet `(x,y,d)` (`y` and `d` are the vectors of the spline ordinates and first derivatives at the xi 's : *yi=s(xi)* and *di=s'(xi)*) and its evaluation at some points must be done by the interp function.

Remarks

When `wd` is not given, all the points have the same weight 1.

A point *(xd(k),yd(k))* is considered in the fit if *xd(k) in [x1,xn] and wd(k) > 0*. In particular you can put a null (or even negative) weight to all data points you want to ignore in the fitting. When the total number of points taken into account in the fit procedure is (strictly) less than 4 an error is issued.

The vector `xd` do not need to be in increasing order.

Depending on the number and on the positions of the *xd(k)* 's and on the choice of the *x(i)* 's there may be several solutions but only one is selected. When this occurs a warning message is displayed in the Scilab command window. This function is intended to be used when *m* is much larger than *n* and in this case no such problem may occured.

Examples

```
// this is an artifical example where the datas xd and yd
// are build from a perturbed sin function
```

```
a = 0; b = 2*%pi;
sigma = 0.1;  // standard deviation of the gaussian noise
m = 200;       // number of experimental points
xd = linspace(a,b,m)';
yd = sin(xd) + grand(xd,"nor",0,sigma);

n = 6; // number of breakpoints
x = linspace(a,b,n)';

// compute the spline
[y, d] = lsq_splin(xd, yd, x);  // use equal weights

// plotting
ye = sin(xd);
ys = interp(xd, x, y, d);
xbasc()
plot2d(xd,[ye yd ys],style=[2 -2 3], ...
       leg="exact function@experimental measures (gaussian perturbation)@fitted
xtitle("a least square spline")
xselect()
```

See Also

interp, splin

Authors

C. De Boor, A.H. Morris (code from the NSWC fortran lib)
B. Pincon (scilab interface and slight modifications)

Name

smooth — smoothing by spline functions

```
[pt]=smooth(ptd [,step])
```

Parameters

ptd
: (2xn) real vector

step
: real (discretization step of abscissae)

pt
: (2xn) real vector

Description

this function computes interpolation by spline functions for a given set of points in the plane. The coordinates are `(ptd(1,i),ptd(2,i))`. The components `ptd(1,:)` must be in ascending order. The default value for the step is `abs(maxi(ptd(1,:))-mini(ptd(1,:)))/100`

Examples

```
x=[1 10 20 30 40];
y=[1 30 -10 20 40];
plot2d(x',y',[3],"011"," ",[-10,-40,50,50]);
yi=smooth([x;y],0.1);
plot2d(yi(1,:)',yi(2,:)',[1],"000");
```

See Also

splin, interp, interpln

Name

splin — cubic spline interpolation

```
d = splin(x, y [,spline_type [, der]])
```

Parameters

x
: a strictly increasing (row or column) vector (x must have at least 2 components)

y
: a vector of same format than `x`

spline_type
: (optional) a string selecting the kind of spline to compute

der
: (optional) a vector with 2 components, with the end points derivatives (to provide when spline_type="clamped")

d
: vector of the same format than `x` (`di` is the derivative of the spline at `xi`)

Description

This function computes a cubic spline or sub-spline *s* which interpolates the *(xi,yi)* points, ie, we have *s(xi)=yi* for all *i=1,..,n*. The resulting spline *s* is completly defined by the triplet `(x,y,d)` where `d` is the vector with the derivatives at the `xi`: *s'(xi)=di* (this is called the *Hermite* form). The evaluation of the spline at some points must be done by the interp function. Several kind of splines may be computed by selecting the appropriate `spline_type` parameter:

"not_a_knot"
: this is the default case, the cubic spline is computed by using the following conditions (considering n points x1,...,xn):

$$s'''(x2^-) = s'''(x2^+)$$
$$s'''(x_{n-1}^-) = s'''(x_{n-1}^+)$$

"clamped"
: in this case the cubic spline is computed by using the end points derivatives which must be provided as the last argument `der`:

$$s'(x1) = der(1)$$
$$s'(xn) = der(2)$$

"natural"
: the cubic spline is computed by using the conditions:

$$s''(x1) = 0$$
$$s''(xn) = 0$$

"periodic"
: a periodic cubic spline is computed (`y` must verify *y1=yn*) by using the conditions:

$$s'(x1) = s'(xn)$$
$$s''(x1) = s''(xn)$$

"monotone"
: in this case a sub-spline (*s* is only one continuously differentiable) is computed by using a local scheme for the *di* such that *s* is monotone on each interval:

$$\text{if } y(i) \le y(i+1) \text{ s is increasing on } [x(i), x(i+1)]$$
$$\text{if } y(i) \ge y(i+1) \text{ s is decreasing on } [x(i), x(i+1)]$$

"fast"
: in this case a sub-spline is also computed by using a simple local scheme for the *di* : d(i) is the derivative at x(i) of the interpolation polynomial of (x(i-1),y(i-1)), (x(i),y(i)),(x(i+1),y(i+1)), except for the end points (d1 being computed from the 3 left most points and dn from the 3 right most points).

"fast_periodic"
: same as before but use also a centered formula for *d1 = s'(x1) = dn = s'(xn)* by using the periodicity of the underlying function (`y` must verify *y1=yn*).

Remarks

From an accuracy point of view use essentially the **clamped** type if you know the end point derivatives, else use **not_a_knot**. But if the underlying approximated function is periodic use the **periodic** type. Under the good assumptions these kind of splines got an `O(h^4)` asymptotic behavior of the error. Don't use the **natural** type unless the underlying function have zero second end points derivatives.

The **monotone**, **fast** (or **fast_periodic**) type may be useful in some cases, for instance to limit oscillations (these kind of sub-splines have an `O(h^3)` asymptotic behavior of the error).

If *n=2* (and `spline_type` is not **clamped**) linear interpolation is used. If *n=3* and `spline_type` is **not_a_knot**, then a **fast** sub-spline type is in fact computed.

Examples

```
// example 1
deff("y=runge(x)","y=1 ./(1 + x.^2)")
a = -5; b = 5; n = 11; m = 400;
x = linspace(a, b, n)';
y = runge(x);
d = splin(x, y);
xx = linspace(a, b, m)';
yyi = interp(xx, x, y, d);
yye = runge(xx);
xbasc()
plot2d(xx, [yyi yye], style=[2 5], leg="interpolation spline@exact function")
plot2d(x, y, -9)
xtitle("interpolation of the Runge function")

// example 2 : show behavior of different splines on random datas
a = 0; b = 1;        // interval of interpolation
n = 10;              // nb of interpolation  points
m = 800;             // discretisation for evaluation
x = linspace(a,b,n)'; // abscissae of interpolation points
y = rand(x);          // ordinates of interpolation points
```

```
xx = linspace(a,b,m)';
yk = interp(xx, x, y, splin(x,y,"not_a_knot"));
yf = interp(xx, x, y, splin(x,y,"fast"));
ym = interp(xx, x, y, splin(x,y,"monotone"));
xbasc()
plot2d(xx, [yf ym yk], style=[5 2 3], strf="121", ...
       leg="fast@monotone@not a knot spline")
plot2d(x,y,-9, strf="000")  // to show interpolation points
xtitle("Various spline and sub-splines on random datas")
xselect()
```

See Also

interp, lsq_splin

Authors

B. Pincon
F. N. Fritsch (pchim.f Slatec routine is used for monotone interpolation)

Name

splin3d — spline gridded 3d interpolation

```
tl = splin3d(x, y, z, v, [order])
```

Parameters

x,y,z
: strictly increasing row vectors (each with at least 3 components) defining the 3d interpolation grid

v
: nx x ny x nz hypermatrix (nx, ny, nz being the length of `x`, `y` and `z`)

order
: (optional) a 1x3 vector [kx,ky,kz] given the order of the tensor spline in each direction (default [4,4,4], i.e. tricubic spline)

tl
: a tlist of type splin3d defining the spline

Description

This function computes a 3d tensor spline *s* which interpolates the *(xi,yj,zk,vijk)* points, ie, we have *s(xi,yj,zk)=vijk* for all *i=1,..,nx*, *j=1,..,ny* and *k=1,..,nz*. The resulting spline *s* is defined by `tl` which consists in a B-spline-tensor representation of *s*. The evaluation of *s* at some points must be done by the interp3d function (to compute *s* and its first derivatives) or by the bsplin3val function (to compute an arbitrary derivative of *s*) . Several kind of splines may be computed by selecting the order of the spline in each direction `order=[kx,ky,kz]`.

Remark

This function works under the conditions:

$$\begin{aligned} nx,ny,nz &\geq 3 \\ 2 \leq kx &< nx \\ 2 \leq ky &< ny \\ 2 \leq kz &< nz \end{aligned}$$

an error being issued when they are not respected.

Examples

```
// example 1
// ============================================================================

func =  "v=cos(2*%pi*x).*sin(2*%pi*y).*cos(2*%pi*z)";
deff("v=f(x,y,z)",func);
n = 10;  // n x n x n  interpolation points
x = linspace(0,1,n); y=x; z=x; // interpolation grid
[X,Y,Z] = ndgrid(x,y,z);
V = f(X,Y,Z);
tl = splin3d(x,y,z,V,[5 5 5]);
m = 10000;
```

```
// compute an approximated error
xp = grand(m,1,"def"); yp = grand(m,1,"def"); zp = grand(m,1,"def");
vp_exact = f(xp,yp,zp);
vp_interp = interp3d(xp,yp,zp, tl);
er = max(abs(vp_exact - vp_interp))
// now retry with n=20 and see the error



// example 2 (see linear_interpn help page which have the
//            same example with trilinear interpolation)
// ============================================================================

getf("SCI/modules/interpolation/demos/interp_demo.sci")
func =  "v=(x-0.5).^2 + (y-0.5).^3 + (z-0.5).^2";
deff("v=f(x,y,z)",func);
n = 5;
x = linspace(0,1,n); y=x; z=x;
[X,Y,Z] = ndgrid(x,y,z);
V = f(X,Y,Z);
tl = splin3d(x,y,z,V);
// compute (and display) the 3d spline interpolant on some slices
m = 41;
dir = ["z="  "z="  "z="  "x="  "y="];
val = [ 0.1   0.5   0.9   0.5   0.5];
ebox = [0 1 0 1 0 1];
XF=[]; YF=[]; ZF=[]; VF=[];
for i = 1:length(val)
   [Xm,Xp,Ym,Yp,Zm,Zp] = slice_parallelepiped(dir(i), val(i), ebox, m, m, m);
   Vm = interp3d(Xm,Ym,Zm, tl);
   [xf,yf,zf,vf] = nf3dq(Xm,Ym,Zm,Vm,1);
   XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf]; VF = [VF vf];
   Vp = interp3d(Xp,Yp,Zp, tl);
   [xf,yf,zf,vf] = nf3dq(Xp,Yp,Zp,Vp,1);
   XF = [XF xf]; YF = [YF yf]; ZF = [ZF zf]; VF = [VF vf];
end
nb_col = 128;
vmin = min(VF); vmax = max(VF);
color = dsearch(VF,linspace(vmin,vmax,nb_col+1));
xset("colormap",jetcolormap(nb_col));
xbasc(); xset("hidden3d",xget("background"));
colorbar(vmin,vmax)
plot3d(XF, YF, list(ZF,color), flag=[-1 6 4])
xtitle("3d spline interpolation of "+func)
xselect()
```

See Also

linear_interpn, interp3d, bsplin3val

Authors

R.F. Boisvert, C. De Boor (code from the CMLIB fortran lib)
B. Pincon (scilab interface)

Intersci

Name

intersci — scilab tool to interface C of Fortran functions with scilab

Description

All scilab primitive functions are defined in a set of interface routines. For each function the interfacing code checks first number of rhs and lhs arguments. Then it get pointers on input arguments in the Scilab data base and checks their types. After that it calls procedure associated with Scilab functions, checks returned errors flags and set the results in the data base.

`intersci`\ is a program which permits to interface automatically FORTRAN subroutines or C functions to Scilab

With `intersci`, a user can group all his FORTRAN or C code into a same set, called an interface, and use them in Scilab as Scilab functions. The interfacing is made by creating a FORTRAN subroutine which has to be linked to Scilab together with the user code. This complex FORTRAN subroutine is automatically generated by `intersci`\ from a description file of the interface.

Refer to intersci documentation for more details.

See Also

fort , external , addinter

JVM

Name

javaclasspath — set and get dynamic Java class path

```
res=javaclasspath()
javaclasspath(path)
```

Parameters

res
: a string matrix

Description

set and get the dynamic Java path to one or more directory or file specifications given in path.

Examples

```
res=javaclasspath();
javaclasspath(SCI);
javaclasspath([SCI,SCI+'/java']);
```

Authors

A.C

Name

javalibrarypath — set and get dynamic java.library.path

```
res=javalibrarypath()
javalibrarypath(path)
```

Parameters

res
: a string matrix

Description

set and get the dynamic Java Library path to one or more directory given in path.

When you use java classes with native methods, you need to define path where is dynamic library.

Examples

```
res=javalibrarypath();
javalibrarypath(SCI);
javalibrarypath([SCI,SCI+'/libs']);
```

See Also

javaclasspath

Authors

A.C

Name

jre_path — returns Java Runtime Environment used by Scilab

```
p=jre_path()
```

Parameters

p
: a string path of JRE

Description

returns Java Runtime Environment used by Scilab.

See Also

system_getproperty

Authors

A.C

Name

system_getproperty — gets the system property indicated by a specified key.

```
res=system_getproperty(key)
```

Parameters

res
: a string value

key
: a string

Description

gets the system property indicated by a specified key.

java.version	Java Runtime Environment version
java.vendor	Java Runtime Environment vendor
java.vendor.url	Java vendor URL
java.home	Java installation directory
java.vm.specification.version	Java Virtual Machine specification version
java.vm.specification.vendor	Java Virtual Machine specification vendor
java.vm.specification.name	Java Virtual Machine specification name
java.vm.version	Java Virtual Machine implementation version
java.vm.vendor	Java Virtual Machine implementation vendor
java.vm.name	Java Virtual Machine implementation name
java.specification.version	Java Runtime Environment specification version
java.specification.vendor	Java Runtime Environment specification vendor
java.specification.name	Java Runtime Environment specification name
java.class.version	Java class format version number
java.class.path	Java class path
java.library.path	List of paths to search when loading libraries
java.io.tmpdir	Default temp file path
java.compiler	Name of JIT compiler to use
java.ext.dirs	Path of extension directory or directories
os.name	Operating system name
os.arch	Operating system architecture
os.version	Operating system version
file.separator	File separator ("/" on UNIX)
path.separator	Path separator (":" on UNIX)
line.separator Line separator	("\n" on UNIX)
user.name	User's account name
user.home	User's home directory
user.dir	User's current working directory

Examples

```
system_getproperty('awt.toolkit')
system_getproperty('file.encoding')
system_getproperty('file.encoding.pkg')
system_getproperty('java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment')
system_getproperty('java.awt.printerjob=sun.awt.windows.WPrinterJob')
system_getproperty('java.class.path')
system_getproperty('java.class.version')
system_getproperty('java.endorsed.dirs')
system_getproperty('java.ext.dirs')
system_getproperty('java.home')
system_getproperty('java.io.tmpdir')
system_getproperty('java.library.path')
system_getproperty('java.runtime.name')
system_getproperty('java.runtime.version')
system_getproperty('java.specification.name')
system_getproperty('java.specification.vendor')
system_getproperty('java.specification.version')
system_getproperty('java.vendor')
system_getproperty('java.vendor.url')
system_getproperty('java.vendor.url.bug')
system_getproperty('java.version')
system_getproperty('java.vm.info')
system_getproperty('java.vm.name')
system_getproperty('java.vm.specification.name')
system_getproperty('java.vm.specification.vendor')
system_getproperty('java.vm.specification.version')
system_getproperty('java.vm.vendor')
system_getproperty('java.vm.version')
system_getproperty('line.separator')
system_getproperty('os.arch')
system_getproperty('os.name')
system_getproperty('os.version')
system_getproperty('path.separator')
system_getproperty('sun.arch.data.model')
system_getproperty('sun.boot.class.path')
system_getproperty('sun.boot.library.path')
system_getproperty('sun.cpu.endian')
system_getproperty('sun.cpu.isalist')
system_getproperty('sun.desktop')
system_getproperty('sun.io.unicode.encoding')
system_getproperty('sun.jnu.encoding')
system_getproperty('sun.management.compiler')
system_getproperty('sun.os.patch.level')
system_getproperty('user.country')
system_getproperty('user.dir')
system_getproperty('user.home')
system_getproperty('user.language')
system_getproperty('user.name')
system_getproperty('user.timezone')
system_getproperty('user.variant')
```

Authors

A.C

Name

system_setproperty — set a system property indicated by a specified key and value.

```
prev = system_setproperty(key,value)
```

Parameters

prev
: a string previous value or []

key
: a string

value
: a string

Description

Sets the system property indicated by the specified key.

Warning : change property with precaution.

Examples

```
system_getproperty('myproperty')
  system_setproperty('myproperty','hello')
  system_getproperty('myproperty')
```

Authors

A.C

Name

with_embedded_jre — checks if scilab uses a embedded JRE

```
res=with_embedded_jre()
```

Parameters

res
: a boolean

Description

checks if scilab uses a embedded JRE.

Examples

```
res=with_embedded_jre();
```

Authors

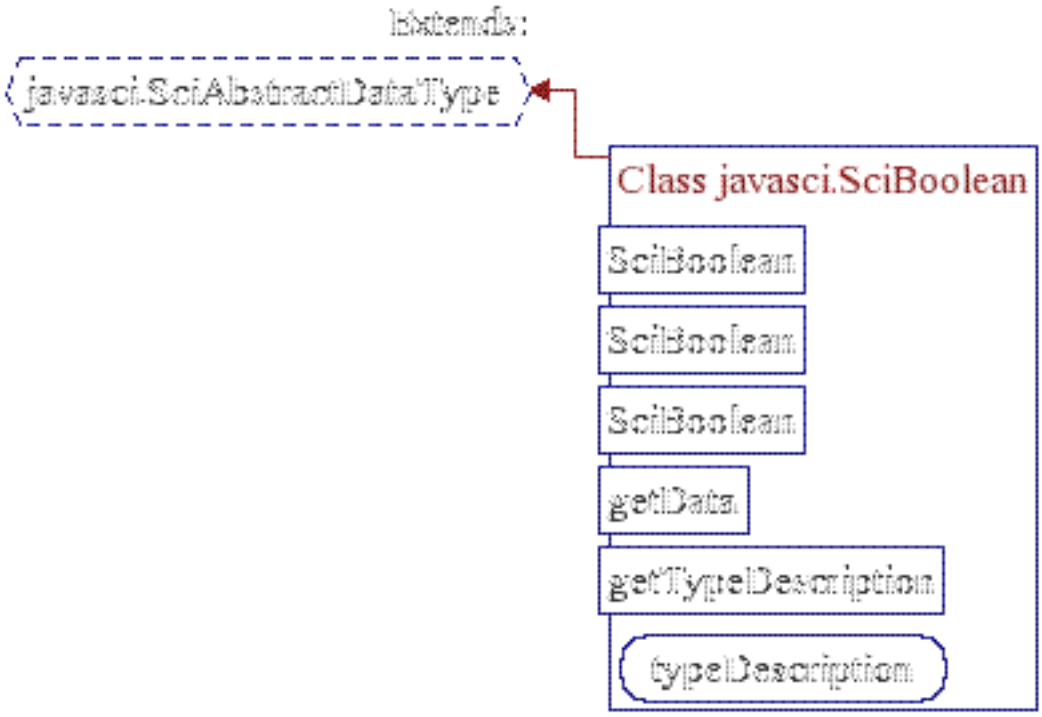
A.C

Java Interface

Name

javasci.SciBoolean — Class to use boolean object with scilab

Description



Method Summary :

`public SciBoolean(String name,SciBoolean Obj)`

`public SciBoolean(String name)` Constructor (if name exists in Scilab and has the same type, variable is imported from Scilab)

`public SciBoolean(String name,boolean Value )`

`public String getName()`Get Name of scilab object

`public boolean getData()`Get Value of scilab object

`public void Get()`Get in java object , value of scilab object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

`public void Send()`Send to scilab object , value of java object

`public void disp()`disp object

Examples

```
// See SCI/modules/javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciBooleanArray , SciDoubleArray , SciString , SciStringArray

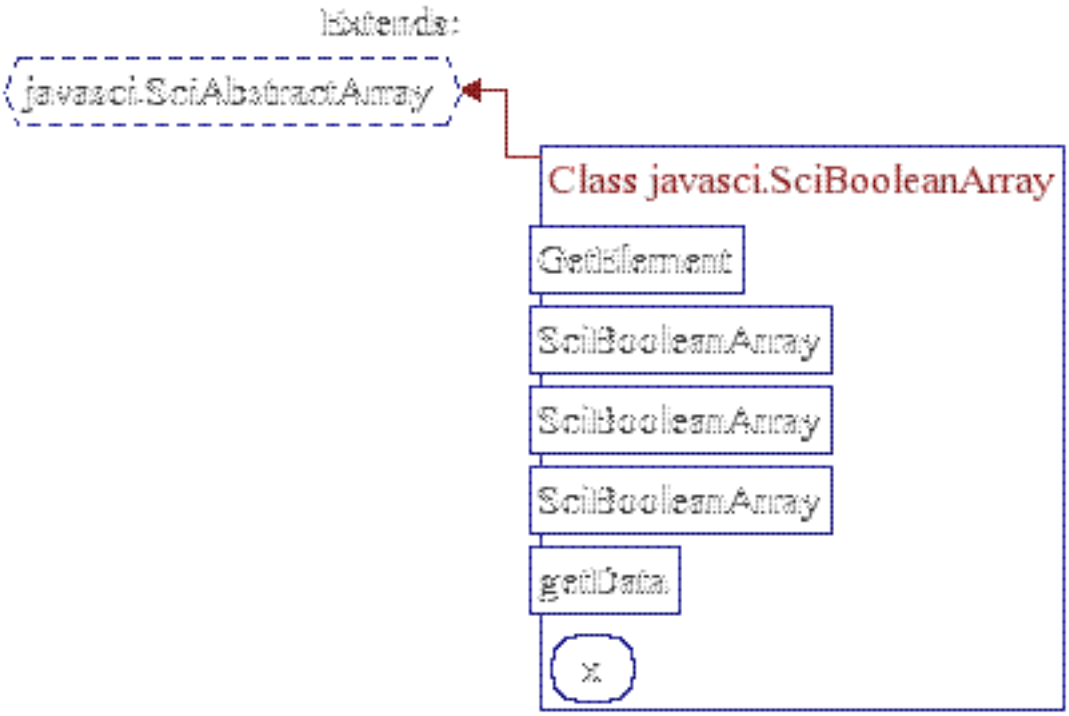
Authors

A.C

Name

javasci.SciBooleanArray — Class to use boolean matrix in Scilab.

Description



Method Summary :

`public SciBooleanArray(String name,SciBooleanArray Obj)`

`public SciBooleanArray(String name,int r,int c)`

`public SciBooleanArray(String name,int r,int c,boolean [] x )`Constructor

`public int getNumbersOfRows()`Get number of rows

`public int getNumbersOfCols()` Get number of colons

`public int getRow()`(deprecated) Get number of rows

`public int getCol()` (deprecated) Get number of colons

`public String getName()`Get Name of scilab object

`public boolean[] getData()` Get Value of scilab object

`public void disp()` disp object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

`public void Get()`Get in java object , value of scilab object

`public void Send()`Send to scilab object , value of java object

`public boolean GetElement(int indr, int indc)`Get a specific element of scilab object

Examples

```
// See SCI/modules/javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciBoolean , SciDouble , SciString , SciStringArray

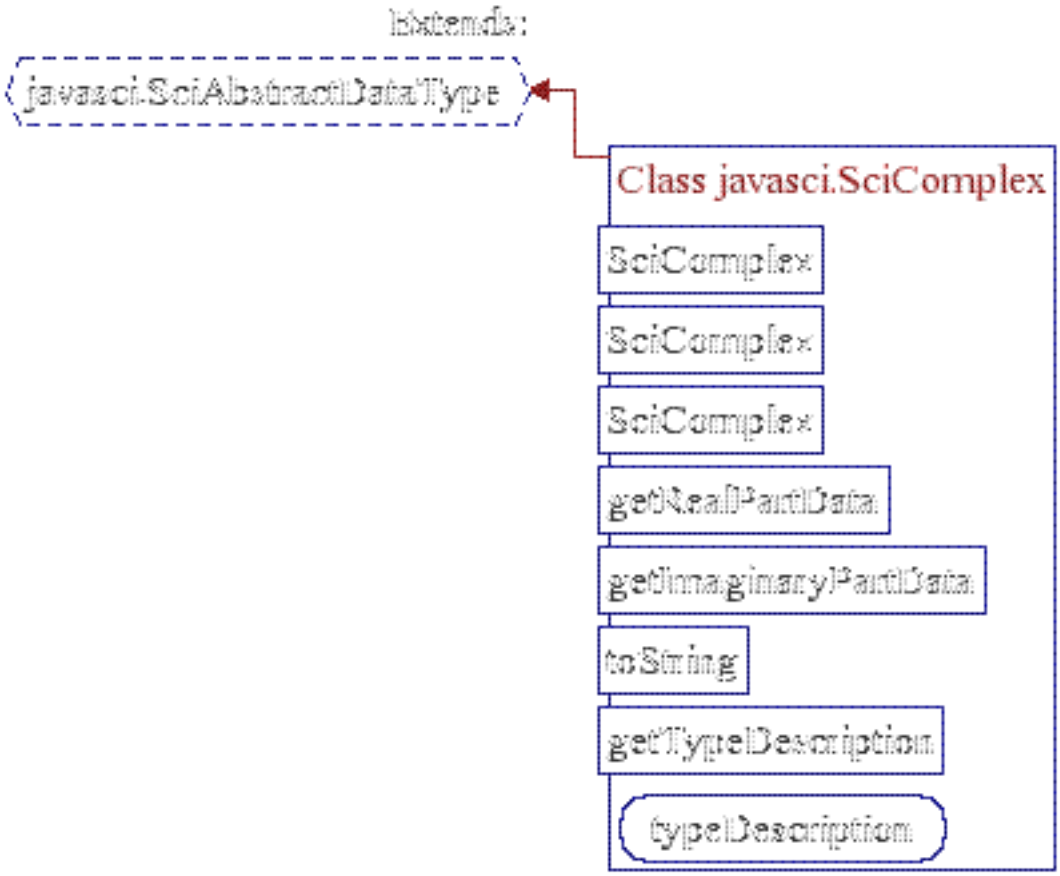
Authors

A.C

Name

javasci.SciComplex — Class to use complex object with scilab

Description



Method Summary :

`public SciComplex(String name,SciComplex Obj)`

`public SciComplex(String name)` Constructor (if name exists in Scilab and has the same type, variable is imported from Scilab)

`public SciComplex(String name,double realpart,double imaginarypart )`Constructor

`public String getName()`Get Name of scilab object

`public double getRealPartData()`Get Real Part Value of scilab object

`public double getImaginaryPartData()`Get Imaginary Part Value of scilab object

`public void Get()`Get in java object , value of scilab object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

`public void Send()`Send to scilab object , value of java object

`public void disp()`disp object

`public void toString()` convert complex to a string

Examples

```
// See SCI/modules/javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciComplexArray , SciString , SciStringArray , SciDoubleArray , SciDouble

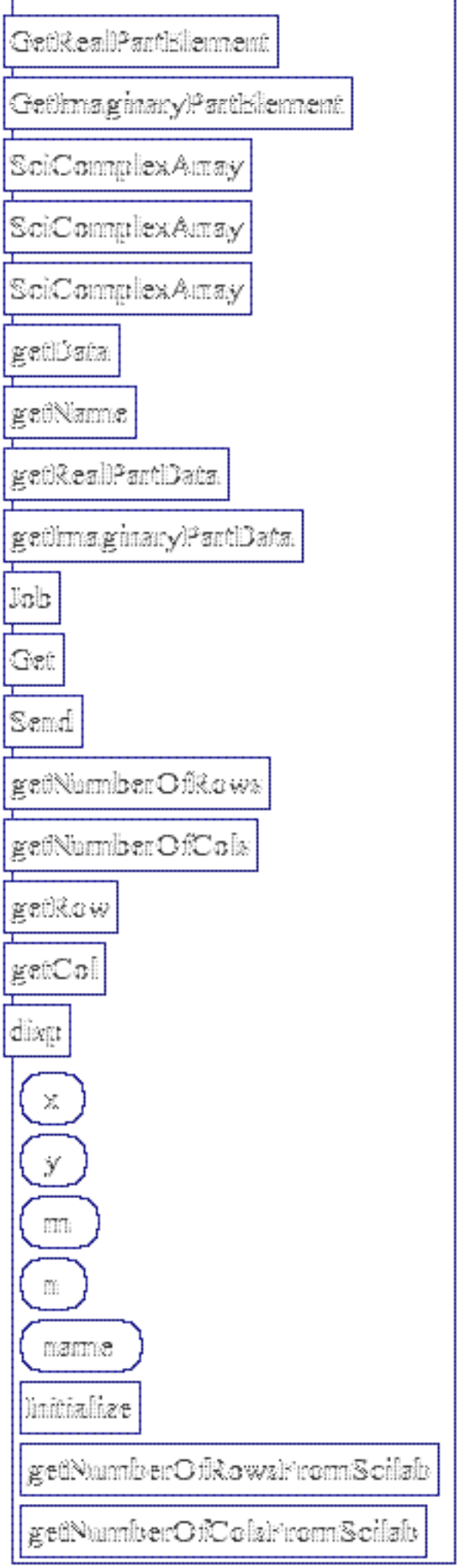
Authors

A.C

Name

javasci.SciComplexArray — Class to use complex matrix in Scilab.

Descr


GetRealPartElement
GetImaginaryPartElement
SciComplexArray
SciComplexArray
SciComplexArray
getData
getName
getRealPartData
getImaginaryPartData
Job
Get
Send
getNumberOfRows
getNumberOfCols
getRow
getCol
disp
x
y
m
n
name
Initialize
getNumberOfRowsFromScilab
getNumberOfColsFromScilab

Method Summary :

`public SciComplexArray(String name,SciComplexArray Obj)`

`public SciComplexArray(String name,int r,int c)`

`public SciComplexArray(String name,int r,int c,double [] realpart,double [] imaginarypart)`Constructor

`public int getNumbersOfRows()`Get number of rows

`public int getNumbersOfCols()` Get number of colons

`public int getRow()`(deprecated) Get number of rows

`public int getCol()` (deprecated) Get number of colons

`public String getName()`Get Name of scilab object

`public double[] getRealPartData()`Get Real Part Value of scilab object

`public double[] getImaginaryPartData()`Get Imaginary Part Value of scilab object

`public void disp()` disp object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

`public void Get()`Get in java object , value of scilab object

`public void Send()`Send to scilab object , value of java object

`public double GetRealPartElement(int indr, int indc)`Get a specific element of scilab object

`public double GetImaginaryPartElement(int indr, int indc)`Get a specific element of scilab object

Examples

```
// See SCI/modules/Javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciComplex , SciDouble , SciDoubleArray , SciString , SciStringArray

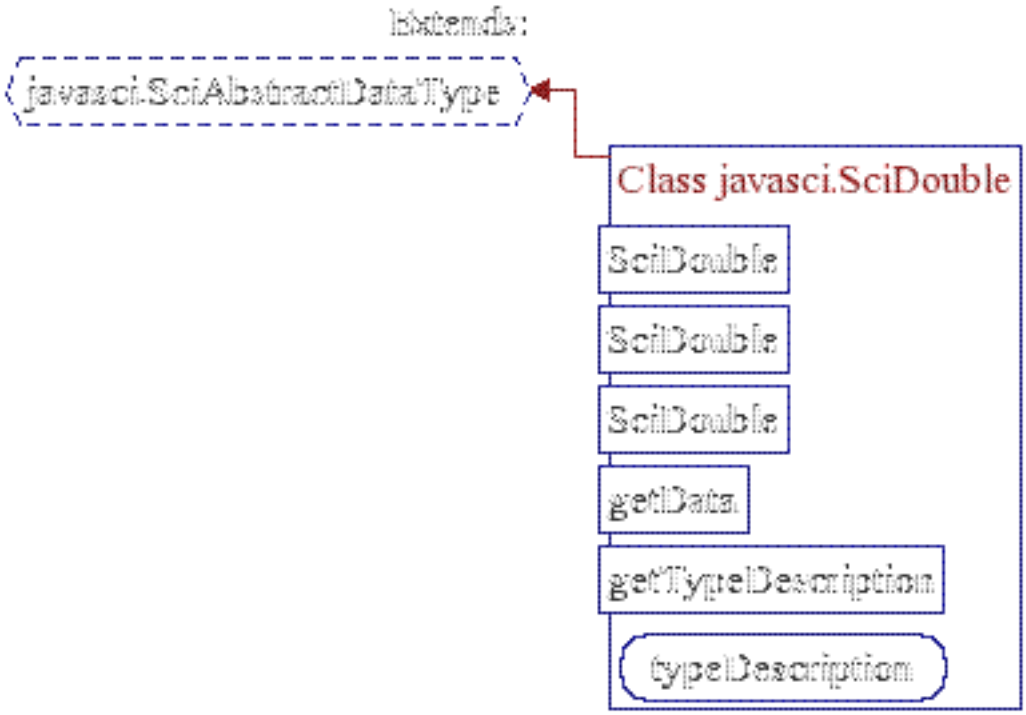
Authors

A.C

Name

javasci.SciDouble — Class to use double object with scilab

Description



Method Summary :

`public SciDouble(String name,SciDouble Obj)`

`public SciDouble(String name)` Constructor (if name exists in Scilab and has the same type, variable is imported from Scilab)

`public SciDouble(String name,double Value )`

`public String getName()`Get Name of scilab object

`public double getData()`Get Value of scilab object

`public void Get()`Get in java object , value of scilab object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

`public void Send()`Send to scilab object , value of java object

`public void disp()`disp object

Examples

```
// See SCI/modules/Javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciDoubleArray , SciString , SciStringArray

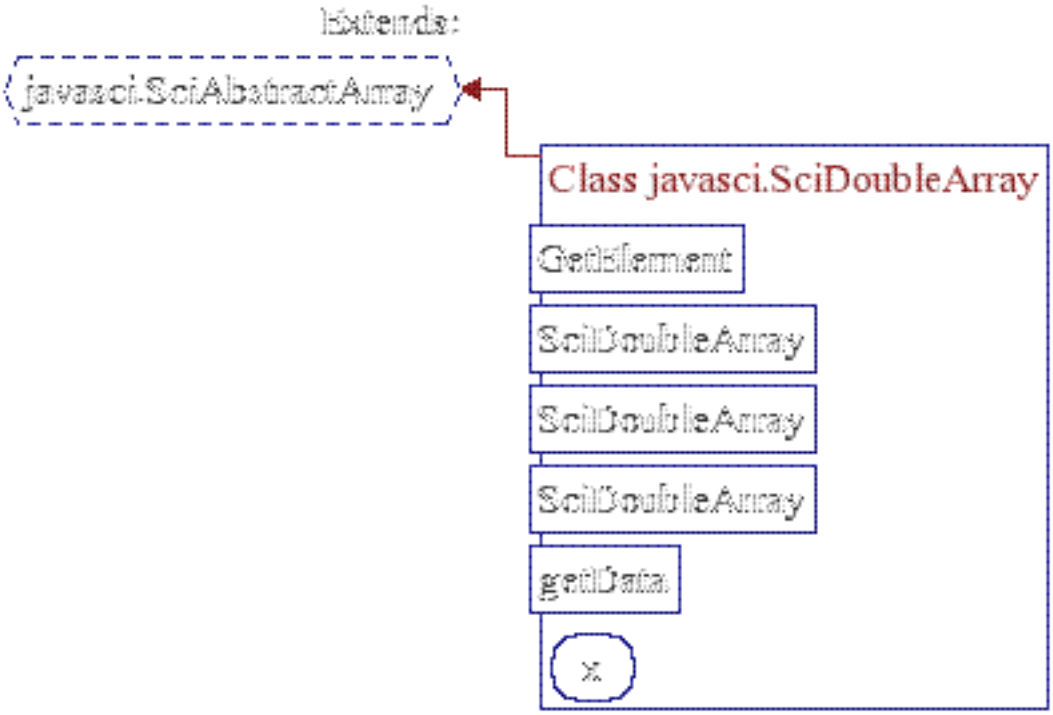
Authors

A.C

Name

javasci.SciDoubleArray — Class to use real matrix in Scilab.

Description



Method Summary :

`public SciDoubleArray(String name,SciDoubleArray Obj)`

`public SciDoubleArray(String name,int r,int c)`

`public SciDoubleArray(String name,int r,int c,double [] x )`Constructor

`public int getNumbersOfRows()`Get number of rows

`public int getNumbersOfCols()` Get number of colons

`public int getRow()`(deprecated) Get number of rows

`public int getCol()` (deprecated) Get number of colons

`public String getName()`Get Name of scilab object

`public double[] getData()` Get Value of scilab object

`public void disp()` disp object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

`public void Get()`Get in java object , value of scilab object

`public void Send()`Send to scilab object , value of java object

`public double GetElement(int indr, int indc)`Get a specific element of scilab object

Examples

```
// See SCI/modules/Javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciDouble , SciString , SciStringArray

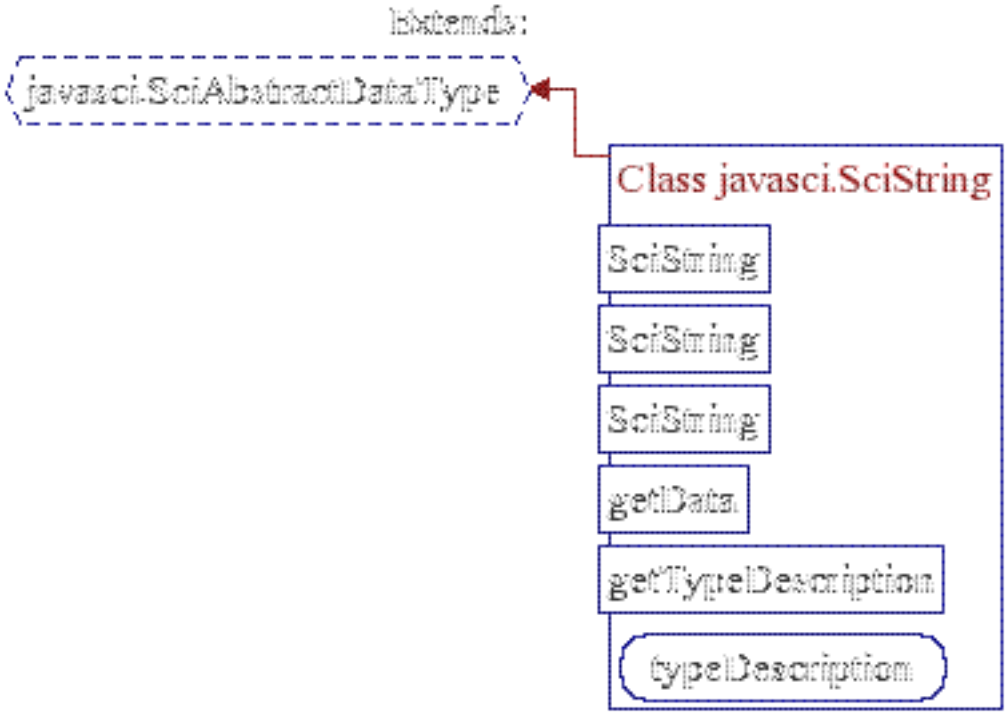
Authors

A.C

Name

javasci.SciString — Map a Java String into a Scilab string.

Description



Method Summary :

`public SciString(String name,SciString Obj)`

`public SciString(String name)` Constructor (if name exists in Scilab and has the same type, variable is imported from Scilab)

`public String getName()`Get Name of scilab object

`public void Get()`Get in java object , value of scilab object

`public String getData()` :Get Value of scilab object

`public void Send()`Send to scilab object , value of java object

`public void disp()` disp object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

Examples

```
// See SCI/modules/Javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciDouble , SciDoubleArray , SciStringArray

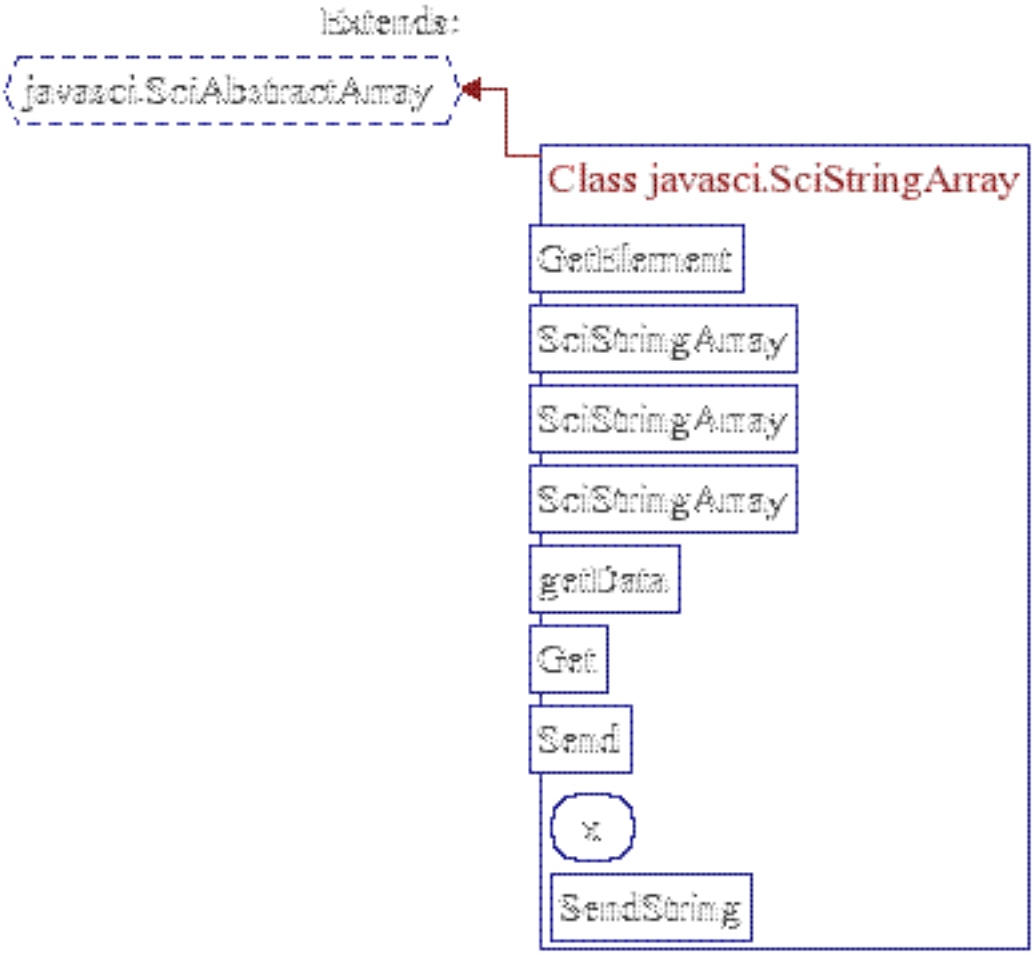
Authors

A.C

Name

javasci.SciStringArray — Classe to use String matrix in Scilab.

Description



Method Summary :

`public SciDoubleArray(String name,SciDoubleArray Obj)`

`public SciStringArray(String name,int r,int c)`

`public SciStringArray(String name,int r,int c,String [] x )`Constructor

`public int getNumbersOfRows()`Get number of rows

`public int getNumbersOfCols()` Get number of colons

`public int getRow()`(deprecated) Get number of rows

`public int getCol()` (deprecated) Get number of colons

`public String getName()`Get Name of scilab object

`public String[] getData()`Get Value of scilab object

`public void Get()` Get in java object , value of scilab object

`public void Send()` Send to scilab object , value of java object

`public void disp()` disp object

`public boolean Job(String job)`(deprecated see Scilab.Exec) Execute a job in scilab

`public String GetElement(int indr, int indc)`Get a specific element of scilab object

Examples

```
// See SCI/modules/Javasci/examples directory
```

See Also

javasci.Scilab , Compile and run with Javasci, SciDouble , SciDoubleArray , SciString

Authors

A.C

Name

javasci.Scilab — This class provides the basic methods to execute Scilab code and scripts.

Description

This class is static. Since the Scilab environnement is persistant, all variables will remain accessible with the Java application.

Method Summary :

- *`public static void Events()`* - Execute a Scilab event
- *`public static boolean HaveAGraph()`* -Check if there is any scilab graphic window open (return True if it is the case).
- *`public static boolean Exec(String job)`* - Execute a job in scilab. return true if there is no error.
- *`public static native boolean ExistVar(String VarName)`* - Detect if VarName exists in Scilab. return true if Varname exist.
- *`public static native int TypeVar(String VarName)`* - Return Scilab type of VarName. See type
- *`public static native int GetLastErrorCode()`* - Return last Error code. See lasterror
- *`public static boolean ExecuteScilabScript(String scilabscriptfilename)`* - Execute a scilab script (.sce) return true if there is no error.
- *`public static native boolean Finish()`* - Terminate scilab (call scilab.quit , close a scilab object)

Examples

```
// A Scilab / Java example
// Filename: ScilabExample.java

import javasci.Scilab;

public class ScilabExample {
 public static void main(String []args){
  String myVar="myVariable";
  Scilab.Exec(myVar+"=(%pi^4)/90;disp(myVariable);"); // Simple display
  if (Scilab.ExistVar(myVar)) {
   System.out.println("Variable "+myVar+" exists. Type: "+Scilab.TypeVar(myVar)
  }
  if (!Scilab.Exec("disp(plop);")) { // Error
   System.err.println("Last error: "+Scilab.GetLastErrorCode()); // Error
  }
  Scilab.Finish();
 }
}
```

See Also

Compile and run with Javasci, SciDouble, SciDoubleArray, SciString, SciStringArray, type, lasterror

Authors

A.C

Name

Compile and run with javasci — How to compile a Java application using Javasci

Description

To compile a Java code based on Javasci, it is only necessary to have javasci.jar defined in the classpath.

For example, with the code defined in the example of this page, the command would be:

```
$ javac -cp $SCI/modules/javasci/jar/javasci.jar BasicExample.java
```

To run Scilab, there are a few other things to set up.

Some global variables must me set:

- SCI - Path to Scilab files
  - Linux/Unix/MacOSX:
    - In the binary version of Scilab, SCI will point to /path/to/scilab/share/scilab/
    - In the source tree of Scilab, SCI will point to the root of the source tree /path/to/scilab/source/tree/
  - Windows
- LD_LIBRARY_PATH - Paths to libscilab.so and libjavasci.so (or .dll, .jnilib...)
  - Linux/Unix/MacOSX:
    - In the binary version of Scilab, SCI will point to /path/to/scilab/lib/scilab/
    - In the source tree of Scilab, SCI will point to the root of the source tree /path/to/scilab/modules/javasci/.libs/ and /path/to/scilab/.libs/

To launch the Java Application, you can either provide them with environnement variable

- LD_LIBRARY_PATH=/path/to/libjavasci/ SCI=/path/to/scilab/ java -cp modules/javasci/jar/javasci.jar:. BasicExample

  or with the arguments

- SCI=/path/to/scilab/ java -Djava.library.path=/path/to/libjavasci/ -cp modules/javasci/jar/javasci.jar:. BasicExample

Note that two environnement variables are taken in account for specific needs:

- `SCI_DISABLE_TK=1` Disables Tk (Tcl's GUI)
- `SCI_JAVA_ENABLE_HEADLESS=1` Launch Java in headless mode (no AWT/Swing)

Examples

```
// A simple Java example
// Filename: BasicExample.java

import javasci.Scilab;

public class BasicExample {
```

```
public static void main(String []args){
Scilab.Exec("disp((%pi^2)/6);");
}
}
```

See Also

Scilab, SciDouble, SciDoubleArray, SciString, SciStringArray

Authors

Allan Cornet

Sylvestre Ledru

Name

javasci — Call Scilab engine from a Java application

Description

Scilab offers the possibility to be called from a Java application.

This help describes the features of the javasci API.

Examples

```
// A simple Java example
// Filename: DisplayPI.java

import javasci.Scilab;

public class DisplayPI {
public static void main(String []args){
Scilab.Exec("disp(%pi);");
}
}
```

See Also

Scilab, Compile and run with Javasci, SciDouble, SciDoubleArray, SciString, SciStringArray

Authors

Allan Cornet

Sylvestre Ledru

Linear Algebra

Name

aff2ab — linear (affine) function to A,b conversion

```
[A,b]=aff2ab(afunction,dimX,D [,flag])
```

Parameters

afunction
: a scilab function `Y =fct(X,D)` where `X, D, Y` are `list` of matrices

dimX
: a p x 2 integer matrix (`p` is the number of matrices in `X`)

D
: a `list` of real matrices (or any other valid Scilab object).

flag
: optional parameter (`flag='f'` or `flag='sp'`)

A
: a real matrix

b
: a real vector having same row dimension as `A`

Description

`aff2ab` returns the matrix representation of an affine function (in the canonical basis).

`afunction` is a function with imposed syntax: `Y=afunction(X,D)` where `X=list(X1,X2,...,Xp)` is a list of p real matrices, and `Y=list(Y1,...,Yq)` is a list of q real real matrices which depend linearly of the `Xi`'s. The (optional) input `D` contains parameters needed to compute Y as a function of X. (It is generally a list of matrices).

`dimX` is a p x 2 matrix: `dimX(i)=[nri,nci]` is the actual number of rows and columns of matrix `Xi`. These dimensions determine `na`, the column dimension of the resulting matrix `A`: `na=nr1*nc1 +...+ nrp*ncp`.

If the optional parameter `flag='sp'` the resulting `A` matrix is returned as a sparse matrix.

This function is useful to solve a system of linear equations where the unknown variables are matrices.

Examples

```
// Lyapunov equation solver (one unknown variable, one constraint)
deff('Y=lyapunov(X,D)','[A,Q]=D(:);Xm=X(:); Y=list(A''*Xm+Xm*A-Q)')
A=rand(3,3);Q=rand(3,3);Q=Q+Q';D=list(A,Q);dimX=[3,3];
[Aly,bly]=aff2ab(lyapunov,dimX,D);
[Xl,kerA]=linsolve(Aly,bly); Xv=vec2list(Xl,dimX); lyapunov(Xv,D)
Xm=Xv(:); A'*Xm+Xm*A-Q

// Lyapunov equation solver with redundant constraint X=X'
// (one variable, two constraints) D is global variable
deff('Y=ly2(X,D)','[A,Q]=D(:);Xm=X(:); Y=list(A''*Xm+Xm*A-Q,Xm''-Xm)')
A=rand(3,3);Q=rand(3,3);Q=Q+Q';D=list(A,Q);dimX=[3,3];
```

```
[Aly,bly]=aff2ab(ly2,dimX,D);
[Xl,kerA]=linsolve(Aly,bly); Xv=vec2list(Xl,dimX); ly2(Xv,D)

// Francis equations
// Find matrices X1 and X2 such that:
// A1*X1 - X1*A2 + B*X2 -A3 = 0
// D1*X1 -D2 = 0
deff('Y=bruce(X,D)','[A1,A2,A3,B,D1,D2]=D(:),...
[X1,X2]=X(:);Y=list(A1*X1-X1*A2+B*X2-A3,D1*X1-D2)')
A1=[-4,10;-1,2];A3=[1;2];B=[0;1];A2=1;D1=[0,1];D2=1;
D=list(A1,A2,A3,B,D1,D2);
[n1,m1]=size(A1);[n2,m2]=size(A2);[n3,m3]=size(B);
dimX=[[m1,n2];[m3,m2]];
[Af,bf]=aff2ab(bruce,dimX,D);
[Xf,KerAf]=linsolve(Af,bf);Xsol=vec2list(Xf,dimX)
bruce(Xsol,D)

// Find all X which commute with A
deff('y=f(X,D)','y=list(D(:)*X(:)-X(:)*D(:))')
A=rand(3,3);dimX=[3,3];[Af,bf]=aff2ab(f,dimX,list(A));
[Xf,KerAf]=linsolve(Af,bf);[p,q]=size(KerAf);
Xsol=vec2list(Xf+KerAf*rand(q,1),dimX);
C=Xsol(:); A*C-C*A
```

See Also

linsolve

Name

balanc — matrix or pencil balancing

```
[Ab,X]=balanc(A)
[Eb,Ab,X,Y]=balanc(E,A)
```

Parameters

A:
: a real square matrix

X:
: a real square invertible matrix

E:
: a real square matrix (same dimension as A)

Y:
: a real square invertible matrix.

Description

Balance a square matrix to improve its condition number.

`[Ab,X] = balanc(A)` finds a similarity transformation `X` such that

`Ab = inv(X)*A*X` has approximately equal row and column norms.

For matrix pencils,balancing is done for improving the generalized eigenvalue problem.

`[Eb,Ab,X,Y] = balanc(E,A)` returns left and right transformations `X` and `Y` such that `Eb=inv(X)*E*Y, Ab=inv(X)*A*Y`

Remark

Balancing is made in the functions `bdiag` and `spec`.

Examples

```
A=[1/2^10,1/2^10;2^10,2^10];
[Ab,X]=balanc(A);
norm(A(1,:))/norm(A(2,:))
norm(Ab(1,:))/norm(Ab(2,:))
```

See Also

bdiag , spec , schur

Name

bdiag — block diagonalization, generalized eigenvectors

```
[Ab [,X [,bs]]]=bdiag(A [,rmax])
```

Parameters

A
: real or complex square matrix

rmax
: real number

Ab
: real or complex square matrix

X
: real or complex non-singular matrix

bs
: vector of integers

Description

```
[Ab [,X [,bs]]]=bdiag(A [,rmax])
```

performs the block-diagonalization of matrix `A`. bs gives the structure of the blocks (respective sizes of the blocks). `X` is the change of basis i.e `Ab = inv(X)*A*X`is block diagonal.

`rmax` controls the conditioning of `X`; the default value is the l1 norm of `A`.

To get a diagonal form (if it exists) choose a large value for `rmax` (`rmax=1/%eps` for example). Generically (for real random A) the blocks are (1x1) and (2x2) and `X` is the matrix of eigenvectors.

Examples

```
//Real case: 1x1 and 2x2 blocks
a=rand(5,5);[ab,x,bs]=bdiag(a);ab
//Complex case: complex 1x1 blocks
[ab,x,bs]=bdiag(a+%i*0);ab
```

See Also

schur , sylv , spec

Name

chfact — sparse Cholesky factorization

```
spcho=chfact(A)
```

Parameters

A
: square symmetric positive sparse matrix

spcho
: list containing the Cholesky factors in coded form

Description

`spcho=chfact(A)` computes the sparse Cholesky factors of sparse matrix `A`, assumed symmetric positive definite. This function is based on the Ng-Peyton programs (ORNL). See the Fortran programs for a complete description of the variables in `spcho`. This function is to be used with `chsolve`.

See Also

chsolve , sparse , lufact , luget , spchol

Name

chol — Cholesky factorization

```
[R]=chol(X)
```

Parameters

X
: a symmetric positive definite real or complex matrix.

Description

If `X` is positive definite, then `R = chol(X)` produces an upper triangular matrix `R` such that `R'*R = X`.

`chol(X)` uses only the diagonal and upper triangle of `X`. The lower triangular is assumed to be the (complex conjugate) transpose of the upper.

References

Cholesky decomposition is based on the Lapack routines DPOTRF for real matrices and ZPOTRF for the complex case.

Examples

```
W=rand(5,5)+%i*rand(5,5);
X=W*W';
R=chol(X);
norm(R'*R-X)
```

See Also

spchol , qr , svd , bdiag , fullrf

Name

chsolve — sparse Cholesky solver

```
sol=chsolve(spcho,rhs)
```

Parameters

spcho
: list containing the Cholesky factors in coded form returned by chfact

rhs, sol
: full column vectors

Description

`sol=chsolve(spcho,rhs)` computes the solution of `rhs=A*sol`, with `A` a symmetric sparse positive definite matrix. This function is based on the Ng-Peyton programs (ORNL). See the Fortran programs for a complete description of the variables in `spcho`.

Examples

```
A=sprand(20,20,0.1);
A=A*A'+eye();
spcho=chfact(A);
sol=(1:20)';rhs=A*sol;
spcho=chfact(A);
chsolve(spcho,rhs)
```

See Also

chfact , sparse , lufact , luget , spchol

Name

classmarkov — recurrent and transient classes of Markov matrix

```
[perm,rec,tr,indsRec,indsT]=classmarkov(M)
```

Parameters

M
: real N x N Markov matrix. Sum of entries in each row should add to one.

perm
: integer permutation vector.

rec, tr
: integer vector, number (number of states in each recurrent classes, number of transient states).

indsRec,indsT
: integer vectors. (Indexes of recurrent and transient states).

Description

Returns a permutation vector `perm` such that

```
M(perm,perm) = [M11 0 0 0 0   0]
               [0 M22 0 0     0]
               [0 0 M33       0]
               [      ...      ]
               [0 0       Mrr 0]
               [* *        *  Q]
```

Each `Mii` is a Markov matrix of dimension `rec(i)  i=1,..,r`. `Q` is sub-Markov matrix of dimension `tr`. States 1 to sum(rec) are recurrent and states from r+1 to n are transient. One has `perm=[indsRec,indsT]` where indsRec is a vector of size sum(rec) and indsT is a vector of size tr.

Examples

```
//P has two recurrent classes (with 2 and 1 states) 2 transient states
P=genmarkov([2,1],2,'perm')
[perm,rec,tr,indsRec,indsT]=classmarkov(P);
P(perm,perm)
```

See Also

genmarkov , eigenmarkov

Name

cmb_lin — symbolic linear combination

```
[x]=cmb_lin(alfa,x,beta,y)
```

Description

Evaluates `alfa*x-beta*y`. `alfa, beta, x, y` are character strings. (low-level routine)

See Also

mulf , addf

Name

coff — resolvent (cofactor method)

```
[N,d]=coff(M [,var])
```

Parameters

M
: square real matrix

var
: character string

N
: polynomial matrix (same size as `M`)

d
: polynomial (characteristic polynomial `poly(A,'s')`)

Description

`coff` computes R=`(s*eye()-M)^-1` for `M` a real matrix. R is given by `N/d`.

`N` = numerator polynomial matrix.

`d` = common denominator.

`var` character string ('`s`' if omitted)

Examples

```
M=[1,2;0,3];
[N,d]=coff(M)
N/d
inv(%s*eye()-M)
```

See Also

coffg , ss2tf , nlev , poly

Name

colcomp — column compression, kernel, nullspace

```
[W,rk]=colcomp(A [,flag] [,tol])
```

Parameters

A
: real or complex matrix

flag
: character string

tol
: real number

W
: square non-singular matrix (change of basis)

rk
: integer (rank of A)

Description

Column compression of A: `Ac = A*W` is column compressed i.e

`Ac=[0,Af]` with `Af` full column rank, rank(`Af`) = rank(A) = `rk`.

`flag` and `tol` are optional parameters: `flag = 'qr'` or `'svd'` (default is `'svd'`).

`tol` = tolerance parameter (of order `%eps` as default value).

The `ma-rk` first columns of `W` span the kernel of `A` when `size(A)=(na,ma)`

Examples

```
A=rand(5,2)*rand(2,5);
[X,r]=colcomp(A);
norm(A*X(:,1:$-r),1)
```

See Also

rowcomp , fullrf , fullrfk , kernel

Authors

F.D.;

Name

companion — companion matrix

```
A=companion(p)
```

Parameters

p
: polynomial or vector of polynomials

A
: square matrix

Description

Returns a matrix A with characteristic polynomial equal to p if p is monic. If p is not monic the characteristic polynomial of A is equal to p/c where c is the coefficient of largest degree in p.

If p is a vector of monic polynomials, A is block diagonal, and the characteristic polynomial of the ith block is p(i).

Examples

```
s=poly(0,'s');
p=poly([1,2,3,4,1],'s','c')
det(s*eye()-companion(p))
roots(p)
spec(companion(p))
```

See Also

spec , poly , randpencil

Authors

F.D.

Name

cond — condition number

```
cond(X)
```

Parameters

X
: real or complex square matrix

Description

Condition number in 2-norm. `cond(X)` is the ratio of the largest singular value of `X` to the smallest.

Examples

```
A=testmatrix('hilb',6);
cond(A)
```

See Also

rcond , svd

Name

det — determinant

```
det(X)
[e,m]=det(X)
```

Parameters

X
: real or complex square matrix, polynomial or rational matrix.

m
: real or complex number, the determinant base 10 mantissae

e
: integer, the determinant base 10 exponent

Description

`det(X)` (`m*10^e` is the determinant of the square matrix `X`.

For polynomial matrix `det(X)` is equivalent to `determ(X)`.

For rational matrices `det(X)` is equivalent to `detr(X)`.

References

det computations are based on the Lapack routines DGETRF for real matrices and ZGETRF for the complex case.

Examples

```
x=poly(0,'x');
det([x,1+x;2-x,x^2])
w=ssrand(2,2,4);roots(det(systmat(w))),trzeros(w)   //zeros of linear system
A=rand(3,3);
det(A), prod(spec(A))
```

See Also

detr , determ

Name

eigenmarkov — normalized left and right Markov eigenvectors

```
[M,Q]=eigenmarkov(P)
```

Parameters

P
: real N x N Markov matrix. Sum of entries in each row should add to one.

M
: real matrix with N columns.

Q
: real matrix with N rows.

Description

Returns normalized left and right eigenvectors associated with the eigenvalue 1 of the Markov transition matrix P. If the multiplicity of this eigenvalue is m and P is N x N, M is a m x N matrix and Q a N x m matrix. M(k,:) is the probability distribution vector associated with the kth ergodic set (recurrent class). M(k,x) is zero if x is not in the k-th recurrent class. Q(x,k) is the probability to end in the k-th recurrent class starting from x. If `P^k` converges for large `k` (no eigenvalues on the unit circle except 1), then the limit is `Q*M` (eigenprojection).

Examples

```
//P has two recurrent classes (with 2 and 1 states) 2 transient states
P=genmarkov([2,1],2)
[M,Q]=eigenmarkov(P);
P*Q-Q
Q*M-P^20
```

See Also

genmarkov , classmarkov

Name

ereduc — computes matrix column echelon form by qz transformations

```
[E,Q,Z [,stair [,rk]]]=ereduc(X,tol)
```

Parameters

X
: m x n matrix with real entries.

tol
: real positive scalar.

E
: column echelon form matrix

Q
: m x m unitary matrix

Z
: n x n unitary matrix

stair
: vector of indexes,

  * `ISTAIR(i) = + j` if the boundary element `E(i,j)` is a corner point.
  * `ISTAIR(i) = - j` if the boundary element `E(i,j)` is not a corner point.

  `(i=1,...,M)`

rk
: integer, estimated rank of the matrix

Description

Given an `m x n` matrix `X` (not necessarily regular) the function ereduc computes a unitary transformed matrix `E=Q*X*Z` which is in column echelon form (trapezoidal form). Furthermore the rank of matrix `X` is determined.

Examples

```
X=[1 2 3;4 5 6]
[E,Q,Z ,stair ,rk]=ereduc(X,1.d-15)
```

See Also

fstair

Authors

Th.G.J. Beelen (Philips Glass Eindhoven). SLICOT

Name

expm — square matrix exponential

```
expm(X)
```

Parameters

X
: square matrix with real or complex entries.

Description

`X` is a square matrix `expm(X)` is the matrix

```
expm(X) = I + X + X^2 /2 + ...
```

The computation is performed by first block-diagonalizing `X` and then applying a Pade approximation on each block.

Examples

```
X=[1 2;3 4]
expm(X)
logm(expm(X))
```

See Also

logm , bdiag , coff , log , exp

Name

fstair — computes pencil column echelon form by qz transformations

```
[AE,EE,QE,ZE,blcks,muk,nuk,muk0,nuk0,mnei]=fstair(A,E,Q,Z,stair,rk,tol)
```

Parameters

A
: m x n matrix with real entries.

tol
: real positive scalar.

E
: column echelon form matrix

Q
: m x m unitary matrix

Z
: n x n unitary matrix

stair
: vector of indexes (see ereduc)

rk
: integer, estimated rank of the matrix

AE
: m x n matrix with real entries.

EE
: column echelon form matrix

QE
: m x m unitary matrix

ZE
: n x n unitary matrix

nblcks
: :is the number of submatrices having full row rank >= 0 detected in matrix A.

muk:
: integer array of dimension (n). Contains the column dimensions mu(k) (k=1,...,nblcks) of the submatrices having full column rank in the pencil sE(eps)-A(eps)

nuk:
: integer array of dimension (m+1). Contains the row dimensions nu(k) (k=1,...,nblcks) of the submatrices having full row rank in the pencil sE(eps)-A(eps)

muk0:
: integer array of dimension (n). Contains the column dimensions mu(k) (k=1,...,nblcks) of the submatrices having full column rank in the pencil sE(eps,inf)-A(eps,inf)

nuk:
: integer array of dimension (m+1). Contains the row dimensions nu(k) (k=1,...,nblcks) of the submatrices having full row rank in the pencil sE(eps,inf)-A(eps,inf)

mnei:
integer array of dimension (4). mnei(1) = row dimension of sE(eps)-A(eps)

Description

Given a pencil `sE-A` where matrix `E` is in column echelon form the function `fstair` computes according to the wishes of the user a unitary transformed pencil `QE(sEE-AE)ZE` which is more or less similar to the generalized Schur form of the pencil `sE-A`. The function yields also part of the Kronecker structure of the given pencil.

`Q,Z` are the unitary matrices used to compute the pencil where E is in column echelon form (see ereduc)

See Also

quaskro , ereduc

Authors

Th.G.J. Beelen (Philips Glass Eindhoven). SLICOT

Name

fullrf — full rank factorization

```
[Q,M,rk]=fullrf(A,[tol])
```

Parameters

A
: real or complex matrix

tol
: real number (threshold for rank determination)

Q,M
: real or complex matrix

rk
: integer (rank of A)

Description

Full rank factorization : `fullrf` returns `Q` and `M` such that `A = Q*M` with range(`Q`)=range(`A`) and ker(`M`)=ker(`A`), `Q` full column rank , `M` full row rank, `rk = rank(A) = #columns(Q) = #rows(M)`.

`tol` is an optional real parameter (default value is `sqrt(%eps)`). The rank `rk` of `A` is defined as the number of singular values larger than `norm(A)*tol`.

If A is symmetric, `fullrf` returns `M=Q'`.

Examples

```
A=rand(5,2)*rand(2,5);
[Q,M]=fullrf(A);
norm(Q*M-A,1)
[X,d]=rowcomp(A);Y=X';
svd([A,Y(:,1:d),Q])        //span(Q) = span(A) = span(Y(:,1:2))
```

See Also

svd , qr , fullrfk , rowcomp , colcomp

Authors

F.D.;

Name

fullrfk — full rank factorization of A^k

```
[Bk,Ck]=fullrfk(A,k)
```

Parameters

A
: real or complex matrix

k
: integer

Bk,Ck
: real or complex matrices

Description

This function computes the full rank factorization of `A^k` i.e. `Bk*Ck=A^k` where `Bk` is full column rank and `Ck` full row rank. One has range(`Bk`)=range(`A^k`) and ker(`Ck`)=ker(`A^k`).

For `k=1`, `fullrfk` is equivalent to `fullrf`.

Examples

```
A=rand(5,2)*rand(2,5);[Bk,Ck]=fullrfk(A,3);
norm(Bk*Ck-A^3,1)
```

See Also

fullrf , range

Authors

F.D (1990);

Name

genmarkov — generates random markov matrix with recurrent and transient classes

```
M=genmarkov(rec,tr)
M=genmarkov(rec,tr,flag)
```

Parameters

rec
: integer row vector (its dimension is the number of recurrent classes).

tr
: integer (number of transient states)

M
: real Markov matrix. Sum of entries in each row should add to one.

flag
: string `'perm'`. If given, a random permutation of the states is done.

Description

Returns in M a random Markov transition probability matrix with `size(rec,1)` recurrent classes with `rec(1),...rec($)` entries respectively and tr transient states.

Examples

```
//P has two recurrent classes (with 2 and 1 states) 2 transient states
P=genmarkov([2,1],2,'perm')
[perm,rec,tr,indsRec,indsT]=classmarkov(P);
P(perm,perm)
```

See Also

classmarkov , eigenmarkov

Name

givens — Givens transformation

```
U=givens(xy)
U=givens(x,y)
[U,c]=givens(xy)
[U,c]=givens(x,y)
```

Parameters

x,y
: two real or complex numbers

xy
: real or complex size 2 column vector

U
: 2x2 unitary matrix

c
: real or complex size 2 column vector

Description

`U = givens(x, y)` or `U = givens(xy)` with `xy = [x;y]` returns a 2x2 unitary matrix `U` such that:

`U*xy=[r;0]=c`.

Note that `givens(x,y)` and `givens([x;y])` are equivalent.

Examples

```
A=[3,4;5,6];
U=givens(A(:,1));
U*A
```

See Also

qr

Name

glever — inverse of matrix pencil

```
[Bfs,Bis,chis]=glever(E,A [,s])
```

Parameters

E, A
: two real square matrices of same dimensions

s
: character string (default value 's')

Bfs,Bis
: two polynomial matrices

chis
: polynomial

Description

Computation of

`(s*E-A)^-1`

by generalized Leverrier's algorithm for a matrix pencil.

```
(s*E-A)^-1 = (Bfs/chis) - Bis.
```

`chis` = characteristic polynomial (up to a multiplicative constant).

`Bfs` = numerator polynomial matrix.

`Bis` = polynomial matrix (- expansion of `(s*E-A)^-1` at infinity).

Note the - sign before `Bis`.

Caution

This function uses `cleanp` to simplify `Bfs,Bis` and `chis`.

Examples

```
s=%s;F=[-1,s,0,0;0,-1,0,0;0,0,s-2,0;0,0,0,s-1];
[Bfs,Bis,chis]=glever(F)
inv(F)-((Bfs/chis) - Bis)
```

See Also

rowshuff , det , invr , coffg , pencan , penlaur

Authors

F. D. (1988)

Name

gschur — generalized Schur form (obsolete).

```
[As,Es]=gschur(A,E)
[As,Es,Q,Z]=gschur(A,E)
[As,Es,Z,dim] = gschur(A,E,flag)
[As,Es,Z,dim]= gschur(A,E,extern)
```

Description

This function is obsolete and is now included in the `schur` function. In most cases the `gschur` function will still work as before, but it will be removed in the future release.

The first three syntaxes can be replaced by

```
[As,Es]=schur(A,E)
[As,Es,Q,Z]=schur(A,E);Q=Q' //NOTE THE TRANPOSITION HERE
[As,Es,Z,dim] = schur(A,E,flag)
```

The last syntax requires little more adaptations:

if
: extern is a scilab function the new calling sequence should be `[As,Es,Z,dim]= schur(A,E,Nextern)` with Nextern defined as follow:

  ```
  function t=Nextern(R)
  if R(2)==0 then
    t=extern([1,R(1),R(3)])==1
  else
    c=(R(1)+%i*R(2))/R(3)
    t=extern([2,real(c+c'),real(c*c')])==1
  end
  endfunction
  ```

if
: extern is the name of an external function coded in Fortran or C the new calling sequence should be `[As,Es,Z,dim]= schur(A,E,'nextern')` with nextern defined as follow:

  ```
     logical function nextern(ar,ai,beta)
     double precision ar,ai,beta
     integer r,extern
     if (ai.eq.0.0d0) then
       r=extern(1,ar,beta,0.0d0,0.0d0)
     else
       r=extern(2,0.0d0,0.0d0,2.0d0*ar,ar*ar+ai*ai)
     endif
     nextern=r.eq.1
     end
  ```

See Also

external , schur

Name

gspec — eigenvalues of matrix pencil (obsolete)

```
[al,be]=gspec(A,E)
[al,be,Z]=gspec(A,E)
```

Description

This function is now included in the `spec` function. the calling syntax must be replaced by

```
[al,be]=spec(A,E)
[al,be,Z]=spec(A,E)
```

See Also

spec

Name

hess — Hessenberg form

```
H = hess(A)
[U,H] = hess(A)
```

Parameters

A
: real or complex square matrix

H
: real or complex square matrix

U
: orthogonal or unitary square matrix

Description

`[U,H] = hess(A)` produces a unitary matrix `U` and a Hessenberg matrix `H` so that `A = U*H*U'` and `U'*U` = Identity. By itself, `hess(A)` returns `H`.

The Hessenberg form of a matrix is zero below the first subdiagonal. If the matrix is symmetric or Hermitian, the form is tridiagonal.

References

hess function is based on the Lapack routines DGEHRD, DORGHR for real matrices and ZGEHRD, ZORGHR for the complex case.

Examples

```
A=rand(3,3);[U,H]=hess(A);
and( abs(U*H*U'-A)<1.d-10 )
```

See Also

qr , contr , schur

Used Functions

`hess` function is based on the Lapack routines DGEHRD, DORGHR for real matrices and ZGEHRD, ZORGHR for the complex case.

Name

householder — Householder orthogonal reflexion matrix

```
u=householder(v [,w])
```

Parameters

v
: real or complex column vector

w
: real or complex column vector with same size as `v`. Default value is `eye(v)`

u
: real or complex column vector

Description

given 2 column vectors `v`, `w` of same size, `householder(v,w)` returns a unitary column vector `u`, such that `(eye()-2*u*u')*v` is proportional to `w`. `(eye()-2*u*u')` is the orthogonal Householder reflexion matrix .

`w` default value is `eye(v)`. In this case vector `(eye()-2*u*u')*v` is the vector `eye(v)*norm(v)`.

See Also

qr , givens

Name

im_inv — inverse image

```
[X,dim]=im_inv(A,B [,tol])
[X,dim,Y]=im_inv(A,B, [,tol])
```

Parameters

A,B
: two real or complex matrices with equal number of columns

X
: orthogonal or unitary square matrix of order equal to the number of columns of A

dim
: integer (dimension of subspace)

Y
: orthogonal matrix of order equal to the number of rows of A and B.

Description

`[X,dim]=im_inv(A,B)` computes `(A^-1)(B)` i.e vectors whose image through A are in range(B)

The `dim` first columns of X span `(A^-1)(B)`

`tol` is a threshold used to test if subspace inclusion; default value is `tol = 100*%eps`. If Y is returned, then `[Y*A*X,Y*B]` is partitioned as follows: `[A11,A12;0,A22],[B1;0]`

where B1 has full row rank (equals `rank(B)`) and A22 has full column rank and has `dim` columns.

Examples

```
A=[rand(2,5);[zeros(3,4),rand(3,1)]];B=[[1,1;1,1];zeros(3,2)];
W=rand(5,5);A=W*A;B=W*B;
[X,dim]=im_inv(A,B)
svd([A*X(:,1:dim),B])   //vectors A*X(:,1:dim) belong to range(B)
[X,dim,Y]=im_inv(A,B);[Y*A*X,Y*B]
```

See Also

rowcomp , spaninter , spanplus , linsolve

Authors

F. Delebecque INRIA

Name

inv — matrix inverse

```
inv(X)
```

Parameters

X
: real or complex square matrix, polynomial matrix, rational matrix in transfer or state-space representation.

Description

`inv(X)` is the inverse of the square matrix `X`. A warning message is printed if `X` is badly scaled or nearly singular.

For polynomial matrices or rational matrices in transfer representation, `inv(X)` is equivalent to `invr(X)`.

For linear systems in state-space representation (`syslin` list), `invr(X)` is equivalent to `invsyslin(X)`.

References

`inv` function for matrices of numbers is based on the Lapack routines DGETRF, DGETRI for real matrices and ZGETRF, ZGETRI for the complex case. For polynomial matrix and rational function matrix `inv` is based on the `invr` Scilab function.

Examples

```
A=rand(3,3);inv(A)*A
//
x=poly(0,'x');
A=[x,1,x;x^2,2,1+x;1,2,3];inv(A)*A
//
A=[1/x,2;2+x,2/(1+x)]
inv(A)*A
//
A=ssrand(2,2,3);
W=inv(A)*A
clean(ss2tf(W))
```

See Also

slash , backslash , pinv , qr , lufact , lusolve , invr , coff , coffg

Name

kernel — kernel, nullspace

```
W=kernel(A [,tol,[,flag])
```

Parameters

A
: full real or complex matrix or real sparse matrix

flag
: character string `'svd'` (default) or `'qr'`

tol
: real number

W
: full column rank matrix

Description

`W=kernel(A)` returns the kernel (nullspace) of `A`. If A has full column rank then an empty matrix [] is returned.

`flag` and `tol` are optional parameters: `flag = 'qr'` or `'svd'` (default is `'svd'`).

`tol` = tolerance parameter (of order `%eps` as default value).

Examples

```
A=rand(3,1)*rand(1,3);
A*kernel(A)
A=sparse(A);
clean(A*kernel(A))
```

See Also

colcomp , fullrf , fullrfk , linsolve

Authors

F.D.;

Name

kroneck — Kronecker form of matrix pencil

```
[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck(F)
[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck(E,A)
```

Parameters

F
: real matrix pencil `F=s*E-A`

E,A
: two real matrices of same dimensions

Q,Z
: two square orthogonal matrices

Qd,Zd
: two vectors of integers

numbeps,numeta
: two vectors of integers

Description

Kronecker form of matrix pencil: `kroneck` computes two orthogonal matrices `Q, Z` which put the pencil `F=s*E -A` into upper-triangular form:

```
           | sE(eps)-A(eps) |        X       |      X     |      X         |
           |----------------|----------------|------------|----------------|
           |        0       | sE(inf)-A(inf) |      X     |      X         |
Q(sE-A)Z = |--------------------------------|----------------------------|
           |                |                |            |                |
           |        0       |       0        | sE(f)-A(f) |      X         |
           |-------------------------------------------------------------|
           |                |                |            |                |
           |        0       |       0        |      0     | sE(eta)-A(eta)|
```

The dimensions of the four blocks are given by:

`eps=Qd(1) x Zd(1)`, `inf=Qd(2) x Zd(2)`, `f = Qd(3) x Zd(3)`, `eta=Qd(4)xZd(4)`

The `inf` block contains the infinite modes of the pencil.

The `f` block contains the finite modes of the pencil

The structure of epsilon and eta blocks are given by:

`numbeps(1)` = # of eps blocks of size 0 x 1

`numbeps(2)` = # of eps blocks of size 1 x 2

`numbeps(3)` = # of eps blocks of size 2 x 3 etc...

`numbeta(1)` = # of eta blocks of size 1 x 0

`numbeta(2)` = # of eta blocks of size 2 x 1

`numbeta(3)` = # of eta blocks of size 3 x 2 etc...

The code is taken from T. Beelen (Slicot-WGS group).

Examples

```
F=randpencil([1,1,2],[2,3],[-1,3,1],[0,3]);
Q=rand(17,17);Z=rand(18,18);F=Q*F*Z;
//random pencil with eps1=1,eps2=1,eps3=1; 2 J-blocks @ infty
//with dimensions 2 and 3
//3 finite eigenvalues at -1,3,1 and eta1=0,eta2=3
[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck(F);
[Qd(1),Zd(1)]    //eps. part is sum(epsi) x (sum(epsi) + number of epsi)
[Qd(2),Zd(2)]    //infinity part
[Qd(3),Zd(3)]    //finite part
[Qd(4),Zd(4)]    //eta part is (sum(etai) + number(eta1)) x sum(etai)
numbeps
numbeta
```

See Also

gschur , gspec , systmat , pencan , randpencil , trzeros

Name

linsolve — linear equation solver

```
[x0,kerA]=linsolve(A,b [,x0])
```

Parameters

A
: a `na x ma` real matrix (possibly sparse)

b
: a `na x 1` vector (same row dimension as `A`)

x0
: a real vector

kerA
: a `ma x k` real matrix

Description

`linsolve` computes all the solutions to `A*x+b=0`.

`x0` is a particular solution (if any) and `kerA=` nullspace of `A`. Any `x=x0+kerA*w` with arbitrary `w` satisfies `A*x+b=0`.

If compatible `x0` is given on entry, `x0` is returned. If not a compatible `x0`, if any, is returned.

Examples

```
A=rand(5,3)*rand(3,8);
b=A*ones(8,1);[x,kerA]=linsolve(A,b);A*x+b   //compatible b
b=ones(5,1);[x,kerA]=linsolve(A,b);A*x+b   //uncompatible b
A=rand(5,5);[x,kerA]=linsolve(A,b), -inv(A)*b  //x is unique
```

See Also

inv , pinv , colcomp , im_inv

Name

lsq — linear least square problems.

```
X=lsq(A,B [,tol])
```

Parameters

A
: Real or complex (m x n) matrix

B
: real or complex (m x p) matrix

tol
: positive scalar, used to determine the effective rank of A (defined as the order of the largest leading triangular submatrix R11 in the QR factorization with pivoting of A, whose estimated condition number <= 1/tol. The tol default value is set to `sqrt(%eps)`.

X
: real or complex (n x p) matrix

Description

`X=lsq(A,B)` computes the minimum norm least square solution of the equation `A*X=B`, while `X=A \ B` compute a least square solution with at at most `rank(A)` nonzero components per column.

References

`lsq` function is based on the LApack functions DGELSY for real matrices and ZGELSY for complex matrices.

Examples

```
//Build the data
x=(1:10)';

y1=3*x+4.5+3*rand(x,'normal');
y2=1.8*x+0.5+2*rand(x,'normal');
plot2d(x,[y1,y2],[-2,-3])
//Find the linear regression
A=[x,ones(x)];B=[y1,y2];
X=lsq(A,B);

y1e=X(1,1)*x+X(2,1);
y2e=X(1,2)*x+X(2,2);
plot2d(x,[y1e,y2e],[2,3])

//Difference between lsq(A,b) and A\b
A=rand(4,2)*rand(2,3);//a rank 2 matrix
b=rand(4,1);
X1=lsq(A,b)
X2=A\b
[A*X1-b, A*X2-b] //the residuals are the same
```

See Also

backslash , inv , pinv , rank

Name

lu — LU factors of Gaussian elimination

```
[L,U]= lu(A)
[L,U,E]= lu(A)
```

Parameters

A
: real or complex matrix (m x n).

L
: real or complex matrices (m x min(m,n)).

U
: real or complex matrices (min(m,n) x n).

E
: a (n x n) permutation matrix.

Description

`[L,U]= lu(A)` produces two matrices `L` and `U` such that `A = L*U` with `U` upper triangular and `E*L` lower triangular for a permutation matrix `E`.

If `A` has rank `k`, rows `k+1` to `n` of `U` are zero.

`[L,U,E]= lu(A)` produces three matrices `L`, `U` and `E` such that `E*A = L*U` with `U` upper triangular and `E*L` lower triangular for a permutation matrix `E`.

If `A` is a real matrix, using the function `lufact` and `luget` it is possible to obtain the permutation matrices and also when `A` is not full rank the column compression of the matrix `L`.

Examples

```
a=rand(4,4);
[l,u]=lu(a)
norm(l*u-a)

[h,rk]=lufact(sparse(a))  // lufact fonctionne avec des matrices creuses
[P,L,U,Q]=luget(h);
ludel(h)
P=full(P);L=full(L);U=full(U);Q=full(Q);
norm(P*L*U*Q-a) // P,Q sont des matrices de permutation
```

See Also

lufact , luget , lusolve , qr , svd

Used Functions

lu decompositions are based on the Lapack routines DGETRF for real matrices and ZGETRF for the complex case.

Name

lyap — Lyapunov equation

```
[X]=lyap(A,C,'c')
[X]=lyap(A,C,'d')
```

Parameters

A, C
: real square matrices, `C` must be symmetric

Description

`X = lyap(A,C,flag)` solves the continuous time or discrete time matrix Lyapunov matrix equation:

```
 A'*X + X*A = C       ( flag='c' )
A'*X*A - X = C       ( flag='d' )
```

Note that a unique solution exist if and only if an eigenvalue of `A` is not an eigenvalue of `-A` (`flag='c'`) or 1 over an eigenvalue of `A` (`flag='d'`).

Examples

```
A=rand(4,4);C=rand(A);C=C+C';
X=lyap(A,C,'c');
A'*X + X*A -C
X=lyap(A,C,'d');
A'*X*A - X -C
```

See Also

sylv , ctr_gram , obs_gram

Name

nlev — Leverrier's algorithm

```
[num,den]=nlev(A,z [,rmax])
```

Parameters

A
: real square matrix

z
: character string

rmax
: optional parameter (see `bdiag`)

Description

`[num,den]=nlev(A,z [,rmax])` computes `(z*eye()-A)^(-1)`

by block diagonalization of A followed by Leverrier's algorithm on each block.

This algorithm is better than the usual leverrier algorithm but still not perfect!

Examples

```
A=rand(3,3);x=poly(0,'x');
[NUM,den]=nlev(A,'x')
clean(den-poly(A,'x'))
clean(NUM/den-inv(x*eye()-A))
```

See Also

coff , coffg , glever , ss2tf

Authors

F. Delebecque., S. Steer INRIA;

Name

orth — orthogonal basis

```
Q=orth(A)
```

Parameters

A
: real or complex matrix

Q
: real or complex matrix

Description

`Q=orth(A)` returns `Q`, an orthogonal basis for the span of `A`. Range(`Q`) = Range(`A`) and `Q'*Q=eye`.

The number of columns of `Q` is the rank of `A` as determined by the QR algorithm.

Examples

```
A=rand(5,3)*rand(3,4);
[X,dim]=rowcomp(A);X=X';
svd([orth(A),X(:,1:dim)])
```

See Also

qr , rowcomp , colcomp , range

Name

pbig — eigen-projection

```
[Q,M]=pbig(A,thres,flag)
```

Parameters

A
: real square matrix

thres
: real number

flag
: character string (`'c'` or `'d'`)

Q,M
: real matrices

Description

Projection on eigen-subspace associated with eigenvalues with real part >= `thres` (`flag='c'`) or with magnitude >= `thres` (`flag='d'`).

The projection is defined by `Q*M`, `Q` is full column rank, `M` is full row rank and `M*Q=eye`.

If `flag='c'`, the eigenvalues of `M*A*Q` = eigenvalues of `A` with real part >= `thres`.

If `flag='d'`, the eigenvalues of `M*A*Q` = eigenvalues of `A` with magnitude >= `thres`.

If `flag='c'` and if `[Q1,M1]` = full rank factorization (`fullrf`) of `eye()-Q*M` then eigenvalues of `M1*A*Q1` = eigenvalues of `A` with real part < `thres`.

If `flag='d'` and if `[Q1,M1]` = full rank factorization (`fullrf`) of `eye()-Q*M` then eigenvalues of `M1*A*Q1` = eigenvalues of `A` with magnitude < `thres`.

Examples

```
A=diag([1,2,3]);X=rand(A);A=inv(X)*A*X;
[Q,M]=pbig(A,1.5,'d');
spec(M*A*Q)
[Q1,M1]=fullrf(eye()-Q*M);
spec(M1*A*Q1)
```

See Also

psmall , projspec , fullrf , schur

Authors

F. D. (1988); ;

Used Functions

`pbig` is based on the ordered schur form (scilab function `schur`).

Name

pencan — canonical form of matrix pencil

```
[Q,M,i1]=pencan(Fs)
[Q,M,i1]=pencan(E,A)
```

Parameters

Fs
: a regular pencil `s*E-A`

E,A
: two real square matrices

Q,M
: two non-singular real matrices

i1
: integer

Description

Given the regular pencil `Fs=s*E-A`, `pencan` returns matrices `Q` and `M` such than `M*(s*E-A)*Q` is in "canonical" form.

`M*E*Q` is a block matrix

```
[I,0;
 0,N]
```

with `N` nilpotent and `i1` = size of the `I` matrix above.

`M*A*Q` is a block matrix:

```
[Ar,0;
 0,I]
```

Examples

```
F=randpencil([],[1,2],[1,2,3],[]);
F=rand(6,6)*F*rand(6,6);
[Q,M,i1]=pencan(F);
W=clean(M*F*Q)
roots(det(W(1:i1,1:i1)))
det(W($-2:$,$-2:$))
```

See Also

glever , penlaur , rowshuff

Authors

F. D.; ;

Name

penlaur — Laurent coefficients of matrix pencil

```
[Si,Pi,Di,order]=penlaur(Fs)
[Si,Pi,Di,order]=penlaur(E,A)
```

Parameters

Fs
: a regular pencil `s*E-A`

E, A
: two real square matrices

Si,Pi,Di
: three real square matrices

order
: integer

Description

`penlaur` computes the first Laurent coefficients of `(s*E-A)^-1` at infinity.

`(s*E-A)^-1 = ... + Si/s - Pi - s*Di + ...` at `s` = infinity.

`order` = order of the singularity (order=index-1).

The matrix pencil `Fs=s*E-A` should be invertible.

For a index-zero pencil, `Pi, Di,...` are zero and `Si=inv(E)`.

For a index-one pencil (order=0),`Di` =0.

For higher-index pencils, the terms `-s^2 Di(2), -s^3 Di(3),...` are given by:

`Di(2)=Di*A*Di`, `Di(3)=Di*A*Di*A*Di` (up to `Di(order)`).

Remark

Experimental version: troubles when bad conditioning of `so*E-A`

Examples

```
F=randpencil([],[1,2],[1,2,3],[]);
F=rand(6,6)*F*rand(6,6);[E,A]=pen2ea(F);
[Si,Pi,Di]=penlaur(F);
[Bfs,Bis,chis]=glever(F);
norm(coeff(Bis,1)-Di,1)
```

See Also

glever , pencan , rowshuff

Authors

F. Delebecque INRIA(1988,1990) ;

Name

pinv — pseudoinverse

```
pinv(A,[tol])
```

Parameters

A
: real or complex matrix

tol
: real number

Description

`X = pinv(A)` produces a matrix `X` of the same dimensions as `A'` such that:

`A*X*A = A, X*A*X = X` and both `A*X` and `X*A` are Hermitian .

The computation is based on SVD and any singular values lower than a tolerance are treated as zero: this tolerance is accessed by `X=pinv(A,tol)`.

Examples

```
A=rand(5,2)*rand(2,4);
norm(A*pinv(A)*A-A,1)
```

See Also

rank , svd , qr

Used Functions

`pinv` function is based on the singular value decomposition (Scilab function `svd`).

Name

polar — polar form

```
[Ro,Theta]=polar(A)
```

Parameters

A
: real or complex square matrix

Ro,
: real matrix

Theta,
: real or complex matrix

Description

`[Ro,Theta]=polar(A)` returns the polar form of `A` i.e. `A=Ro*expm(%i*Theta)Ro` symmetric >=0 and `Theta` hermitian >=0.

Examples

```
A=rand(5,5);
[Ro,Theta]=polar(A);
norm(A-Ro*expm(%i*Theta),1)
```

See Also

expm , svd

Authors

F. Delebecque INRIA; ;

Name

proj — projection

```
P = proj(X1,X2)
```

Parameters

X1,X2
: two real matrices with equal number of columns

P
: real projection matrix (`P^2=P`)

Description

`P` is the projection on `X2` parallel to `X1`.

Examples

```
X1=rand(5,2);X2=rand(5,3);
P=proj(X1,X2);
norm(P^2-P,1)
trace(P)    // This is dim(X2)
[Q,M]=fullrf(P);
svd([Q,X2])   // span(Q) = span(X2)
```

See Also

projspec , orth , fullrf

Authors

F. D.; ;

Name

projspec — spectral operators

```
[S,P,D,i]=projspec(A)
```

Parameters

A
: square matrix

S, P, D
: square matrices

i
: integer (index of the zero eigenvalue of A).

Description

Spectral characteristics of A at 0.

S = reduced resolvent at 0 (S = -Drazin_inverse(A)).

P = spectral projection at 0.

D = nilpotent operator at 0.

index = index of the 0 eigenvalue.

One has `(s*eye()-A)^(-1) = D^(i-1)/s^i +... + D/s^2 + P/s - S - s*S^2 -...` around the singularity s=0.

Examples

```
deff('j=jdrn(n)','j=zeros(n,n);for k=1:n-1;j(k,k+1)=1;end')
A=sysdiag(jdrn(3),jdrn(2),rand(2,2));X=rand(7,7);
A=X*A*inv(X);
[S,P,D,index]=projspec(A);
index   //size of J-block
trace(P)  //sum of dimensions of J-blocks
A*S-(eye()-P)
norm(D^index,1)
```

See Also

coff

Authors

F. D.; ;

Name

psmall — spectral projection

```
[Q,M]=psmall(A,thres,flag)
```

Parameters

A
: real square matrix

thres
: real number

flag
: character string (`'c'` or `'d'`)

Q,M
: real matrices

Description

Projection on eigen-subspace associated with eigenvalues with real part < `thres` (`flag='c'`) or with modulus < `thres` (`flag='d'`).

The projection is defined by `Q*M`, `Q` is full column rank, `M` is full row rank and `M*Q=eye`.

If `flag='c'`, the eigenvalues of `M*A*Q` = eigenvalues of `A` with real part < `thres`.

If `flag='d'`, the eigenvalues of `M*A*Q` = eigenvalues of `A` with magnitude < `thres`.

If `flag='c'` and if `[Q1,M1]` = full rank factorization (`fullrf`) of `eye()-Q*M` then eigenvalues of `M1*A*Q1` = eigenvalues of `A` with real part >= `thres`.

If `flag='d'` and if `[Q1,M1]` = full rank factorization (`fullrf`) of `eye()-Q*M` then eigenvalues of `M1*A*Q1` = eigenvalues of `A` with magnitude >= `thres`.

Examples

```
A=diag([1,2,3]);X=rand(A);A=inv(X)*A*X;
[Q,M]=psmall(A,2.5,'d');
spec(M*A*Q)
[Q1,M1]=fullrf(eye()-Q*M);
spec(M1*A*Q1)
```

See Also

pbig , proj , projspec

Authors

F. Delebecque INRIA. (1988);

Used Functions

This function is based on the ordered schur form (scilab function `schur`).

Name

qr — QR decomposition

```
[Q,R]=qr(X [,"e"])
[Q,R,E]=qr(X [,"e"])
[Q,R,rk,E]=qr(X [,tol])
```

Parameters

X
: real or complex matrix

tol
: nonnegative real number

Q
: square orthogonal or unitary matrix

R
: matrix with same dimensions as `X`

E
: permutation matrix

rk
: integer (QR-rank of `X`)

Description

[Q,R] = qr(X)
: produces an upper triangular matrix `R` of the same dimension as `X` and an orthogonal (unitary in the complex case) matrix `Q` so that `X = Q*R`. `[Q,R] = qr(X,"e")` produces an "economy size": If `X` is m-by-n with m > n, then only the first n columns of `Q` are computed as well as the first n rows of `R`.

  From `Q*R = X`, it follows that the kth column of the matrix `X`, is expressed as a linear combination of the k first columns of `Q` (with coefficients `R(1,k), ..., R(k,k)`). The k first columns of `Q` make an orthogonal basis of the subspace spanned by the k first comumns of `X`. If column `k` of `X` (i.e. `X(:,k)`) is a linear combination of the first `p` columns of `X`, then the entries `R(p+1,k), ..., R(k,k)` are zero. It this situation, `R` is upper trapezoidal. If `X` has rank `rk`, rows `R(rk+1,:), R(rk+2,:), ...` are zeros.

[Q,R,E] = qr(X)
: produces a (column) permutation matrix `E`, an upper triangular `R` with decreasing diagonal elements and an orthogonal (or unitary) `Q` so that `X*E = Q*R`. If `rk` is the rank of `X`, the `rk` first entries along the main diagonal of `R`, i.e. `R(1,1), R(2,2), ..., R(rk,rk)` are all different from zero. `[Q,R,E] = qr(X,"e")` produces an "economy size": If `X` is m-by-n with m > n, then only the first n columns of `Q` are computed as well as the first n rows of `R`.

[Q,R,rk,E] = qr(X ,tol)
: returns `rk` = rank estimate of `X` i.e. `rk` is the number of diagonal elements in `R` which are larger than a given threshold `tol`.

[Q,R,rk,E] = qr(X)
: returns `rk` = rank estimate of `X` i.e. `rk` is the number of diagonal elements in `R` which are larger than `tol=R(1,1)*%eps*max(size(R))`. See `rankqr` for a rank revealing QR factorization, using the condition number of `R`.

Examples

```
// QR factorization, generic case
// X is tall (full rank)
X=rand(5,2);[Q,R]=qr(X); [Q'*X R]
//X is fat (full rank)
X=rand(2,3);[Q,R]=qr(X); [Q'*X R]
//Column 4 of X is a linear combination of columns 1 and 2:
X=rand(8,5);X(:,4)=X(:,1)+X(:,2); [Q,R]=qr(X); R, R(:,4)
//X has rank 2, rows 3 to $ of R are zero:
X=rand(8,2)*rand(2,5);[Q,R]=qr(X); R
//Evaluating the rank rk: column pivoting ==> rk first
//diagonal entries of R are non zero :
A=rand(5,2)*rand(2,5);
[Q,R,rk,E] = qr(A,1.d-10);
norm(Q'*A-R)
svd([A,Q(:,1:rk)])    //span(A) =span(Q(:,1:rk))
```

See Also

rankqr , rank , svd , rowcomp , colcomp

Used Functions

qr decomposition is based the Lapack routines DGEQRF, DGEQPF, DORGQR for the real matrices and ZGEQRF, ZGEQPF, ZORGQR for the complex case.

Name

quaskro — quasi-Kronecker form

```
[Q,Z,Qd,Zd,numbeps,numbeta]=quaskro(F)
[Q,Z,Qd,Zd,numbeps,numbeta]=quaskro(E,A)
[Q,Z,Qd,Zd,numbeps,numbeta]=quaskro(F,tol)
[Q,Z,Qd,Zd,numbeps,numbeta]=quaskro(E,A,tol)
```

Parameters

F
: real matrix pencil `F=s*E-A` (`s=poly(0,'s')`)

E,A
: two real matrices of same dimensions

tol
: a real number (tolerance, default value=1.d-10)

Q,Z
: two square orthogonal matrices

Qd,Zd
: two vectors of integers

numbeps
: vector of integers

Description

Quasi-Kronecker form of matrix pencil: `quaskro` computes two orthogonal matrices `Q, Z` which put the pencil `F=s*E -A` into upper-triangular form:

```
           | sE(eps)-A(eps) |        X       |      X     |
           |----------------|----------------|------------|
           |        0       | sE(inf)-A(inf) |      X     |
Q(sE-A)Z = |================================|============|
           |                                |            |
           |                0               | sE(r)-A(r) |
```

The dimensions of the blocks are given by:

`eps=Qd(1) x Zd(1)`, `inf=Qd(2) x Zd(2)`, `r = Qd(3) x Zd(3)`

The `inf` block contains the infinite modes of the pencil.

The `f` block contains the finite modes of the pencil

The structure of epsilon blocks are given by:

`numbeps(1)` = # of eps blocks of size 0 x 1

`numbeps(2)` = # of eps blocks of size 1 x 2

`numbeps(3)` = # of eps blocks of size 2 x 3 etc...

The complete (four blocks) Kronecker form is given by the function `kroneck` which calls `quaskro` on the (pertransposed) pencil `sE(r)-A(r)`.

The code is taken from T. Beelen

See Also

kroneck , gschur , gspec

Name

randpencil — random pencil

```
F=randpencil(eps,infi,fin,eta)
```

Parameters

eps
: vector of integers

infi
: vector of integers

fin
: real vector, or monic polynomial, or vector of monic polynomial

eta
: vector of integers

F
: real matrix pencil `F=s*E-A(s=poly(0,'s'))`

Description

Utility function. `F=randpencil(eps,infi,fin,eta)` returns a random pencil `F` with given Kronecker structure. The structure is given by: `eps=[eps1,...,epsk]`: structure of epsilon blocks (size eps1x(eps1+1),....) `fin=[l1,...,ln]` set of finite eigenvalues (assumed real) (possibly []) `infi=[k1,...,kp]` size of J-blocks at infinity `ki>=1` (infi=[] if no J blocks). `eta=[eta1,...,etap]`: structure ofeta blocks (size eta1+1)xeta1,...)

`epsi`'s should be >=0, `etai`'s should be >=0, `infi`'s should be >=1.

If `fin` is a (monic) polynomial, the finite block admits the roots of `fin` as eigenvalues.

If `fin` is a vector of polynomial, they are the finite elementary divisors of `F` i.e. the roots of `p(i)` are finite eigenvalues of `F`.

Examples

```
F=randpencil([0,1],[2],[-1,0,1],[3]);
[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck(F);
Qd, Zd
s=poly(0,'s');
F=randpencil([],[1,2],s^3-2,[]); //regular pencil
det(F)
```

See Also

kroneck , pencan , penlaur

Name

range — range (span) of A^k

```
[X,dim]=range(A,k)
```

Parameters

A
: real square matrix

k
: integer

X
: orthonormal real matrix

dim
: integer (dimension of subspace)

Description

Computation of Range `A^k` ; the first dim rows of `X` span the range of `A^k`. The last rows of `X` span the orthogonal complement of the range. `X*X'` is the Identity matrix

Examples

```
A=rand(4,2)*rand(2,4);   // 4 column vectors, 2 independent.
[X,dim]=range(A,1);dim   // compute the range

y1=A*rand(4,1);          //a vector which is in the range of A
y2=rand(4,1);            //a vector which is not  in the range of A
norm(X(dim+1:$,:)*y1)    //the last entries are zeros, y1 is in the range o
norm(X(dim+1:$,:)*y2)    //the last entries are not zeros

I=X(1:dim,:)'            //I is a basis of the range
coeffs=X(1:dim,:)*y1     // components of y1 relative to the I basis

norm(I*coeffs-y1)        //check
```

See Also

fullrfk , rowcomp

Authors

F. D. INRIA ;

Used Functions

The `range` function is based on the rowcomp function which uses the svd decomposition.

Name

rank — rank

```
[i]=rank(X)
[i]=rank(X,tol)
```

Parameters

X
: real or complex matrix

tol
: nonnegative real number

Description

`rank(X)` is the numerical rank of `X` i.e. the number of singular values of X that are larger than `norm(size(X),'inf') * norm(X) * %eps`.

`rank(X,tol)` is the number of singular values of `X` that are larger than `tol`.

Note that the default value of `tol` is proportional to `norm(X)`. As a consequence `rank([1.d-80,0;0,1.d-80])` is 2 !.

Examples

```
rank([1.d-80,0;0,1.d-80])
rank([1,0;0,1.d-80])
```

See Also

svd , qr , rowcomp , colcomp , lu

Name

rankqr — rank revealing QR factorization

```
[Q,R,JPVT,RANK,SVAL]=rankqr(A, [RCOND,JPVT])
```

Parameters

A
: real or complex matrix

RCOND
: real number used to determine the effective rank of A, which is defined as the order of the largest leading triangular submatrix R11 in the QR factorization with pivoting ofA, whose estimated condition number < 1/RCOND.

JPVT
: integer vector on entry, if JPVT(i) is not 0, the i-th column of A is permuted to the front of AP, otherwise column i is a free column. On exit, if JPVT(i) = k, then the i-th column of A*P was the k-th column of A.

RANK
: the effective rank of A, i.e., the order of the submatrix R11. This is the same as the order of the submatrix T1 in the complete orthogonal factorization of A.

SVAL
: real vector with 3 components; The estimates of some of the singular values of the triangular factor R.

  SVAL(1) is the largest singular value of R(1:RANK,1:RANK);

  SVAL(2) is the smallest singular value of R(1:RANK,1:RANK);

  SVAL(3) is the smallest singular value of R(1:RANK+1,1:RANK+1), if RANK < MIN(M,N), or of R(1:RANK,1:RANK), otherwise.

Description

To compute (optionally) a rank-revealing QR factorization of a real general M-by-N real or complex matrix A, which may be rank-deficient, and estimate its effective rank using incremental condition estimation.

The routine uses a QR factorization with column pivoting:

```
      A * P = Q * R,  where  R = [ R11 R12 ],
                                 [  0  R22 ]
```

with R11 defined as the largest leading submatrix whose estimated condition number is less than 1/RCOND. The order of R11, RANK, is the effective rank of A.

If the triangular factorization is a rank-revealing one (which will be the case if the leading columns were well- conditioned), then SVAL(1) will also be an estimate for the largest singular value of A, and SVAL(2) and SVAL(3) will be estimates for the RANK-th and (RANK+1)-st singular values of A, respectively.

By examining these values, one can confirm that the rank is well defined with respect to the chosen value of RCOND. The ratio SVAL(1)/SVAL(2) is an estimate of the condition number of R(1:RANK,1:RANK).

Examples

```
A=rand(5,3)*rand(3,7);
[Q,R,JPVT,RANK,SVAL]=rankqr(A,%eps)
```

See Also

qr , rank

Used Functions

Slicot library routines MB03OD, ZB03OD.

Name

rcond — inverse condition number

```
rcond(X)
```

Parameters

X
: real or complex square matrix

Description

`rcond(X)` is an estimate for the reciprocal of the condition of `X` in the 1-norm.

If `X` is well conditioned, `rcond(X)` is close to 1. If not, `rcond(X)` is close to 0.

`[r,z]=rcond(X)` sets `r` to `rcond(X)` and returns `z` such that `norm(X*z,1) = r*norm(X,1)*norm(z,1)`

Thus, if `rcond` is small `z` is a vector in the kernel.

Examples

```
A=diag([1:10]);
rcond(A)
A(1,1)=0.000001;
rcond(A)
```

See Also

svd , cond , inv

Name

rowcomp — row compression, range

```
[W,rk]=rowcomp(A [,flag [,tol]])
```

Parameters

A
: real or complex matrix

flag
: optionnal character string, with possible values `'svd'` or `'qr'`. The default value is `'svd'`.

tol
: optionnal real non negative number. The default value is `sqrt(%eps)*norm(A,1)`.

W
: square non-singular matrix (change of basis)

rk
: integer (rank of A)

Description

Row compression of A. `Ac = W*A` is a row compressed matrix: i.e. `Ac=[Af;0]` with `Af` full row rank.

`flag` and `tol` are optional parameters: `flag='qr'` or `'svd'` (default `'svd'`).

`tol` is a tolerance parameter.

The `rk` first columns of `W'` span the range of A.

The `rk` first (top) rows of `W` span the row range of A.

A non zero vector `x` belongs to range(A) iff `W*x` is row compressed in accordance with `Ac` i.e the norm of its last components is small w.r.t its first components.

Examples

```
A=rand(5,2)*rand(2,4);              // 4 col. vectors, 2 independent.
[X,dim]=rowcomp(A);Xp=X';
svd([Xp(:,1:dim),A])                //span(A) = span(Xp(:,1:dim)
x=A*rand(4,1);                      //x belongs to span(A)
y=X*x
norm(y(dim+1:$))/norm(y(1:dim))     // small
```

See Also

colcomp , fullrf , fullrfk

Authors

F. D.; INRIA

Used Functions

The `rowcomp` function is based on the svd or qr decompositions.

Name

rowshuff — shuffle algorithm

```
[Ws,Fs1]=rowshuff(Fs, [alfa])
```

Parameters

Fs
: square real pencil `Fs = s*E-A`

Ws
: polynomial matrix

Fs1
: square real pencil `F1s = s*E1 -A1` with `E1` non-singular

alfa
: real number (`alfa = 0` is the default value)

Description

Shuffle algorithm: Given the pencil `Fs=s*E-A`, returns Ws=W(s) (square polynomial matrix) such that:

`Fs1 = s*E1-A1 = W(s)*(s*E-A)` is a pencil with non singular `E1` matrix.

This is possible iff the pencil `Fs = s*E-A` is regular (i.e. invertible). The degree of `Ws` is equal to the index of the pencil.

The poles at infinity of `Fs` are put to `alfa` and the zeros of `Ws` are at `alfa`.

Note that `(s*E-A)^-1 = (s*E1-A1)^-1 * W(s) = (W(s)*(s*E-A))^-1 *W(s)`

Examples

```
F=randpencil([],[2],[1,2,3],[]);
F=rand(5,5)*F*rand(5,5);   // 5 x 5 regular pencil with 3 evals at 1,2,3
[Ws,F1]=rowshuff(F,-1);
[E1,A1]=pen2ea(F1);
svd(E1)           //E1 non singular
roots(det(Ws))
clean(inv(F)-inv(F1)*Ws,1.d-7)
```

See Also

pencan , glever , penlaur

Authors

F. D.; ; ; ; ;

Name

rref — computes matrix row echelon form by lu transformations

```
R=rref(A)
```

Parameters

A
: m x n matrix with scalar entries

R
: m x n matrix,row echelon form of a

Description

`rref` computes the row echelon form of the given matrix by left lu decomposition. If ones need the transformation used just call `X=rref([A,eye(m,m)])` the row echelon form `R` is `X(:,1:n)` and the left transformation `L` is given by `X(:,n+1:n+m)` such as `L*A=R`

Examples

```
A=[1 2;3 4;5 6];
X=rref([A,eye(3,3)]);
R=X(:,1:2)
L=X(:,3:5);L*A
```

See Also

lu , qr

Name

schur — [ordered] Schur decomposition of matrix and pencils

```
[U,T] = schur(A)
[U,dim [,T] ]=schur(A,flag)
[U,dim [,T] ]=schur(A,extern1)

[As,Es [,Q,Z]]=schur(A,E)
[As,Es [,Q],Z,dim] = schur(A,E,flag)
[Z,dim] = schur(A,E,flag)
[As,Es [,Q],Z,dim]= schur(A,E,extern2)
[Z,dim]= schur(A,E,extern2)
```

Parameters

A
: real or complex square matrix.

E
: real or complex square matrix with same dimensions as A.

flag
: character string ('c' or 'd')

extern1
: an ``external'', see below

extern2
: an ``external'', see below

U
: orthogonal or unitary square matrix

Q
: orthogonal or unitary square matrix

Z
: orthogonal or unitary square matrix

T
: upper triangular or quasi-triangular square matrix

As
: upper triangular or quasi-triangular square matrix

Es
: upper triangular square matrix

dim
: integer

Description

Schur forms, ordered Schur forms of matrices and pencils

MATRIX SCHUR FORM

Usual schur form:
[U,T] = schur(A) produces a Schur matrix T and a unitary matrix U so that A = U*T*U' and U'*U = eye(U). By itself, schur(A) returns T. If A is complex, the Complex

Schur Form is returned in matrix T. The Complex Schur Form is upper triangular with the eigenvalues of A on the diagonal. If A is real, the Real Schur Form is returned. The Real Schur Form has the real eigenvalues on the diagonal and the complex eigenvalues in 2-by-2 blocks on the diagonal.

Ordered Schur forms

[U,dim]=schur(A,'c') returns an unitary matrix U which transforms A into schur form. In addition, the dim first columns of U make a basis of the eigenspace of A associated with eigenvalues with negative real parts (stable "continuous time" eigenspace).

[U,dim]=schur(A,'d') returns an unitary matrix U which transforms A into schur form. In addition, the dim first columns of U span a basis of the eigenspace of A associated with eigenvalues with magnitude lower than 1 (stable "discrete time" eigenspace).

[U,dim]=schur(A,extern1) returns an unitary matrix U which transforms A into schur form. In addition, the dim first columns of U span a basis of the eigenspace of A associated with the eigenvalues which are selected by the external function extern1 (see external for details). This external can be described by a Scilab function or by C or Fortran procedure:

a Scilab function

If extern1 is described by a Scilab function, it should have the following calling sequence: s=extern1(Ev), where Ev is an eigenvalue and s a boolean.

a C or Fortran procedure

If extern1 is described by a C or Fortran function it should have the following calling sequence: int extern1(double *EvR, double *EvI) where EvR and EvI are eigenvalue real and complex parts. a true or non zero returned value stands for selected eigenvalue.

PENCIL SCHUR FORMS

Usual Pencil Schur form

[As,Es] = schur(A,E) produces a quasi triangular As matrix and a triangular Es matrix which are the generalized Schur form of the pair A, E.

[As,Es,Q,Z] = schur(A,E) returns in addition two unitary matrices Q and Z such that As=Q'*A*Z and Es=Q'*E*Z.

Ordered Schur forms:

[As,Es,Z,dim] = schur(A,E,'c') returns the real generalized Schur form of the pencil s*E-A. In addition, the dim first columns of Z span a basis of the right eigenspace associated with eigenvalues with negative real parts (stable "continuous time" generalized eigenspace).

```
[As,Es,Z,dim] = schur(A,E,'d')
```

returns the real generalized Schur form of the pencil s*E-A. In addition, the dim first columns of Z make a basis of the right eigenspace associated with eigenvalues with magnitude lower than 1 (stable "discrete time" generalized eigenspace).

```
[As,Es,Z,dim] = schur(A,E,extern2)
```

returns the real generalized Schur form of the pencil s*E-A. In addition, the dim first columns of Z make a basis of the right eigenspace associated with eigenvalues of the pencil which are selected according to a rule which is given by the function extern2. (see external for details). This external can be described by a Scilab function or by C or Fortran procedure:

A Scilab function

If extern2 is described by a Scilab function, it should have the following calling sequence: s=extern2(Alpha,Beta), where Alpha and Beta defines a generalized eigenvalue and s a boolean.

C or Fortran procedure

if external `extern2` is described by a C or a Fortran procedure, it should have the following calling sequence:

```
int extern2(double *AlphaR, double *AlphaI, double *Beta)
```

if A and E are real and

```
int extern2(double *AlphaR, double *AlphaI, double *BetaR,
double *BetaI)
```

if A or E are complex. `Alpha`, and `Beta` defines the generalized eigenvalue. a true or non zero returned value stands for selected generalized eigenvalue.

References

Matrix schur form computations are based on the Lapack routines DGEES and ZGEES.

Pencil schur form computations are based on the Lapack routines DGGES and ZGGES.

Examples

```
//SCHUR FORM OF A MATRIX
//----------------------
A=diag([-0.9,-2,2,0.9]);X=rand(A);A=inv(X)*A*X;
[U,T]=schur(A);T

[U,dim,T]=schur(A,'c');
T(1:dim,1:dim)      //stable cont. eigenvalues

function t=mytest(Ev),t=abs(Ev)<0.95,endfunction
[U,dim,T]=schur(A,mytest);
T(1:dim,1:dim)

// The same function in C (a Compiler is required)
C=['int mytest(double *EvR, double *EvI) {' //the C code
   'if (*EvR * *EvR + *EvI * *EvI < 0.9025) return 1;'
   'else return 0; }';]
mputl(C,TMPDIR+'/mytest.c')


//build and link
lp=ilib_for_link('mytest','mytest.o',[],'c',TMPDIR+'/Makefile');
link(lp,'mytest','c');

//run it
[U,dim,T]=schur(A,'mytest');
//SCHUR FORM OF A PENCIL
//----------------------
F=[-1,%s, 0,   1;
    0,-1,5-%s, 0;
    0, 0,2+%s, 0;
    1, 0, 0, -2+%s];
A=coeff(F,0);E=coeff(F,1);
[As,Es,Q,Z]=schur(A,E);
Q'*F*Z //It is As+%s*Es
```

```
[As,Es,Z,dim] = schur(A,E,'c')
function t=mytest(Alpha,Beta),t=real(Alpha)<0,endfunction
[As,Es,Z,dim] = schur(A,E,mytest)

//the same function in Fortran (a Compiler is required)
ftn=['integer function mytestf(ar,ai,b)' //the fortran code
      'double precision ar,ai,b'
      'mytestf=0'
      'if(ar.lt.0.0d0) mytestf=1'
      'end']
mputl('      '+ftn,TMPDIR+'/mytestf.f')

//build and link
lp=ilib_for_link('mytestf','mytestf.o',[],'F',TMPDIR+'/Makefile');
link(lp,'mytestf','f');

//run it

[As,Es,Z,dim] = schur(A,E,'mytestf')
```

See Also

spec , bdiag , ricc , pbig , psmall

Name

spaninter — subspace intersection

```
[X,dim]=spaninter(A,B [,tol])
```

Parameters

A, B
: two real or complex matrices with equal number of rows

X
: orthogonal or unitary square matrix

dim
: integer, dimension of subspace range(A) inter range(B)

Description

computes the intersection of range(A) and range(B).

The first `dim` columns of X span this intersection i.e. `X(:,1:dim)` is an orthogonal basis for

range(A) inter range(B)

In the X basis A and B are respectively represented by:

`X'*A` and `X'*B`.

`tol` is a threshold (`sqrt(%eps)` is the default value).

Examples

```
A=rand(5,3)*rand(3,4);     // A is 5 x 4, rank=3
B=[A(:,2),rand(5,1)]*rand(2,2);
[X,dim]=spaninter(A,B);
X1=X(:,1:dim);    //The intersection
svd(A),svd([X1,A])   // X1 in span(A)
svd(B),svd([B,X1])   // X1 in span(B)
```

See Also

spanplus , spantwo

Authors

F. D.; ;

Name

spanplus — sum of subspaces

```
[X,dim,dima]=spanplus(A,B[,tol])
```

Parameters

A, B
: two real or complex matrices with equal number of rows

X
: orthogonal or unitary square matrix

dim, dima
: integers, dimension of subspaces

tol
: nonnegative real number

Description

computes a basis X such that:

the first `dima` columns of `X` span Range(`A`) and the following (`dim-dima`) columns make a basis of `A+B` relative to `A`.

The `dim` first columns of `X` make a basis for `A+B`.

One has the following canonical form for `[A,B]`:

```
         [*,*]    (dima rows)
X'*[A,B]=[0,*]    (dim-dima rows)
         [0,0]
```

`tol` is an optional argument (see function code).

Examples

```
A=rand(6,2)*rand(2,5);      // rank(A)=2
B=[A(:,1),rand(6,2)]*rand(3,3);   //two additional independent vectors
[X,dim,dimA]=spanplus(A,B);
dimA
dim
```

See Also

spaninter , im_inv , spantwo

Authors

F. D.; ;

Name

spantwo — sum and intersection of subspaces

```
[Xp,dima,dimb,dim]=spantwo(A,B, [tol])
```

Parameters

A, B
: two real or complex matrices with equal number of rows

Xp
: square non-singular matrix

dima, dimb, dim
: integers, dimension of subspaces

tol
: nonnegative real number

Description

Given two matrices `A` and `B` with same number of rows, returns a square matrix `Xp` (non singular but not necessarily orthogonal) such that :

```
         [A1, 0]    (dim-dimb rows)
Xp*[A,B]=[A2,B2]    (dima+dimb-dim rows)
         [0, B3]    (dim-dima rows)
         [0 , 0]
```

The first `dima` columns of `inv(Xp)` span range(`A`).

Columns `dim-dimb+1` to `dima` of `inv(Xp)` span the intersection of range(A) and range(B).

The `dim` first columns of `inv(Xp)` span range(`A`)+range(`B`).

Columns `dim-dimb+1` to `dim` of `inv(Xp)` span range(`B`).

Matrix `[A1;A2]` has full row rank (=rank(A)). Matrix `[B2;B3]` has full row rank (=rank(B)). Matrix `[A2,B2]` has full row rank (=rank(A inter B)). Matrix `[A1,0;A2,B2;0,B3]` has full row rank (=rank(A+B)).

Examples

```
A=[1,0,0,4;
   5,6,7,8;
   0,0,11,12;
   0,0,0,16];
B=[1,2,0,0]';C=[4,0,0,1];
Sl=ss2ss(syslin('c',A,B,C),rand(A));
[no,X]=contr(Sl('A'),Sl('B'));CO=X(:,1:no);  //Controllable part
[uo,Y]=unobs(Sl('A'),Sl('C'));UO=Y(:,1:uo);  //Unobservable part
```

```
[Xp,dimc,dimu,dim]=spantwo(CO,UO);    //Kalman decomposition
Slcan=ss2ss(Sl,inv(Xp));
```

See Also

spanplus , spaninter

Authors

F. D.

Name

spec — eigenvalues of matrices and pencils

```
evals=spec(A)
[R,diagevals]=spec(A)

evals=spec(A,B)
[alpha,beta]=spec(A,B)
[alpha,beta,Z]=spec(A,B)
[alpha,beta,Q,Z]=spec(A,B)
```

Parameters

A
: real or complex square matrix

B
: real or complex square matrix with same dimensions as `A`

evals
: real or complex vector, the eigenvalues

diagevals
: real or complex diagonal matrix (eigenvalues along the diagonal)

alpha
: real or complex vector, al./be gives the eigenvalues

beta
: real vector, al./be gives the eigenvalues

R
: real or complex invertible square matrix, matrix right eigenvectors.

L
: real or complex invertible square matrix, pencil left eigenvectors.

R
: real or complex invertible square matrix, pencil right eigenvectors.

Description

evals=spec(A)
: returns in vector `evals` the eigenvalues.

[R,diagevals] =spec(A)
: returns in the diagonal matrix `evals` the eigenvalues and in `R` the right eigenvectors.

evals=spec(A,B)
: returns the spectrum of the matrix pencil A - s B, i.e. the roots of the polynomial matrix s B - A.

[alpha,beta] = spec(A,B)
: returns the spectrum of the matrix pencil `A - s B`, i.e. the roots of the polynomial matrix `A - s B`. Generalized eigenvalues alpha and beta are so that the matrix `A - alpha./beta B` is a singular matrix. The eigenvalues are given by `al./be` and if `beta(i) = 0` the ith eigenvalue is at infinity. (For `B = eye(A), alpha./beta` is `spec(A)`). It is usually represented as the pair (alpha,beta), as there is a reasonable interpretation for beta=0, and even for both being zero.

[alpha,beta,R] = spec(A,B)
returns in addition the matrix R of generalized right eigenvectors of the pencil.

[al,be,L,R] = spec(A,B)
returns in addition the matrix L and R of generalized left and right eigenvectors of the pencil.

References

Matrix eigenvalues computations are based on the Lapack routines

- DGEEV and ZGEEV when the matrix are not symmetric,
- DSYEV and ZHEEV when the matrix are symmetric.

A complex symetric matrix has conjugate offdiagonal terms and real diagonal terms.

Pencil eigenvalues computations are based on the Lapack routines DGGEV and ZGGEV.

Real and complex matrices

It must be noticed that the type of the output variables, such as evals or R for example, is not necessarily the same as the type of the input matrices A and B. In the following paragraph, we analyse the type of the output variables in the case where one computes the eigenvalues and eigenvectors of one single matrix A.

- Real A matrix
  - Symetric

    The eigenvalues and the eigenvectors are real.
  - Not symetric

    The eigenvalues and eigenvectors are complex.
- Complex A matrix
  - Symetric

    The eigenvalues are real but the eigenvectors are complex.
  - Not symetric

    The eigenvalues and the eigenvectors are complex.

Examples

```
// MATRIX EIGENVALUES
A=diag([1,2,3]);
X=rand(3,3);
A=inv(X)*A*X;
spec(A)
//
x=poly(0,'x');
pol=det(x*eye()-A)
roots(pol)
//
[S,X]=bdiag(A);
```

```
clean(inv(X)*A*X)

// PENCIL EIGENVALUES
A=rand(3,3);
[al,be,R] = spec(A,eye(A));
al./be
clean(inv(R)*A*R)  //displaying the eigenvalues (generic matrix)
A=A+%i*rand(A);
E=rand(A);
roots(det(A-%s*E))   //complex case
```

See Also

poly, det, schur, bdiag, colcomp

Name

sqroot — W*W' hermitian factorization

```
sqroot(X)
```

Parameters

X
: symmetric non negative definite real or complex matrix

Description

returns W such that `X=W*W'` (uses SVD).

Examples

```
X=rand(5,2)*rand(2,5);X=X*X';
W=sqroot(X)
norm(W*W'-X,1)
//
X=rand(5,2)+%i*rand(5,2);X=X*X';
W=sqroot(X)
norm(W*W'-X,1)
```

See Also

chol , svd

Name

squeeze — squeeze

```
hypOut = squeeze(hypIn)
```

Parameters

hypIn
: hypermatrix or matrix of constant type.

hypOut
: hypermatrix or matrix of constant type.

Description

Remove sigleton dimensions of a hypermatrix, that is any dimension for which the size is 1. If the input is a matrix, it is unaffected.

See Also

hypermat , hypermatrices

Authors

Eric Dubois, Jean-Baptiste Silvy

Name

sva — singular value approximation

```
[U,s,V]=sva(A,k)
[U,s,V]=sva(A,tol)
```

Parameters

A
: real or complex matrix

k
: integer

tol
: nonnegative real number

Description

Singular value approximation.

`[U,S,V]=sva(A,k)` with `k` an integer >=1, returns `U,S` and `V` such that `B=U*S*V'` is the best L2 approximation of `A` with rank(`B`)=`k`.

`[U,S,V]=sva(A,tol)` with `tol` a real number, returns `U,S` and `V` such that `B=U*S*V'` such that L2-norm of `A-B` is at most `tol`.

Examples

```
A=rand(5,4)*rand(4,5);
[U,s,V]=sva(A,2);
B=U*s*V';
svd(A)
svd(B)
clean(svd(A-B))
```

See Also

svd

Name

svd — singular value decomposition

```
s=svd(X)
[U,S,V]=svd(X)
[U,S,V]=svd(X,0) (obsolete)
[U,S,V]=svd(X,"e")
[U,S,V,rk]=svd(X [,tol])
```

Parameters

X
: a real or complex matrix

s
: real vector (singular values)

S
: real diagonal matrix (singular values)

U,V
: orthogonal or unitary square matrices (singular vectors).

tol
: real number

Description

`[U,S,V] = svd(X)` produces a diagonal matrix `S` , of the same dimension as `X` and with nonnegative diagonal elements in decreasing order, and unitary matrices `U` and `V` so that `X = U*S*V'`.

`[U,S,V] = svd(X,0)` produces the "economy size" decomposition. If `X` is m-by-n with m > n, then only the first n columns of `U` are computed and `S` is n-by-n.

`s = svd(X)` by itself, returns a vector `s` containing the singular values.

`[U,S,V,rk]=svd(X,tol)` gives in addition `rk`, the numerical rank of `X` i.e. the number of singular values larger than `tol`.

The default value of `tol` is the same as in `rank`.

Examples

```
X=rand(4,2)*rand(2,4)
svd(X)
sqrt(spec(X*X'))
```

See Also

rank , qr , colcomp , rowcomp , sva , spec

Used Functions

svd decompositions are based on the Lapack routines DGESVD for real matrices and ZGESVD for the complex case.

Name

sylv — Sylvester equation.

```
sylv(A,B,C,flag)
```

Parameters

A,B,C
: three real matrices of appropriate dimensions.

flag
: character string (`'c'` or `'d'`)

Description

`X = sylv(A,B,C,'c')` computes `X`, solution of the "continuous time" Sylvester equation

```
A*X+X*B=C
```

`X=sylv(A,B,C,'d')` computes `X`, solution of the "discrete time" Sylvester equation

```
A*X*B-X=C
```

Examples

```
A=rand(4,4);C=rand(4,3);B=rand(3,3);
X = sylv(A,B,C,'c');
norm(A*X+X*B-C)
X=sylv(A,B,C,'d')
norm(A*X*B-X-C)
```

See Also

lyap

Name

trace — trace

```
trace(X)
```

Parameters

X
: real or complex square matrix, polynomial or rational matrix.

Description

`trace(X)` is the trace of the matrix `X`.

Same as `sum(diag(X))`.

Examples

```
A=rand(3,3);
trace(A)-sum(spec(A))
```

See Also

det

Localization

Name

dgettext — get text translated into the current locale and a specific domain domain.

```
msg=dgettext(domain, myString)
```

Parameters

domain
: The name of the message domain

string
: the message to be translated

Description

`dgettext` get the translation of a string to the current locale in a specified message domain.

Examples

```
dgettext('scilab','Startup execution:')
```

See Also

gettext

Authors

Sylvestre Ledru

Name

getdefaultlanguage — getdefaultlanguage() returns the default language used by Scilab.

```
getdefaultlanguage()
```

Description

`getdefaultlanguage()` returns the default language used by Scilab. By default, this function should return en_US.

Examples

```
getdefaultlanguage()
```

See Also

setlanguage getlanguage

Authors

Sylvestre Ledru

Name

getlanguage — getlanguage() returns current language used by Scilab.

```
getlanguage()
```

Description

`getlanguage()` returns current language used by Scilab.

Examples

```
setlanguage('en_US')
getlanguage()
```

See Also

setlanguage

Authors

A.C.
Sylvestre Ledru

Name

gettext — get text translated into the current locale and domain.

```
msg=gettext(myString)
```

Parameters

string
: the message to be translated

Description

`gettext` get the translation of a string to the current locale in the current domain.

Examples

```
gettext('Startup execution:')
```

See Also

dgettext

Authors

Sylvestre Ledru

Name

LANGUAGE — Variable defining the language (OBSOLETE)

Description

LANGUAGE is obsolete. If you need LANGUAGE, add LANGUAGE=getlanguage();

See Also

getlanguage

Name

setdefaultlanguage — sets and saves the internal LANGUAGE value.

```
setdefaultlanguage(language)
```

Parameters

language
: with language='fr', 'en', 'ru_RU', 'zh_TW', ...

Description

`setdefaultlanguage(language)` changes current language and save this value in scilab.

You need to restart scilab, if you want to use menus.

setdefaultlanguage('') resets language to the system value.

setdefaultlanguage is used only Windows. On others operating systems , it returns always %f.

Examples

```
setdefaultlanguage('en_US')
// restart scilab
getlanguage()
setdefaultlanguage('fr_FR')
// restart scilab
getlanguage()
setdefaultlanguage('')
// restart scilab
```

See Also

getlanguage, setlanguage

Authors

A.C.

Name

setlanguage — Sets the internal LANGUAGE value.

```
setlanguage(language)
```

Parameters

language
: with language='fr' or 'en', ...

Description

`setlanguage(language)` changes current language in scilab.

Examples

```
setlanguage('en_US')
getlanguage()
setlanguage('en')
getlanguage()
setlanguage('fr')
getlanguage()
setlanguage('fr_FR')
getlanguage()
```

See Also

getlanguage

Authors

A.C.

Maple Interface

Name

sci2map — Scilab to Maple variable conversion

```
txt=sci2map(a,Map-name)
```

Parameters

a
: Scilab object (matrix, polynomial, list, string)

Map-name
: string (name of the Maple variable)

txt
: vector of strings containing the corresponding Maple code

Description

Makes Maple code necessary to send the Scilab variable `a` to Maple : the name of the variable in Maple is `Map-name`. A Maple procedure `maple2scilab` can be found in `SCIDIR/maple` directory.

Examples

```
txt=[sci2map([1 2;3 4],'a');
     sci2map(%s^2+3*%s+4,'p')]
```

Matlab binary files I/O

Name

loadmatfile — loads a Matlab V6 MAT-file (binary or ASCII) into Scilab

```
loadmatfile(format,filename[,var1[,var2[,...]]])
loadmatfile(filename[,format[,var1[,var2[,...]]]])
loadmatfile(filename[,var1[,var2,[,...[,format]]]])
```

Parameters

filename
: character string containing the path of the file (needed)

format
: file format (if not given and file has extension ".mat", file is considered to be binary)

  "-mat"
  : binary file

  "-ascii"
  : option to force Scilab to read file as an ASCII file

var1, var2
: character strings containing the name of the variables to load (only for binary files)

Description

loads a Matlab MAT-file into Scilab. The Matlab data types are converted into the Scilab equivalents.

See Also

load , savematfile , save , mfile2sci , matfile2sci

Authors

Serge Steer (INRIA)
V.C

Bibliography

This function has been developped following the "MAT-File Format" description: `Mat-File Format [http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/matfile_format.pdf]`

Name

matfile_close — Closes a Matlab V5 binary MAT-file.

```
status = matfile_close(fd)
```

Parameters

fd
: Real: file descriptor (returned by matfile_open).

status
: Boolean: *%T* if closure succeeds, *%F* otherwise.

Description

Closes a Matlab binary MAT-file opened by matfile_open.

See Also

matfile_open , matfile_varreadnext , matfile_varwrite , matfile_listvar

Authors

V.C

Bibliography

This function uses MATIO library (`http://sourceforge.net/projects/matio/`).

Name

matfile_listvar — Lists variables of a Matlab V5 binary MAT-file.

```
[names[, classes[, types]]] = matfile_listvar(fd)
```

Parameters

fd
: Real: file descriptor (returned by matfile_open).

names
: String vector: names of the variables.

classes
: Real vector: classes of the variables.

types
: Real vector: data types of the variables.

Description

Lists variables of a Matlab binary MAT-file opened by matfile_open.

See Also

matfile_open , matfile_close , matfile_varwrite , matfile_varreadnext

Authors

V.C

Bibliography

This function uses MATIO library (`http://sourceforge.net/projects/matio/`).

Name

matfile_open — Opens a Matlab V5 binary MAT-file.

```
fd = matfile_open(filename[, mode])
```

Parameters

filename
: String: the path of the file. Must contain only ANSI character.

mode
: String: file access type (`"r"` by default).

- *"r"*: opens the file for reading.
- *"w"*: opens the file for writing.

fd
: Real: file descriptor (-1 if opening failed).

Description

Opens a Matlab binary MAT-file for reading or writing data.

See Also

matfile_close , matfile_varreadnext , matfile_varwrite , matfile_listvar

Authors

V.C

Bibliography

This function uses MATIO library (`http://sourceforge.net/projects/matio/`).

Name

matfile_varreadnext — Reads next variable in a Matlab V5 binary MAT-file.

```
[name[, value[, vartype]]] = matfile_varreadnext(fd)
```

Parameters

fd
: Real: file descriptor (returned by matfile_open).

name
: String: name of the variable read or `""` if reading failed.

value
: Any Scilab type: value of the variable read or an empty matrix if reading failed.

vartype
: Real: type of the variable if reading succeeds or:

  - *0*: if the variable type is unknown.
  - *-1*: if end of file has been reached.

Description

Reads next variable in a Matlab binary MAT-file opened by matfile_open.

See Also

matfile_open , matfile_close , matfile_varwrite , matfile_listvar

Authors

V.C

Bibliography

This function uses MATIO library (`http://sourceforge.net/projects/matio/`).

Name

matfile_varwrite — Write a variable in a Matlab V5 binary MAT-file.

```
status = matfile_varreadnext(fd, name, value, compressionflag)
```

Parameters

fd
: Real: file descriptor (returned by matfile_open).

name
: String: name of the variable to write in the file.

value
: Any Scilab type: value of the variable to write in the file.

compressionflag
: Boolean: indicate if data compression has to be used (flag equaled to *%T*) or not.

status
: Boolean: *%T* if writing succeeds, *%F* otherwise.

Description

Writes a variable in a Matlab binary MAT-file opened by matfile_open.

See Also

matfile_open , matfile_close , matfile_varreadnext , matfile_listvar

Authors

V.C

Bibliography

This function uses MATIO library (`http://sourceforge.net/projects/matio/`).

Name

savematfile — write a Matlab MAT-file (binary or ASCII)

```
savematfile('filename')
savematfile('filename', 'var1', 'var2', ...)
savematfile('filename', '-struct', 's')
savematfile('filename', '-struct', 's', 'f1', 'f2', ...)
savematfile(..., 'format')
savematfile filename var1 var2 ...
```

Parameters

filename
: character string containing the path of the file (needed)

format
: data format to use

  "-mat"
  : binary MAT-file (default)

  "-ascii"
  : 8-bit ASCII format

  "-ascii" "-double"
  : 16-bit ASCII format

  "-ascii" "-tabs"
  : delimits with tabs

  "-ascii" "-double" "-tabs"
  : 16-digit ASCII format, tab delimited

  "-v4"
  : A format that MATLAB Version 4 can open

  "-v6"
  : A format that MATLAB Version 6 and earlier can open (default)

var1, var2
: character strings containing the name of the variables to load (only for binary files)

"-struct" "s"
: saves all fields of the scalar structure s as individual variables within the file filename.

"-struct" "s" "f1" "f2"
: saves as individual variables only those structure fields specified (s.f1, s.f2, ...).

Description

saves variables in a Matlab MAT-file from Scilab. The Scilab data types are converted into the Matlab equivalents.

See Also

load , save , loadmatfile , mfile2sci

Authors

Serge Steer (INRIA)

V.C

Bibliography

This function has been developped following the "MAT-File Format" description: `Mat-File Format [http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/matfile_format.pdf]`

Matlab to Scilab Conversion Tips

Name

About_M2SCI_tools — Generally speaking about tools to convert Matlab files to Scilab...

Description

Scilab 3.0 includes a new version of useful tools to convert Matlab M-files to Scilab.

Taking a Matlab M-file, `mfile2sci` modifies this files so that it can be compiled by Scilab. After that this compiled code is converted to a "tree" of instructions by `macr2tree`. This "tree" is an imbrication of Scilab lists and tlists and is the basis for conversion. Each instruction of this "tree" is converted to Scilab and inference is done to known what are the variables. Once this "tree" is converted to Scilab, code is generated using `tree2code`.

All tlists used for coding this tree (and we call "MSCI tlists") are listed below:

- funcall
  tlist representing a function call created by `Funcall`

  operation
  tlist representing an operation created by `Operation`

  variable
  tlist representing a variable created by `Variable`

  cste
  tlist representing a constant created by `Cste`

  equal
  tlist representing an instruction created by `Equal`

  ifthenelse
  tlist representing an IF/THEN/ELSE control instruction created inside M2SCI kernel functions

  while
  tlist representing a WHILE control instruction created inside M2SCI kernel functions

  selectcase
  tlist representing a SELECT/CASE control instruction created inside M2SCI kernel functions

  for
  tlist representing a FOR control instruction created inside M2SCI kernel functions

The contents of these tlists is described in corresponding help pages.

Operations are converted using a fonction named `%<opcode>2sci` with opcode the Scilab code for this operator. See help page for overloading to have these codes. All these functions are already written and are in directory SCI/modules/m2sci/macros/percent/.

Function calls are converted using a function called `sci_<Matlab_function_name>`. Some of these functions have been written and are in directory SCI/modules/m2sci/macros/sci_files/. We are working on increasing the set of Matlab functions converted. However, everybody can written such functions using help page sci_files.

Inference is done using tlists of type "infer" containing fields:

- dims
  list of dimensions

  type
  "type" tlist

contents
"contents" tlist if a Cell or a Struct

Type is a tlist of type "type" containing fields:

- vtype
  data type

property
property

To have more details about inference see help page for m2scideclare.

See Also

mfile2sci , translatepaths , overloading , sci_files , Funcall , Operation , Variable , Cste , Infer , Type , Equal , m2scideclare

Authors

V.C.

Name

Contents — Create a tree containing contents inference data

```
contents=Contents(list_of_index,list_of_infer)
```

Parameters

list_of_index
: list of indexes similar to indexes returned by macr2tree.

list_of_infer
: list of "infer" tlists containing inference data for matching index.

contents
: a "contents" tlist

Description

This function create a `tlist` representing inference data for the contents of a Cell or a Struct when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists". (Unknown=-1 in M2SCI) Please ensure that for each entry you insert in `list_of_index`, you also insert an entry in `list_of_infer`.

See Also

get_contents_infer , Funcall , Operation , Variable , Cste , Infer , Type , Equal

Authors

V.C.

Name

Cste — Create a tree representing a constant

```
const=Cste(value)
```

Parameters

value
: constante value

const
: a "cste" tlist

Description

This function create a `tlist` representing a constant when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists".

See Also

Funcall , Operation , Variable , Infer , Contents , Type , Equal

Authors

V.C.

Name

Equal — Create a tree representing an instruction

```
eq=Equal(lhslist,expression)
```

Parameters

lhslist
: list of lhs parameters (list of "M2SCI tlists")

expression
: right member of equal (an "M2SCI tlist")

eq
: an "equal" tlist

Description

This function create a `tlist` representing an instruction when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists".

See Also

Funcall , Operation , Variable , Cste , Infer , Contents , Type

Authors

V.C.

Name

Funcall — Create a tree representing a function call

```
fc=Funcall(name,lhsnb,rhslist,lhslist)
```

Parameters

name
: function name (character string)

lhsnb
: number of outputs (constant)

rhslist
: list of inputs (list of "M2SCI tlists")

lhslist
: list of outputs (list of "M2SCI tlists")

fc
: a "funcall" tlist

Description

This function create a `tlist` representing a function call when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists".

See Also

Operation , Variable , Cste , Infer , Contents , Type , Equal

Authors

V.C.

Name

Infer — Create a tree containing inference data

```
infer=Infer(varargin)
```

Parameters

varargin
: data for inference

  varargin(1)
  : list of dimensions default value is list(Unknown,Unknown)

  varargin(2)
  : type ("type" tlist, see Type help page) default value is Type(Unknown,Unknown)

  varargin(3)
  : contents ("contents" tlist, see Contents help page) default value is Contents(list(),list()). This field is only used if represented data is a Cell or a Struct.

infer
: an "infer" tlist

Description

This function create a `tlist` representing inference data when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists". (Unknown=-1 in M2SCI)

See Also

Funcall , Operation , Variable , Cste , Contents , Type , Equal

Authors

V.C.

Name

Matlab-Scilab_character_strings — Generally speaking about...

Description

Matlab and Scilab character strings are not considered in the same way. Here is a little talk about differences between them.

Matlab considers a character string as Scilab considers a matrix of characters. For example, a Scilab equivalent for Matlab 'mystring' could be ["m","y","s","t","r","i","n","g"]. So in Scilab, a character string is a object of type string (10) and always have size 1 x 1 but in Matlab, a character string have size equal to 1 x number_of_characters.

Considering this, we can see that a Matlab character string matrix column can only be made of same-size character strings what is not true in Scilab. We can say that a Scilab character string matrix is equivalent to a Matlab cell of character strings.

All these differences can lead to different results while executing same commands in Scilab or in Matlab, particularly for "dimension" functions such as length() or size().

See Also

mstr2sci

Authors

V.C.

Name

Operation — Create a tree representing an operation

```
op=Operation(operator,operands,out)
```

Parameters

operator
: operator symbol (character string)

operands
: list of operands (list of "M2SCI tlists")

out
: list of outputs (list of "M2SCI tlists")

op
: an "operation" tlist

Description

This function create a `tlist` representing an operation when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists".

See Also

Funcall , Variable , Cste , Infer , Contents , Type , Equal

Authors

V.C.

Name

Type — Create a tree containing type inference data

```
tp=Type(vtype,property)
```

Parameters

vtype
: data type (see m2scideclare)

property
: property of data (see m2scideclare)

tp
: a "type" tlist

Description

This function create a `tlist` representing type inference data when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists". (Unknown=-1 in M2SCI)

See Also

Funcall , Operation , Variable , Cste , Infer , Contents , Equal , m2scideclare

Authors

V.C.

Name

Variable — Create a tree representing a variable

```
var=Variable(name,infer)
```

Parameters

var
: variable name (character string)

infer
: inference data (a tlist of type "infer", see Infer help page)

var
: a "variable" tlist

Description

This function create a `tlist` representing a variable when using M2SCI. All input parameters values are verified to be compatible with "M2SCI tlists".

See Also

Funcall , Operation , Cste , Infer , Contents , Type , Equal

Authors

V.C.

Name

get_contents_infer — Search for information in a "M2SCi tlist" contents

```
[infer,pos]=get_contents_infer(m2scitlist,index)
```

Parameters

m2scitlist
: a "M2SCI tlist"

index
: an index similar to indexes returned by `macr2tree`.

infer
: an "infer" tlist

pos
: position of information in list

Description

This functions searches for inference informations of a given index in the contents of a Cell or a Struct taken in account the `*`. If no information has been found, returned values are infer=infer() and pos=0.

See Also

Infer , Contents

Authors

V.C.

Name

m2scideclare — Giving tips to help M2SCI...

Description

The main difficulty for M2SCI (`mfile2sci`) is to find what variables are: dimensions, type...

To help this tool, just add comments beginning with `%m2scideclare` in the M-file to convert, (`%m2sciassume` was used in previous Scilab versions and is now obsolete).

The syntax of this command is:

```
%m2scideclare variable_name|dimensions|data_type|property
```

with :

- • variable_name: name of the variable declared. It can be a Struct field (e.g. `x(1,2).name`) or describe the contents of a Cell using syntax `x(1,2).entries`. NOTE that for Cells and Structs, `*` can be used as an index (see examples below).
  - dimensions: dimensions of the variable declared separated by blanks, if a dimension is unknown, replace it by `?`. NOTE that String dimensions must be similar to Matlab ones e.g. `1 6` for character string 'string'.
  - data_type: data type of the variable which can be:

m2scideclare data type	Scilab "equivalent" type
Double	1
Boolean	4
Sparse	5
Int	8
Handle	9
String	10
Struct	Matlab struct (16)
Cell	Matlab cell (17)
Void	No type (0)
?	Unknown type

  - property: property of the variable which can be:

m2scideclare property	Scilab "equivalent"
Real	Real data
Complex	Complex data
?	Unknown property

    This field is ignored for following datatypes: `Cell`, `Struct`, `String` and `Boolean`.

All data given by `m2scideclare` are compared with infered data, in case of conflict, infered data are kept and a warning message is displayed. If you are sure about your data, report a bug.

Some examples are given below:

- • `%m2scideclare var1|2 3|Double|Real`var1 is declared as a 2x3 Double matrix containing real data

- `%m2scideclare var2|2 3 10|Double|Complex`var2 is declared as a 2x3x10 Double hypermatrix containing complex data
- `%m2scideclare var3(1,2).name|1 10|String|?`var3 is declared as a Struct array containing a 1x10 character string in field 'name' of struct at index (1,2)
- `%m2scideclare var4(1,5).entries|1 ?|Boolean|?`var4 is declared as a Cell containing a row boolean vector at index (1,5)
- `%m2scideclare var4(1,6).entries|? ?|Int|?`var4 is declared as a Cell containing a row boolean vector at index (1,5) and integer data at index (1,6)
- `%m2scideclare var5(*,*).name|1 ?|String|?`var5 is declared as a Struct array containing a 1xn character string in all fields 'name'
- `%m2scideclare var6(2,*).entries|1 3|Double|Real`var6 is declared as a Cell array containing a 1x3 double vector in each element of its second row

Authors

V.C.

Name

matfile2sci — converts a Matlab 5 MAT-file into a Scilab binary file

```
matfile2sci(mat_file_path,sci_file_path)
```

Parameters

mat_file_path
: character string containing the path of the Matlab input file

sci_file_path
: character string containing the path of the Scilab output file

Description

Converts a Matlab 5 MAT-file into a Scilab binary file compatible with the function `load` The Matlab data types are converted into the Scilab equivalents.

See Also

loadmatfile , load , mfile2sci

Authors

Serge Steer (INRIA)

Bibliography

This function has been developped according to the document "MAT-File Format": >Mat-File Format [http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/matfile_format.pdf]

Name

mfile2sci — Matlab M-file to Scilab conversion function

```
mfile2sci([M-file-path [,result-path [,Recmode [,only-double [,verbose-mode [,p
```

Parameters

M-file-path
: a character string which gives the path of Matlab M-file to convert

result-path
: a character string which gives the directory where the result has to be written. Default value is current directory.

Recmode
: Boolean flag, used by translatepaths function for recursive conversion. Must be %F to convert a single mfile. Default value : %f

only-double
: Boolean flag, if %T mfile2sci considers that numerical function have been used only with numerical data (no Scilab overloading function is needed). Default value: %F

verbose-mode
: display information mode

  0
  : no information displayed

  1
  : information written as comment is resulting SCI-file

  2
  : information written as comment is resulting SCI-file and in logfile

  3
  : information written as comment is resulting SCI-file, in logfile and displayed in Scilab window

prettyprint
: Boolean flag, if %T generated code is beautified. Default value: %F

Description

M2SCI (and particularly mfile2sci) is Matlab M-file to Scilab function conversion tools. It tries whenever possible to replace call to Matlab functions by the equivalent Scilab primitives and functions.

To convert a Matlab M-file just enter the Scilab instruction: mfile2sci(file)

where file is a character string giving the path name of the M-file mfile2sci will generate three files in the same directory

<function-name>.sci
: the Scilab equivalent of the M-file

<function-name>.cat
: the Scilab help file associated to the function

sci_<function-name>.sci
: the Scilab function required to convert the calls to this Matlab M-file in other Matlab M-files. This function may be improved "by hand". This function is only useful for conversion not for use of translated functions.

Some functions like eye, ones, size, sum,... behave differently according to the dimension of their arguments. When mfile2sci cannot infer dimensions it replaces these function call by a call to an emulation function named mtlb_<function_name>. For efficiency these functions may be replaced by the proper scilab equivalent instructions. To get information about replacement, enter: help mtlb_<function_name> in Scilab command window

Some other functions like plot, has no straightforward ï¿½quivalent in scilab. They are also replaced by an emulation function named mtlb_<function_name>.

When translation may be incorrect or may be improved mfile2sci adds a comment which begins by "//!" (according to verbose-mode)

When called without rhs, `mfile2sci()` launches a GUI to help to select a file/directory and options.

Examples

```
// Create a simple M-file
write(TMPDIR+'/rot90.m',['function B = rot90(A,k)'
 '[m,n] = size(A);'
 'if nargin == 1'
 '    k = 1;'
 'else'
 '    k = rem(k,4);'
 '    if k < 0'
 '        k = k + 4;'
 '    end'
 'end'
 'if k == 1'
 '    A = A.'';'
 '    B = A(n:-1:1,:);'
 'elseif k == 2'
 '    B = A(m:-1:1,n:-1:1);'
 'elseif k == 3'
 '    B = A(m:-1:1,:);'
 '    B = B.'';'
 'else'
 '    B = A;'
 'end']);
// Convert it to scilab
mfile2sci(TMPDIR+'/rot90.m',TMPDIR)
// Show the new code
write(%io(2),read(TMPDIR+'/rot90.sci',-1,1,'(a)'))
// get it into scilab
getf(TMPDIR+'/rot90.sci')
// Execute it
m=rand(4,2);rot90(m,1)
```

See Also

translatepaths

Authors

V. Couvert
S. Steer

Name

sci_files — How to write conversion functions

Description

To convert calls to Matlab functions, `mfile2sci` uses a function called `sci_<Matlab_function_name>`. All these functions are defined in `sci_files` in directory SCI/modules/m2sci/macros/sci_files/. The set of `sci_files` given in Scilab distribution does not allow to convert calls to all Matlab functions yet. However, a Scilab user can add `sci_files` (for Matlab functions or for user defined functions) to Scilab using the following tips.

In M2SCI, a function call is considered as a "tree" (it is also the case for the instructions of the file to convert), represented in Scilab by a `tlist` with following fields:

- name
  Matlab function name

  lhsnb
  number of Matlab function output parameters

  lhs
  list of Matlab function output parameters

  rhs
  list of Matlab function input parameters

A `sci_function` has one input called `tree` which is also the output of the function. A `sci_function` has to convert this incoming "tree" so that it is compatible with Scilab by changing name, lhsnb, lhs and/or rhs. The other task that has to be done by this function is inference. Incoming tree contains inference data in its lhs that have to be updated with what can be infered for the outputs of this function.

Some useful functions have been written to help to create M2SCI tlists while writing this conversion function:

- Funcall
  create a tree representing a function call

  Operation
  create a tree representing an operation

  Variable
  create a tree representing a variable

  Cste
  create a tree representing a constante value

  Infer
  create a tree representing inference data

  Type
  create a tree representing type for inference

  Equal
  create a tree representing an instruction

Some other functions have been designed to get properties of operands/inputs. Considering A is tlist used in macro tree, you can use the following functions:

Function	returns %T if...

`is_empty(A)`	all dimensions of A are 0
`not_empty(A)`	all dimensions of A are known and at least one dimension of A is not 0
`is_a_scalar(A)`	all dimensions of A are 1
`not_a_scalar(A)`	all dimensions of A are known and at least one dimension of A is not 1
`is_a_vector(A)`	all dimensions of A are known and all dimensions of A but one are equal to 1
`not_a_vector(A)`	all dimensions of A are known and at least two dimensions of A are greater than one
`is_real(A)`	A is real
`is_complex(A)`	A is complex
`isdefinedvar(A)`	A is a variable already created in M-file currently converted
`allunknown(A)`	all dimensions of A are unknown

Some other functions have been written for specific needs while writing conversion files:

- first_non_singleton
  is an equivalent to firstnonsingleton for an M2SCI tlist. Calling sequence: `dim = first_non_singleton(A)`

  gettempvar
  generates a temporary variable having a name which does not already exist. Calling sequence: `v = gettempvar()`

  insert
  allows to insert instructions. Calling sequence: `insert(Equal(...),opt)` with `opt~=1` to insert before current instruction and `opt=1` to insert after it.

  getoperands
  can be used to get each operand as a variable. Calling sequence: `[A,B] = getoperands(operation_tlist)`

  getrhs
  can be used to get each parameter as a variable. Calling sequence: `[A,...] = getrhs(funcall_tlist)`

  convert2double
  change type of input when this type is not implemented for a particular function is Scilab. Calling sequence: `A = convert2double(A)`

To have more information about how to write such files, refer to directory SCI/modules/m2sci/macros/sci_files/ which gives many examples from very simple ones (e.g. sci_abs.sci) to very complex ones (e.g. sci_zeros.sci).

See Also

m2scideclare , Funcall , Operation , Variable , Cste , Infer , Type , Equal

Authors

V.C.

Name

translatepaths — convert a set of Matlab M-files directories to Scilab

```
translatepaths(dirs_path [,res_path])
```

Parameters

dirs_path
: a character string vector which gives the paths of Matlab M-file directories to convert

res_path
: a character string which gives the path of the directory where the Scilab functions are written to. Default value is current directory.

Description

`translatepaths`, converts all Matlab M-file contained in a set of directories to Scilab functions. Each function is converted by `mfile2sci`.

Trace of conversion information is stored in a file named `"log"` in the `res_path` directory

When called without rhs, `translatepaths()` launches a GUI to help to select a file/directory and options.

See Also

mfile2sci

Authors

V. Couvert
S. Steer

Metanet : Graph and Network toolbox

Name

add_edge — adds an edge or an arc between two nodes

```
g1 = add_edge(i,j,g)
g1 = add_edge(ij,g)
```

Parameters

i
: vector of integers, number of start nodes

j
: vector of integers, number of end nodes

ij
: 2 by n matrix of integers, first row contains the start node numbers, second row contains the end node numbers.

g
: :a graph list (see graph_data_structure).

g1
: graph list of the new graph with the added edges

Description

`add_edge(i,j,g)` returns the graph `g1` with a new edges connecting node number `i(k)` to node number `j(k)`. If the graph is directed, the edge is an arc. The number of edges plus 1 is taken as the name of the new edge.

`add_edge(ij,g)` returns the graph `g1` with a new edges connecting node number `ij(1,k)` to node number `ij(2,k)`. If the graph is directed, the edge is an arc. The number of edges plus 1 is taken as the name of the new edge.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
g.nodes.graphics.display='number';
show_graph(g);
ne=edge_number(g);
g=add_edge([1 1 9;7 16 9],g);
show_graph(g);
hilite_edges(ne+1:ne+3)
```

See Also

add_node , delete_arcs , delete_nodes

Name

add_edge_data — associates new data fields to the edges data structure of a graph

```
g = add_edge_data(g,name [,value])
```

Parameters

g
: a graph data structure (see graph_data_structure)

name
: a character string, the name of the data field.

value
: a row vector or a matrix with column size equal to the number of edges. This parameter is optional. If it is omitted the data field is set to `[]`.

Description

`g = add_edge_data(g,name [,value])` associates the data fields named `name` to the edges data structure of the graph `g` and assign it the value given by the parameter `value`. If the last argument is not given the empty matrix `[]` is assigned.

`value` can be a matrix of any type. The `i`th column is associated with the`i`th edge.

Examples

```
//create a simple graph
ta=[1  1 2 7 8 9 10 10 10 10 11 12 13 13];
he=[2 10 7 8 9 7  7 11 13 13 12 13  9 10];
g=make_graph('simple',1,13,ta,he);
g.nodes.graphics.x=[40,33,29,63,146,233,75,42,114,156,237,260,159];
g.nodes.graphics.y=[7,61,103,142,145,143,43,120,145,18,36,107,107];
show_graph(g,'new')


g=add_edge_data(g,'length',round(10*rand(1,14,'u')));
g=add_edge_data(g,'label','e'+string(1:14));
edgedatafields(g)
g.edges.data.label
```

See Also

edgedatafields , edges_data_structure

Name

add_node — adds disconnected nodes to a graph

```
g1 = add_node(g [,xy] [,name]])
```

Parameters

g
: graph list (see graph_data_structure).

xy
: optional new nodes coordinates, can be a 2 vector or a matrix with two rows `[x;y]`.

name
: optional vector of strings, names of the added nodes

g1
: graph list of the new graph with the added nodes

Description

`add_node` adds disconnected nodes to graph `g` and returns the new graph `g1`.

The coordinates of the new nodes can be given as a vector of coordinates in `xy`. If the nodes of graph `g` have no coordinates (elements `node_x` and `node_y` are `[]`), to give `xy` has no effect. If the nodes of graph `g` have coordinates and `xy` is not given, the new node has `(0,0)` as coordinates.

If `name` is given, it is the vector of the new node names, otherwise the node number is taken as the name of each new node.

`add_node` initializes the node graphic properties to their default values.

Examples

```
//create a graph
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
//set node coordinates for visualization
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
show_graph(g);

g1=add_node(g,[270  350 700;140 0 400]);
show_graph(g1);
hilite_nodes(18:20);
```

See Also

graph_data_structure , add_edge , delete_arcs , delete_nodes

Name

add_node_data — associates new data fields to the nodes data structure of a graph

```
g = add_node_data(g,name [,value])
```

Parameters

g
: a graph data structure (see graph_data_structure)

name
: a character string, the name of the data field.

value
: a row vector or a matrix with column size equal to the number of nodes. This parameter is optional. If it is omitted the data field is set to `[]`.

Description

`g = add_node_data(g,name [,value])` associates the data fields named `name` to the nodes data structure of the graph `g` and assign it the value given by the parameter `value`. If the last argument is not given the empty matrix `[]` is assigned.

`value` can be a matrix of any type. The `i`th column is associated with the`i`th node.

Examples

```
//create a simple graph
ta=[1  1 2 7 8 9 10 10 10 10 11 12 13 13];
he=[2 10 7 8 9 7  7 11 13 13 12 13  9 10];
g=make_graph('simple',1,13,ta,he);
g.nodes.graphics.x=[40,33,29,63,146,233,75,42,114,156,237,260,159];
g.nodes.graphics.y=[7,61,103,142,145,143,43,120,145,18,36,107,107];
show_graph(g,'new')
nodedatafields(g)


g=add_node_data(g,'demand',round(10*rand(1,13,'u')));
g=add_node_data(g,'label','e'+string(1:13));
nodedatafields(g)
g.nodes.data.label
```

See Also

nodedatafields , nodes_data_structure

Name

adj_lists — computes adjacency lists

```
[lp,la,ls] = adj_lists(g)
[lp,la,ls] = adj_lists(directed,n,tail,head)
```

Parameters

g
: :a graph (see graph_data_structure).

directed
: integer, 0 (undirected graph) or 1 (directed graph)

n
: integer, the number of nodes of the graph

tail
: the row vector of the numbers of the tail nodes of the graph (its size is the number of edges of the graph)

head
: the row vector of the numbers of the head nodes of the graph (its size is the number of edges of the graph)

lp
: row vector, pointer array of the adjacency lists description of the graph (its size is the number of nodes of the graph + 1)

la
: row vector, arc array of the adjacency lists description of the graph (its size is the number of edges of the graph)

ls
: row vector, node array of the adjacency lists description of the graph (its size is the number of edges of the graph)

Description

`adj_lists` computes the row vectors of the adjacency lists description of the graph `g`. It is also possible to give `adj_lists` the description of the graph given by the number of nodes `n` and the row vectors `tail` and `head`.

For a node numbered `k`, the connected edges are given by `la(lp(k):(lp(k+1)-1))`, while the other bounds of these edges are connected to nodes ls((lp(k):(lp(k+1)-1))).

Examples

```
ta=[2 3 3 5 3 4 4 5 8];
he=[1 2 4 2 6 6 7 7 4];
g=make_graph('foo',1,8,ta,he);
g.nodes.graphics.x=[129 200 283 281 128 366 122 333];
g.nodes.graphics.y=[61 125 129 189 173 135 236 249];
g.nodes.graphics.display='number';
g.edges.graphics.display='number';
```

```
show_graph(g);
//directed graph
[lp,la,ls]=adj_lists(g)
[lp,la,ls]=adj_lists(1,g.nodes.number,ta,he)
for k=1:node_number(g)
  sel=lp(k):(lp(k+1)-1);
  g1=g;
  g1.nodes.graphics.colors(2,k)=color('red');
  g1.edges.graphics.foreground(la(sel))=color('green');
  g1.nodes.graphics.colors(1,ls(sel))=color('red');
  show_graph(g1);
  halt()
end

//non directed graph
g.directed=0;
[lp,la,ls]=adj_lists(g);
for k=1:node_number(g)
  sel=lp(k):(lp(k+1)-1);
  g1=g;
  g1.nodes.graphics.colors(2,k)=color('red');
  g1.edges.graphics.foreground(la(sel))=color('green');
  g1.nodes.graphics.colors(1,ls(sel))=color('red');
  show_graph(g1);
  halt()
end
```

See Also

chain_struct , graph_2_mat

Name

arc_graph — graph with nodes corresponding to arcs

```
g1 = arc_graph(g)
```

Parameters

g
: a directed graph (see graph_data_structure).

g1
: a directed graph

Description

`arc_graph` returns the directed graph `g1` with the nodes corresponding to the arcs of the directed graph `g`. `g1` is defined in the following way:

- its nodes correspond to the arcs of `g`

- 2 nodes of the new graph are adjacent if and only if the corresponding arcs of the graph `g` are consecutive.

The coordinates of the nodes of `g1` are given by the middle points of the corresponding edges of `g`.

If such an arc graph does not exist, an empty vector is returned.

Examples

```
//create the initial graph
ta=[1 1 2 4 4 5 6 7 2 3 5 1];
he=[2 6 3 6 7 8 8 8 4 7 3 5];
g=make_graph('foo',1,8,ta,he);
g.nodes.graphics.x=[281  284  360  185  405  182  118  45];
g.nodes.graphics.y=[262  179  130  154  368  248  64  309];
//customize display
g.nodes.graphics.display='name';
g.edges.graphics.name=string(1:edge_number(g));
g.edges.graphics.display='name';
show_graph(g);

//compute the arc_graph
g1=arc_graph(g);
g1.edges.graphics.name=string(1:edge_number(g1));
g1.edges.graphics.display='name';
g1.nodes.graphics.display='name';
show_graph(g1,'new');

// merge the two graph
g1.nodes.graphics.colors(2,:)=color('red');
g1.edges.graphics.foreground(:)=color('red');
show_graph(graph_union(g,g1,%f),'new')
```

See Also

line_graph

Name

arc_number — number of arcs of a graph

```
ma = arc_number(g)
```

Parameters

g
: a graph (see graph_data_structure).

ma
: integer, number of arcs

Description

`arc_number` returns the number `ma` of arcs of the graph. If the graph is directed, it is the number of edges. If the graph is undirected, it is twice the number of edges.

See Also

edge_number , node_number

Name

articul — finds one or more articulation points

```
nart = articul([i],g)
```

Parameters

g
: :a graph (see graph_data_structure).

i
: integer

nart
: integer row vector

Description

An articulation of a connected graph is a node whose removal will disconnect the graph. In general, an articulation vertex is a node of a graph whose removal increases the number of components.

`articul` finds one or more articulation points (if they exist) of the graph `g`. `nart` is the row vector of numbers of articulation nodes: deleting one of these nodes increases the number of connected components of the graph. `i` is the optional node number from which the algorithm starts. The default is 1. Note that the result depends strongly on this starting node.

Examples

```
ta=[2  1 3 2 2 4 4 5 6 7 8 8 9 10 10 10 10 11 12 13 14 15 16 17 17];
he=[1 10 2 5 7 3 2 4 5 8 6 9 7  7 11 13 15 12 13 14 11 16 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
show_graph(g);
nart = articul(g)
hilite_nodes(nart);
```

Bibliography

Chartrand, G. "Cut-Vertices and Bridges." §2.4 in Introductory Graph Theory. New York: Dover, pp. 45-49, 1985.

Harary, F. Graph Theory. Reading, MA: Addison-Wesley, 1994.

Name

bandwr — bandwidth reduction for a sparse matrix

```
[iperm,mrepi,prof,ierr] = bandwr(sp,[iopt])
[iperm,mrepi,prof,ierr] = bandwr(lp,ls,n,[iopt])
```

Parameters

sp
: sparse matrix

lp
: integer row vector

ls
: integer row vector

n
: integer

iopt
: integer

iperm
: integer row vector

mrepi
: integer row vector

prof
: integer row vector

ierr
: integer

Description

`bandwr` solves the problem of bandwidth reduction for a sparse matrix: the matrix is supposed to be upper triangular with a full diagonal (it is easy to complete a non symmetric matrix, and then discards the added terms).

In the first calling sequence, `sp` denotes a sparse matrix; the optional argument `iopt` is 0 or 1: 1 if reducing the profile of the matrix is more important than reducing the bandwidth and 0 if bandwidth reduction is most important.

The second calling sequence corresponds to the description of a graph: `lp` is a row vector, pointer array of the adjacency lists description of a graph (its size is the number of nodes of the graph + 1); `ls` is a row vector, node array of the adjacency lists description (its size is the number of edges of the graph i.e. the number of non-zero terms of the corresponding sparse matrix). `n` is the number of nodes (dimension of `sp`).

`iperm` is the permutation vector for reordering the rows and columns which reduces the bandwidth and/or profile (new numbering of the nodes of the graph); `mrepi` is the inverse permutation (mrepi(iperm) is the identity). `prof` is the array giving the profile of the sparse matrix after the bandwidth reduction if `iopt` is 1. If `iopt` is 0 this array is zero except for the first term giving the bandwidth. The simple command `max(prof(2:$)-prof(1:($-1)))` returns the bandwidth of the matrix. `ierr` is an integer indicating an error if its value is not zero.

Examples

```
//Build the initial graph
ta=[2  1 3 2 2 4 4 5 6 7 8 8 9 10 10 10 10 11 12 13 13 14 15 16 16 17 17];
he=[1 10 2 5 7 3 2 4 5 8 6 9 7 7 11 13 15 12 13  9 14 11 16 1 17 14 15];
g=make_graph('foo',0,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
//Initial Display
g.nodes.graphics.name=string(1:17);
g.nodes.graphics.display='name';
show_graph(g);

a=graph_2_mat(g,'node-node');
ww=tril(a)'+eye();
ww1=full(ww);
scf(1);
hist3d((ww1+tril(ww1',-1)+tril(ww1,-1)'),52,85);
// BANDWIDTH REDUCTION FOR THE MATRIX
[iperm,mrepi,prof,ierr]=bandwr(ww);
disp(max(prof(2:$)-prof(1:($-1))));

// GRAPH WITH THE NEW NUMBERING
g2=g;
g2.nodes.graphics.name=string(iperm);
show_graph(g2,'new')
// NEW MATRIX
n=g.nodes.number;
yy=ww1(mrepi,mrepi);
scf(3)
hist3d((yy+tril(yy',-1)+tril(yy,-1)'),52,85);
// STARTING WITH THE SAME MATRIX
[ij,v,mn]=spget(ww);
g1=make_graph('foo',0,n,ij(:,1)',ij(:,2)');
g1.nodes.graphics.x=g.nodes.graphics.x;g1.nodes.graphics.y=g.nodes.graphics.y;
// GRAPH
//show_graph(g1,'rep');
[lp,la,ls] = adj_lists(1,n,g1.edges.tail,g1.edges.head);
[iperm,mrepi,prof,ierr]=bandwr(lp,ls,n,0);
g2=g;g2.nodes.graphics.name=string(iperm);
show_graph(g2,'new');
```

Name

best_match — maximum matching of a graph

```
[card,match] = best_match(g)
```

Parameters

g
: a graph (see graph_data_structure).

card
: integer

match
: integer row vector

Description

A matching on a graph is a set of edges such that no two of them share a node in common. The largest possible matching on a graph with n nodes consists of n/2 edges, and such a matching is called a perfect matching. Although not all graphs have perfect matchings, a maximum matching exists for each graph.

`best_match` finds an maximum matching for the graph `g`. The output are `card` and the vector `match`. `card` is the cardinality of an optimal matching. `match(i)` is the node adjacent to node `i` in the maximum matching or 0 if `i` is unmatched.

Examples

```
//create the graph
ta=[27 27 3 12 11 12 27 26 26 25 25 24 23 23 21 22 21 20 19 18 18];
ta=[ta  16 15 15 14 12 9 10 6 9 17 8 17 10 20 11 23 23 12 18 28];
he=[ 1  2 2  4  5 11 13  1 25 22 24 22 22 19 13 13 14 16 16  9 16];
he=[he  10 10 11 12  2 6  5 5 7  8 7  9  6 11  4 18 13  3 28 17];
n=28;
g=make_graph('foo',0,n,ta,he);

// Graph display
xx=[46 120 207 286 366 453 543 544 473 387 300 206 136 250 346 408];
g.nodes.graphics.x=[xx 527 443 306 326 196 139 264  55  58  46 118 513];
yy=[36  34  37  40  38  40  35 102 102  98  93  96 167 172 101 179];
g.nodes.graphics.y=[yy 198 252 183 148 172 256 259 258 167 109 104 253];
g.nodes.graphics.display='name';
show_graph(g);
[card,match] = best_match(g);
mprintf("number of edge in matching=%d number of nodes=%d\n",card,node_number(g

// compute the edge number of the matching
v=index_from_tail_head(g,1:n,match)
//show the matching edges
hilite_edges(v);
//
// WITH A LARGER GRAPH
g=load_graph(metanet_module_path()+'/demos/mesh1000.graph');
g.directed=0;
```

```
ta=g.edges.tail;he=g.edges.head;n=node_number(g);
show_graph(g,'new',1/2,[1000,400]);
[card,match] = best_match(g);

hilite_edges(index_from_tail_head(g,1:n,match));
```

See Also

perfect_match

Bibliography

Micali, S. and Vazirani, V. V.,"An O(square root of V * E) Algorithm for Finding Maximum Matching in General Graphs", Proc. 21st Annual Symposium on Foundation of Computer Science, IEEE, 1980, pp. 17-27.

Lovász, L. and Plummer, M. D. Matching Theory. Amsterdam, Netherlands: North-Holland, 1986.

Name

chain_struct — chained structure from adjacency lists of a graph

```
[fe,che,fn,chn] = chain_struct(g)
[fe,che,fn,chn] = chain_struct(lp,la,ls)
```

Parameters

g
: a graph_data_structure.

lp
: row vector, pointer array of the adjacency lists description of the graph (its size is the number of nodes of the graph + 1)

la
: row vector, arc array of the adjacency lists description of the graph (its size is the number of edges of the graph)

ls
: row vector, node array of the adjacency lists description of the graph (its size is the number of edges of the graph)

fe
: row vector of the numbers of the first edges starting from nodes (its size is the number of nodes of the graph)

che
: row vector of the numbers of the chained edges (its size is the number of edges of the graph)

fn
: row vector of the numbers of the first nodes reached by the edges of `fe` (its size is the number of nodes of the graph)

chn
: row vector of the nodes reached by the edges of `che`

Description

`chain_struct` computes the row vectors of the edge chained structure description of the graph `g`. It is also possible to give directly `chain_struct` the adjacency lists of the graph. This is more efficient if the adjacency lists are already available since `chain_struct` uses them to make computations.

The vectors `fe`, `che`, `fn` and `chn` describe the chained structure in the following way:

`fe(i)` is the number of the first edge starting from node i

`che(fe(i))` is the number of the second edge starting from node i, `che(che(fe(i)))` is the number of the third edge starting from node i and so on until the value is 0

`fn(i)` is the number of the first node reached from node i

`ch(i)` is the number of the node reached by edge `che(i)`.

Examples

```
ta=[1 1 2 3 5 4 6 7 7 3 3 8 8 5];
he=[2 3 5 4 6 6 7 4 3 2 8 1 7 4];
g=make_graph('foo',1,8,ta,he);
g.nodes.graphics.x=[116 231 192 323 354 454 305 155];
g.nodes.graphics.y=[118 116 212 219 117 185 334 316];
show_graph(g);
[fe,che,fn,chn] = chain_struct(g)
for i=1:node_number(g)
  hilite_nodes(i); xpause(1d6)
  cur=fe(i);while cur<>0;hilite_edges(cur);cur=che(cur);xpause(5d5);end

  unhilite_nodes(i);
  cur=fe(i);while cur<>0;unhilite_edges(cur);cur=che(cur);end
end
```

See Also

adj_lists , graph_2_mat

Name

check_graph — checks a Scilab graph data structure

```
check_graph(g [,opt])
```

Parameters

g
: a graph_data_structure.

opt
: an optional boolean. Default value is `%t`

Description

`check_graph(g,%f)` checks its argument `g` to see if it is a valid graph_data_structure. The checking is not only syntactic (number of elements of the list, compatible sizes of the vectors), but also semantic in the sense that `check_graph` checks that `node_number`, `tail` and `head` elements of the list can really represent a graph.

`check_graph(g,%t)` also check graphics and data fields validity.

See Also

graph_data_structure , nodes_data_structure , edges_data_structure

Name

circuit — finds a circuit or the rank function in a directed graph

```
[p,r] = circuit(g)
```

Parameters

g
: a graph_data_structure.

p
: row vector of integer numbers of the arcs of the circuit if it exists

r
: row vector of rank function if there is no circuit

Description

A cycle of a graph `g`, also called a circuit, is a subset of the edges of `g` that forms a path such that the first node of the path corresponds to the last.

`circuit` tries to find surch a circuit for the directed graph `g`. It returns the circuit `p` as a row vector of the corresponding arc numbers if it exists and it returns the empty vector `[]` otherwise.

If the graph has no circuit, the rank function is returned in `r`, otherwise its value is the empty vector `[]`.

Examples

```
// graph with circuit
ta=[1 1 2 3 5 4 6 7 7 3 3 8 8 5];
he=[2 3 5 4 6 6 7 4 3 2 8 1 7 4];
g=make_graph('foo',1,8,ta,he);
g.nodes.graphics.x=[116 231 192 323 354 454 305 155];
g.nodes.graphics.y=[ 118 116 212 219 117 185 334 316];
g.nodes.graphics.display='number';
g.edges.graphics.display='number';

show_graph(g);
p=circuit(g)
hilite_edges(p)


// graph without circuit
g1=make_graph('foo',1,4,[1 2 2 3],[2 3 4 4]);
g1.nodes.graphics.x=[116 231 192 323];
g1.nodes.graphics.y=[ 118 116 212 219];
g1.nodes.graphics.display='number';
g1.edges.graphics.display='number';

show_graph(g1,'new');
[p,r]=circuit(g)
```

Bibliography

Stanley, R. P. Enumerative Combinatorics, Vol. 1. Cambridge, England: Cambridge University Press, 1999.

Name

con_nodes — set of nodes of a connected component

```
ns = con_nodes(i,g)
```

Parameters

i
: integer, number of the selected connected component

g
: a graph_data_structure.

ns
: row vector, node numbers of the selected connected component

Description

`con_nodes` returns the row vector `ns` of the numbers of the nodes which belong to the connected component number `i`. If `i` is not the number of a connected component, the empty vector `[]` is returned.

Examples

```
ta=[1 1 2 2 2 3 4 4 5 7 7 9 10 12 12 13 13 14 15];
he=[2 6 3 4 5 1 3 5 1 8 9 8 11 10 11 11 15 13 14];
g=make_graph('foo',1,15,ta,he);
g.nodes.graphics.x=[197 191 106 194 296 305 305 418 422 432 552 550 549 416 548
g.nodes.graphics.y=[76 181 276 278 276 83 174 281 177 86 175 90 290 397 399];
show_graph(g);

for k=1:3
  hilite_nodes(con_nodes(k,g));xpause(1d6);unhilite_nodes(con_nodes(k,g));
end
```

See Also

connex , is_connex , strong_connex , strong_con_nodes

Name

connex — connected components

```
[nc,ncomp] = connex(g)
```

Parameters

g
: a graph_data_structure.

nc
: integer, number of connected components

ncomp
: row vector of connected components

Description

`connex` returns the number `nc` of connected components of graph `g` and a row vector `ncomp` giving the number of the connected component for each node. For instance, if `i` is a node number, `ncomp(i)` is the number of the connected component to which node number `i` belongs.

Examples

```
ta=[1 1 2 2 2 3 4 4 5 6 7 7 7 8 9 10 12 12 13 13 14 15];
he=[2 6 3 4 5 1 3 5 1 7 5 8 9 5 8 11 10 11 11 15 13 14];
g=make_graph('foo',1,15,ta,he);
g.nodes.graphics.x=[197 191 106 194 296 305 305 418 422 432 552 550 549 416 548
g.nodes.graphics.y=[76 181 276 278 276 83 174 281 177 86 175 90 290 397 399];
show_graph(g);
[nc,ncomp]=connex(g)
g.nodes.graphics.colors(2,:)=10+ncomp;
show_graph(g);

g=delete_edges([13 11],g);
show_graph(g);
[nc,ncomp]=connex(g)
g.nodes.graphics.colors(2,:)=10+ncomp;
show_graph(g);
```

See Also

con_nodes , is_connex , strong_connex , strong_con_nodes

Bibliography

Chartrand, G. "Cut-Vertices and Bridges." §2.4 in Introductory Graph Theory. New York: Dover, pp. 45-49, 1985.

Harary, F. Graph Theory. Reading, MA: Addison-Wesley, 1994.

Name

contract_edge — contracts edges between two nodes

```
g1 = contract_edge(i,j,g)
```

Parameters

i
: integer, number of start or end node of edge

j
: integer, number of end or start node of edge

g
: a graph_data_structure.

g1
: The new graph

Description

`contract_edge` returns the graph `g1`, the edges between the nodes number `i` and `j` being deleted, the nodes being reduced to one node with the same name as node `i` and located at the middle point between the 2 previous nodes.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 13 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
g.nodes.graphics.name=string(1:node_number(g));
g.nodes.graphics.display='name';

show_graph(g);hilite_nodes([10 13])
g1=contract_edge(10,13,g);

show_graph(g1,'new');hilite_nodes(10)
```

See Also

add_edge , add_node , delete_arcs , delete_nodes

Name

convex_hull — convex hull of a set of points in the plane

```
[nhull,ind] = convex_hull(xy)
```

Parameters

xy
: 2 x n real matrix

nhull
: integer

ind
: integer row vector

Description

`convex_hull` finds the convex hull of a given set of n points in the plane. `xy` is the 2 x n matrix of the (x,y) coordinates of the given points. `convex_hull` returns in `nhull` the number of the points of the boundary of the convex hull and in `ind` the row vector (of size `nhull`) giving the indices in `xy` of the points of the boundary. The order in `ind` corresponds to consecutive points on the boundary.

Examples

```
ta=[27 27 3 12 11 12 27 26 26 25 25 24 23 23 21 22 21 20 19 18 18];
ta=[ta  16 15 15 14 12 9 10 6 9 17 8 17 10 20 11 23 23 12 18 28];
he=[ 1  2 2  4  5 11 13  1 25 22 24 22 22 19 13 13 14 16 16  9 16];
he=[he  10 10 11 12  2 6  5 5 7  8 7  9  6 11  4 18 13  3 28 17];
g=make_graph('foo',0,28,ta,he);
xx=[46 120 207 286 366 453 543 544 473 387 300 206 136 250 346 408];
g.nodes.graphics.x=[xx 527 443 306 326 196 139 264  55  58  46 118 513];
yy=[36  34  37  40  38  40  35 102 102  98  93  96 167 172 101 179];
g.nodes.graphics.y=[yy 198 252 183 148 172 256 259 258 167 109 104 253];
show_graph(g);
xy=[g.nodes.graphics.x;g.nodes.graphics.y];
[nhull,ind] = convex_hull(xy)
hilite_nodes(ind);
```

Name

cycle_basis — basis of cycle of a simple undirected graph

```
spc = cycle_basis(g)
cycles_list = cycle_basis(g,'list')
```

Parameters

g
: a graph_data_structure.

spc
: a sparse matrix with `edge_number(g)` columns

cycles_list
: a list.

Description

First a spanning tree is found by using min_weight_tree and then used to find all fundamental cycles with respect to this tree. They are returned as a set of cycles, each cycle being represented by a set of edges.

The graph `g` is supposed to be a simple undirected and connected graph (`cycle_basis` does not check that the graph is simple, use `graph_simp` before calling it if necessary).

`spc = cycle_basis(g)` returns these cycles in the sparse matrix `spc`: each row of this matrix corresponds to a cycle.

`cycles_list = cycle_basis(g,'list')` returns these cycles in the list `cycles_list`: each entry of this list is the row vector of the cycle edges index.

Examples

```
//create a directed graph
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 13 15 12 13 9 10 14 11 16 1 17 14 15];
gt=make_graph('foo',1,17,ta,he);
gt.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631
gt.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187
show_graph(gt);

//Make simple and undirected
g=graph_simp(gt);
show_graph(g,'new');

//Compute the cycle basis
cycles_list=cycle_basis(g,'list');

//Display the cycles
for c=cycles_list,hilite_edges(c);xpause(1d6),unhilite_edges(c);end;
```

See Also

min_weight_tree , graph_simp

Name

delete_arcs — deletes all the arcs or edges between a set of nodes

```
g1 = delete_arcs(ij,g)
```

Parameters

ij
: matrix of integers (number of nodes)

g
: graph list

g1
: graph list of the new graph without the arcs or edges defined by ij

Description

If `g` is a directed graph, `delete_arcs` returns the graph `g1` with the arcs defined by matrix `ij` being deleted. `ij` must be a n x 2 matrix of node numbers: the n arcs to be deleted are defined by couples of nodes (`ij(i,1)`, `ij(i,2)`).

If `g` is an undirected graph, the edges corresponding to matrix `ij` are deleted.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 13 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642]
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301
show_graph(g);
ij=[13 10;8 6;5 4;4 2];
gt=delete_arcs(ij,g);
show_graph(gt,'new');
g('directed')=0;
gt=delete_arcs(ij,g);
show_graph(gt,'new');
```

See Also

add_edge , add_node , delete_nodes

Name

delete_edges — deletes all the arcs or edges between a set of nodes

```
g1 = delete_edges(ij,g)
```

Parameters

ij
: matrix of integers (number of nodes)

g
: a graph_data_structure.

g1
: graph data structure of the new graph without the arcs or edges defined by ij

Description

If `g` is a directed graph, `delete_edges` returns the graph `g1` with the arcs defined by matrix `ij` being deleted. `ij` must be a n x 2 matrix of node numbers: the `n` edges to be deleted are defined by couples of nodes (`ij(i,1)`, `ij(i,2)`).

If `g` is an undirected graph, the edges corresponding to matrix `ij` are deleted.

`delete_edges` and `delete_arcs` define the same function

Examples

```
//Create a graph
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 13 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
g.nodes.graphics.display='number';
show_graph(g);

//Select edges to be deleted, edges are given by their (tail, head) couple
ij=[13 10;8 6;5 4;4 2];
hilite_edges(index_from_tail_head(g,ij(:,1),ij(:,2)))
//Delete the arcs
gt=delete_edges(ij,g);

show_graph(gt,'new');

g.directed=0;
gt=delete_edges(ij,g);
show_graph(gt,'new');
```

See Also

add_edge , add_node , delete_nodes

Name

delete_nodes — deletes nodes

```
g1 = delete_nodes(v,g)
```

Parameters

v
: vector of integers, numbers of nodes to be deleted

g
: a graph_data_structure.

g1
: graph data structure of the new graph with deleted nodes

Description

`delete_nodes` returns the graph `g1`, with the nodes given by the vector `v` being deleted.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 13 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 1
g.nodes.graphics.display='number';

show_graph(g);
v=[10 13 4];
gt=delete_nodes(v,g);
show_graph(gt,'new');
```

See Also

add_edge , add_node , delete_arcs

Name

edge_number — number of edges of a graph

```
ma = edge_number(g)
```

Parameters

g
: a graph_data_structure.

m
: integer, number of edges

Description

`edge_number` returns the number `m` of edges of the graph. If the graph is directed, it is the number of arcs. It is always equal to the dimension of `g.edges.tail` and `g.edges.head`.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 13 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
g.nodes.graphics.display='number';
show_graph(g);
edge_number(g)
arc_number(g)
size(ta,2)

g.directed=0;
edge_number(g)
arc_number(g)
```

See Also

arc_number , node_number

Name

edgedatafields — returns the vector of edge data fields names

```
F = edgedatafields(g)
```

Parameters

g
: a graph data structure (see graph_data_structure)

F
: a row vector of strings. Each element is a field name of the edges data data structure.

Description

It is possible to associate data to the edges of a graph. This can be done with the add_edge_data function. the `edgedatafields` function allows to retreive the field names of these data. A given edge data can be referenced by its field name `g.edges.data(field_name)`.

Examples

```
//create a simple graph
ta=[1  1 2 7 8 9 10 10 10 10 11 12 13 13];
he=[2 10 7 8 9 7  7 11 13 13 12 13  9 10];
g=make_graph('simple',1,13,ta,he);
g.nodes.graphics.x=[40,33,29,63,146,233,75,42,114,156,237,260,159];
g.nodes.graphics.y=[7,61,103,142,145,143,43,120,145,18,36,107,107];
show_graph(g,'new')


g=add_edge_data(g,'length',round(10*rand(1,14,'u')));
g=add_edge_data(g,'label','e'+string(1:14));
edgedatafields(g)
g.edges.data.label
```

See Also

graph_data_structure , add_edge_data

Name

edges_data_structure — description of the data structure representing the edges of a graph

Description

A edges data structure is represented by a Scilab `mlist` with type `edges` and 4 fields:

- `tail` row vector. `tail(i)` is the index of the node connected to the tail of the `i`th edge.
- `head` row vector. `head(i)` is the index of the node connected to the head of the `i`th edge.
- `graphics`> A Scilab mlist data structure of type `egraphic` which stores the information relative to edges graphical display (see egraphic_data_structure
- `data` A Scilab `mlist` data structure of type `edgedata`. which stores the data associated with nodes. By defaut this data structure is empty. User can add its own fields using the `add_edge_data` function..

  For a given field the associated data should be a row vector or a matrix. In the matrix case a column is associated to an edge.

Examples

```
//create a simple graph
ta=[1  1 2 7 8 9 10 10 10 10 11 12 13 13];
he=[2 10 7 8 9 7  7 11 13 13 12 13  9 10];
g=make_graph('simple',1,13,ta,he);
g.nodes.graphics.x=[40,33,29,63,146,233,75,42,114,156,237,260,159];
g.nodes.graphics.y=[7,61,103,142,145,143,43,120,145,18,36,107,107];
show_graph(g,'new')


g=add_edge_data(g,'length',round(10*rand(1,14,'u')));
g=add_edge_data(g,'label','e'+string(1:14));
edgedatafields(g)
g.edges.data
g.edges.data.label
g.edges.data(1:3)
g.edges.graphics.display='label';
show_graph(g)
g.edges.graphics.display='length';
show_graph(g)
```

See Also

graph_data_structure , add_edge , delete_arcs , edgedatafields , add_edge_data

Name

edit_graph — graph and network graphical editor

```
num=edit_graph()
num=edit_graph(file_name [,zoom [,wsize]])
num=edit_graph(G [,zoom [,wsize]])
```

Parameters

file_name
: character string. The path of a "graph" file

G
: a graph_data_structure.

zoom
: real positive scalar. The zoom factor, its default value is 1.

wsize
: real positive vector `[width height]`. The initial window dimensions, its default value is `[600,400]`.

num
: integer scalar. The associated window number window dimension.

Description

This function starts a network graphical editor. Each time `edit_graph` is executed, a new editor window is created. The editor capabilities and menus are described in edit_graph_menus

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187
edit_graph(g)
edit_graph(g,1,[800 600])
```

See Also

netclose , show_graph , netwindow , edit_graph_menus

Name

edit_graph_menus — edit_graph menus description

Description

The edit_graph editor supports the following menus:

File/New:
: This menu erases the currently edited network.

View/Zoom:
: This menu allows to set the scale factor for the display

View/Replot:
: This menu erases and re-displays the network.

  This edition mode can also be entered pressing the "r" key while locator is in the edition window.

View/Find Node:
: This menu asks for a node number or a node name. If this node exists the view is modified such as the node appears in the middle of the graphic window.

View/Find Arc:
: This menu asks for an arc number or aa arc name. If this arc exists the view is modified such as the arc appears in the middle of the graphic window.

File/SaveAs:
: This menu allows to save the graph under a specified name

File/Save:
: This menu allows to save the graph under its current name

  This edition mode can also be entered pressing the "s" key while locator is in the edition window.

File/Load:
: This menu allows to load a "graph" file.

File/Export:
: This menu allows to export the view of the graph in a regular graphic window.

View/Options:
: This menu allows to set global graph properties This edition mode can also be entered pressing the "o" key while locator is in the edition window.

Graph/Settings:
: This menu allows to set if graph is directed or not. It also sets default values for node diameter, node border width, edge width, and font size.

Graph/Add Node Data Field:
: This menu allows to add a new data field in the nodes data structure.

Graph/Add Edge Data Field:
: This menu allows to add a new data field in the edges data structure.

Information:
: This menu outputs graph information, like number of nodes, number of arcs, nodes properties ans arc properties.

File/Quit:
: Use this menu to exit out of the editor.

Edit/NewNode:
This menu set the current edition mode to "node addition". To add a node just click on its desired position. This edition mode can also be entered pressing the "n" key while locator is in the edition window (Under windows, the edition window should also have the focus). It is the current editing mode when edit_graph is started. This mode remains active until an other edition mode is selected.

This edition mode can also be entered pressing the "n" key while locator is in the edition window.

Edit/NewArc:
This menu set the current edition mode to "edge addition". To add an edge between two nodes just left click on the initial node and on the final node (click on right button cancels current edge addition).

This edition mode can also be entered pressing the "a" key while locator is in the edition window (Under windows, the edition window should also have the focus). This mode remains active until an other edition mode is selected.

This edition mode can also be entered pressing the "a" key while locator is in the edition window.

Edit/Move Node:
This menu set the current edition mode to "move node". To move a node just left click on the selected node move the mouse up to the final position and left click (click on right button cancels move).

This edition mode can also be entered pressing the "m" key while locator is in the edition window.

This mode remains active until an other edition mode is selected.

Edit/Move Region:
This menu set the current edition mode to "move region mode". To move a rectangular region just click left on a rectangle corner, drag rectangle to the desired area, click left to validate the area (click on right button cancels move) then move the rectangle to the desired position and click to validate.

Edit/Copy Region To ClipBoard:
This menu set the current edition mode to "copy region to clipboard mode". To copy a rectangular region into the edit_graph clipboard just click left on a rectangle corner, drag rectangle to the desired area, click left to validate the area (click on right button cancels copy) .

The edit_graph clipboard is shared by all edit_graph editors.

This mode remains active until an other edition mode is selected.

Edit/Paste:
This menu set the current edition mode to "paste from clipboard mode" to clipboard mode". To paste clipboard into the edit_graph editor click left to select the editor window, drag rectangle to the desired area, click left to validate the position (click on right button cancels copy) .

This edition mode can also be entered pressing the "v" key while locator is in the edition window.

This mode remains active until an other edition mode is selected.

Edit/Delete:
This menu set the current edition mode to "delete object mode". To delete a node or an edge just left click on the selected object.

This edition mode can also be entered pressing the "d" key while locator is in the edition window.

This mode remains active until an other edition mode is selected.

Edit/Delete Region
: This menu set the current edition mode to "delete region mode". To delete a rectangular region just click left on a rectangle corner, drag rectangle to the desired area, click left to validate the area (click on right button cancels selection).

  Note that the deleted region is not send to the clipboard

  This mode remains active until an other edition mode is selected. This edition mode can also be entered pressing the "x" key while locator is in the edition window.

Edit/Properties:
: This menu set the current edition mode to "set object properties". To set a node or an edge properties, just left click on the selected object; a popup dialog appears.

  This mode remains active until an other edition mode is selected.

  This edition mode can also be entered pressing the "p" key while locator is in the edition window.

Edit/Default names:
: This menu automatically sets the node and egde names properties to node and egde internal number.

Edit/Undo:
: This menu set the current edition mode to "undo previous edition". It can be used recursively.

  This edition mode can also be entered pressing the "u" key while locator is in the edition window.

Note that Under windows, the shortcuts are taken into accound only if the edition window have the focus.

See Also

edit_graph

Name

egraphic_data_structure — data structure representing the graphic properties used for edges graphical display

Description

A data structure represented by a Scilab `mlist` with type `egraphic` and 8 fields:

- `display` a string. Gives the information that is displayed with the edge. The possible values are `'nothing'`, `'number'`, `'name'` or any edge data field name as given by the `edgedatafields` function. Of course if `display` is set to `'nothing'` no information is displayed.
- `defaults` A Scilab tlist data structure of type `edgedefs`. Contains the default values for `'width'`, `'foreground'`, `'font'`, `'profile_index'` properties.
- `profiles` A Scilab list which stores the different edge profiles used for drawing the egdes.
- `name` A row vector of strings. The name associated with each edge.
- `width`a row vector. The thickness of the polyline used to draw each edge. A zero value stands for the default value.
- `foreground`a row vector. The color index (relative to current colormap) of the polyline used to draw each edge. A zero value stands for the default value.
- `font`a matrix with 3 rows: `font(1,i)` is the font size, `font(2,i)` is the font style,`font(2,i)` is the font color used to draw information on the i th edge..A zero value of either entry stands for the corresponding default value.
- `profile_index`a row vector. The index relative to the `profiles` list of the profile to use to draw the edge polyline.

Examples

```
//create a simple graph
ta=[1  1 2 7 8 9 10 10 10 10 11 12 13 13 4];
he=[2 10 7 8 9 7  7 11 13 13 12 13  9 10 4];
g=make_graph('simple',1,13,ta,he);
g.nodes.graphics.x=[40,33,29,63,146,233,75,42,114,156,237,260,159];
g.nodes.graphics.y=[7,61,103,142,145,143,43,120,145,18,36,107,107];
show_graph(g,'new')

g.edges.graphics.defaults.width=2;
g.edges.graphics.defaults.foreground=color('red');
show_graph(g)

g.edges.graphics.width(1:5)=1;
g.edges.graphics.foreground([10 12])=color('blue');
show_graph(g)

g.edges.graphics.display='number';
show_graph(g)

g.edges.graphics
```

See Also

edges_data_structure

Name

find_path — finds a path between two nodes

```
p = find_path(i,j,g)
```

Parameters

i
: integer, number of start node

j
: integer, number of end node

g
: a graph_data_structure.

p
: row vector of integer numbers of the arcs of the path if it exists

Description

`find_path` returns a path `p` from node number `i` to node number `j` if one exists, and the empty vector `[]` otherwise.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
g.nodes.graphics.display='number';
g.edges.graphics.display='number';
show_graph(g);hilite_nodes([1 14])
p=find_path(1,14,g)
g.edges.graphics.foreground(p)=color('red');
show_graph(g);
```

See Also

nodes_2_path , shortest_path

Name

gen_net — interactive or random generation of a network

```
g = gen_net(name,directed,v)
g = gen_net()
```

Parameters

name
: string, the name of the graph

directed
: integer, 0 (undirected graph) or 1 (directed graph)

v
: row vector with 12 values for defining the network

g
: a graph_data_structure.

Description

`gen_net` generates a network `g`. The arguments are the name of the graph, a flag equal to 0 (undirected graph) or to 1 (directed graph) and a vector describing the network (see below).

If no argument are given, a dialog box for the definition of all the arguments is opened.

`v` must be a row vector with 12 values. The meaning of the values are:

Seed for random: used for initialization of random generation

Number of nodes

Number of sources

Number of sinks

Minimum cost

Maximum cost

Input supply

Output supply

Minimum capacity

Maximum capacity

Percentage of edges with costs: between 0 and 100

Percentage of edges with capacities: between 0 and 100

The cost of edges without cost are put to minimum cost. The maximum capacity of edges without capacity are put to maximum upply

The result is a network `g` built on a planar connected graph, by using a triangulation method. Moreover, computations are made in order to have a coherent network. Values of costs and maximum capacities are put on the edges. Minimum capacities are reduced to 0.

Examples

```
v=[1,10,2,1,0,10,100,100,0,100,50,50];
g=gen_net('foo',1,v);
show_graph(g)
// generating using dialogs
g=gen_net();
show_graph(g)
```

See Also

mesh2d

Name

girth — girth of a directed graph

```
d = girth(g)
```

Parameters

g
: a graph_data_structure.

d
: integer

Description

`girth` computes the length (number of arcs) of the shortest cycle in an unweighted directed graph `g`.

Examples

```
ta=[1 6 2 4 7 5 6 8 4 3 5 1];
he=[2 1 3 6 4 8 8 7 2 7 3 5];
g=make_graph('foo',1,8,ta,he);
g.nodes.graphics.x=[285  284  335  160  405  189  118  45];
g.nodes.graphics.y=[266  179   83  176  368  252  64  309];
show_graph(g);
d=girth(g)
```

Name

glist — Scilab-4.x graph list creation

```
g = glist(a1, ... ,a34)
```

Description

This is an obsolete function, replaced by make_graph

`glist(a1,....a34)` is a shortcut to to

```
      tlist(['graph','name','directed','node_number','tail','head',..
             'node_name','node_type','node_x','node_y','node_color',..
             'node_diam','node_border','node_font_size','node_demand',..
             'edge_name','edge_color','edge_width','edge_hi_width',..
             'edge_font_size','edge_length','edge_cost',..
             'edge_min_cap','edge_max_cap','edge_q_weight','edge_q_orig',..
             'edge_weight','default_node_diam','default_node_border',..
             'default_edge_width','default_edge_hi_width',..
             'default_font_size','node_label','edge_label'],a1, ... ,a34)
```

It is a low level function to create graph lists, mainly used by programmers. No checking is done. For standard creation of graph lists, use `make_graph`.

See Also

check_graph , make_graph

Name

graph-list — description of graph list (obsolete)

Description

A graph in Scilab-4.x was represented by a Scilab typed list called "graph list". This data structure as been replaced by a more structured and flexible one (see graph_data_structure). The function update_graph can be used to translate old graph data structure to the new one. The load_graph function automatically apply the transformation if the file contains an old data structure.

Name

graph_2_mat — node-arc or node-node incidence matrix of a graph

```
a = graph_2_mat(g,mat)
```

Parameters

g
: a graph_data_structure.

mat
: optional string, 'node-arc' or 'node-node' matrix

a
: sparse node-arc or node-node incidence matrix

Description

`graph_2_mat` computes the node-arc or the node-node incidence matrix corresponding to the graph `g`.

If the optional argument `mat` is omitted or is the string `'node-arc'`, the node-arc matrix is computed. If `mat` is the string `'node-node'`, the node-node matrix is computed.

If `n` is the number of nodes of the graph and `m` is the number of edges of the graph, the node-arc matrix is a Scilab sparse matrix of size `(n,m)`.

It is defined as follows. If the graph is directed:

`a(i,j) = +1` if node `i` is the tail of arc `j`

`a(i,j) = -1` if node `i` is the head of arc `j`

If the graph is undirected:

`a(i,j) = 1` if node `i` is the tail or the head of arc `j`

If `n` is the number of nodes of the graph, the node-node matrix is a Scilab sparse matrix of size `(n,n)`.

It is defined as follows:

`a(i,j) = 1` if there is an arc from node `i` to node `j`

Examples

```
ta = [10,3,6,2,3,7,6,9,5,10,8,2,5,8,4,9,1,8,9,4,7]
he = [3,6,10,6,2,6,9,7,3,5,2,5,8,4,9,8,4,1,2,7,10]
g=make_graph('foo',1,10,ta,he);
g.nodes.graphics.x = [398,333,212,312,132,208,46,445,301,69];
g.nodes.graphics.y = [54,217,179,12,245,133,95,283,92,170];
g.nodes.graphics.display='number';
show_graph(g);

a=graph_2_mat(g,'node-node');
a=[['' string(1:10)];[string(1:10)' string(full(a))]]
```

```
a=graph_2_mat(g);
string(full(a))'
```

See Also

mat_2_graph

Name

graph_center — center of a graph

```
[no,rad] = graph_center(g)
```

Parameters

g
: a graph_data_structure.

no
: integer

rad
: integer

Description

`graph_center` computes a center of the graph `g` i.e. the node for which the largest of the shortest paths to all the other nodes is minimum. The lengths of the arcs are supposed to be integer (and the default value is 1). The output is the value `rad` of the length of the radius and `no` which is the node number of the center of the graph.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 11 12 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 15 12 13  9 14 11 16 1 17 14 15];
g=make_graph('foo',0,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
show_graph(g);
[no,rad] = graph_center(g)
hilite_nodes(no);
r1=0;
for k=1:16
 [p,lp] = shortest_path(no,k,g,'arc');
  if lp>r1 then r1=lp;path=p;end
end
r1
hilite_edges(path)
```

See Also

graph_diameter

Bibliography

Harary, F. Graph Theory. Reading, MA: Addison-Wesley, p. 35, 1994.

Name

graph_complement — complement of a graph

```
g1 = graph_complement(g,[gmax])
```

Parameters

g
: a graph_data_structure.

gmax
: :a optional graph data structure.

g1
: graph data structure of the new graph

Description

The complement of a graph g is the graph g1 with the same node set but whose edge set consists of the edges not present in g (i.e., the complement of the edge set of g with respect to all possible edges on the vertex set of G).

`graph_complement` returns the undirected graph `g1` which is the complement of the graph `g` with respect to the corresponding complete graph. When `gmax` is given, the complement is made with respect to `gmax`. `g` and `gmax` are supposed to be simple graphs (use `graph_simp` before calling `graph_complement` if necessary) and to have the same number of nodes.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 11 12 13 13 13 14 15 17 17 16 16];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 15 12 13 9 10 14 11 16 14 15 1 17];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
show_graph(g);
g1=graph_complement(g);
show_graph(g1,'new');
g=graph_complement(g1);
show_graph(g,'new');
```

See Also

graph_sum , graph_simp

Name

graph_data_structure — description of the main graph data structure

Description

A graph in Scilab is represented by a Scilab typed list with type `graph` and 5 fields:

- `version`a simple string which contains the graph data structure version identifier.
- `name` a simple string which contains the graph name.
- `directed`a number with possible values 0 and 1. The value 1 means that the graph edges are oriented.
- `nodes` A Scilab mlist data structure, which stores the information relative to nodes (see nodes_data_structure).
- `edges` A Scilab mlist data structure, which stores the information relative to edges (see edges_data_structure).

Examples

```
//create a simple graph
ta=[1  1 2 7 8 9 10 10 10 10 11 12 13 13];
he=[2 10 7 8 9 7  7 11 13 13 12 13  9 10];
g=make_graph('simple',1,13,ta,he);
g.nodes.graphics.x=[40,33,29,63,146,233,75,42,114,156,237,260,159];
g.nodes.graphics.y=[7,61,103,142,145,143,43,120,145,18,36,107,107];
show_graph(g,'new')

g
g.name
g.directed
g.nodes(1:3)
g.edges(1:5)
hilite_nodes(1:3)
hilite_edges(1:5)
```

See Also

edit_graph , make_graph , show_graph , check_graph

Name

graph_diameter — diameter of a graph

```
[d,p] = graph_diameter(g)
```

Parameters

g
: a graph_data_structure.

d
: integer

p
: integer row vector

Description

`graph_diameter` computes the diameter of the graph `g` i.e. the largest shortest path between two nodes. The length of the arcs are supposed to be integer (and the default value is 1). The output is the value `d` of the length of the diameter and `p` is the corresponding path.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 11 12 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 15 12 13  9 14 11 16 1 17 14 15];
g=make_graph('foo',0,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
show_graph(g);
[d,p] = graph_diameter(g)
hilite_edges(p);
```

See Also

graph_center

Bibliography

Harary, F. Graph Theory. Reading, MA: Addison-Wesley, p. 35, 1994.

Name

graph_power — kth power of a directed 1-graph

```
g1 = graph_power(g,k)
```

Parameters

g
: a graph_data_structure defining a directed graph.

k
: integer

g1
: graph data structure of the new graph

Description

`graph_power` computes the directed graph `g1` which is the kth power of directed 1-graph `g`. There is an arc between two nodes in `g1` if there exists a path between these nodes of length at most k in `g`. `graph_power(g,1)` is graph `g`.

If such a graph does not exist, an empty vector is returned.

Examples

```
ta=[1 1 2 4 4 5 6 7 2 3 5 1];
he=[2 6 3 6 7 8 8 8 4 7 3 5];
g=make_graph('foo',1,8,ta,he);
g.nodes.graphics.x=[285  284  335  160  405  189  118  45];
g.nodes.graphics.y=[266  179   83  176  368  252  64  309];
show_graph(g);

g2=graph_power(g,2);
show_graph(g2,'new');
```

Name

graph_simp — converts a graph to a simple undirected graph

```
g1 = graph_simp(g)
```

Parameters

g
: a graph_data_structure.

g1
: graph data structure of the new graph

Description

`graph_simp` returns the simple undirected graph `g1` corresponding to multigraph `g`. It deletes loops in `g`, replaces directed edges with undirected edges and replaces multiple edges with single edges.A simple graph is also called a strict graph.

Examples

```
ta=[1 1  1 2 2 2 3 4 4 4 5 5 6 7 7 8 8 9 9 10 10 10 10 10 11 12 12 13 13 13 14 1
he=[1 2 10 3 5 7 4 2 9 9 4 6 6 8 2 6 9 7 4  7 11 13 13 15 12 11 13 9 10 14 11 1
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163  63  98 164 162 273 235 267 384 504 493 409 573 601
g.nodes.graphics.y=[ 59 133 223 311 227 299 221 288 384 141 209 299 398 383 187
show_graph(g);
g1=graph_simp(g);
show_graph(g1,'new');
```

Bibliography

Bronshtein, I. N. and Semendyayev, K. A. Handbook of Mathematics, 4th ed. New York: Springer-Verlag, 2004.

Gibbons, A. Algorithmic Graph Theory. Cambridge, England: Cambridge University Press, 1985.

Harary, F. "The Number of Linear, Directed, Rooted, and Connected Graphs." Trans. Amer. Math. Soc. 78, 445-463, 1955.

Harary, F. "Enumeration of Graphs." In Graph Theory. Reading, MA: Addison-Wesley, pp. 185-187, 1994.

Steinbach, P. Field Guide to Simple Graphs. Albuquerque, NM: Design Lab, 1990.

Tutte, W. T. Graph Theory as I Have Known It. Oxford, England: Oxford University Press, 1998.

West, D. B. Introduction to Graph Theory, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2000.

Name

graph_sum — sum of two graphs

```
g2 = graph_sum(g,g1)
```

Parameters

g
: a graph_data_structure.

g1
: an other graph data structure, with same nodes

g2
: graph data structure of the new graph sum.

Description

`graph_sum` creates a graph `g2` with an adjacency matrix equal to the sum of the adjacency matrices of the two graphs `g` and `g1`. `g` and `g1` are supposed to be simple graphs (use `graph_simp` before calling `graph_complement` if necessary) and to have the same number of nodes.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 75
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 1
g.nodes.graphics.name=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N'
g.nodes.graphics.display='name';
show_graph(g);

ta1=[2 3 4 5 11 12 1];
he1=[10 5 6 7 15 17 7];
g1=make_graph('foo',1,17,ta1,he1);
g1.nodes.graphics.x=g.nodes.graphics.x
g1.nodes.graphics.y=g.nodes.graphics.y
g1.nodes.graphics.name=g.nodes.graphics.name
g1.nodes.graphics.display='name';

show_graph(g1,'new');
g2=graph_sum(g,g1);
show_graph(g2,'new');
hilite_edges(index_from_tail_head(g2,ta,he))

//chech if g and g1 adjacency matrices sum is equal to g1 adjacency matrix
a=graph_2_mat(g,'node-node');
a1=graph_2_mat(g1,'node-node');
a2=graph_2_mat(g2,'node-node');
and(a+a1==a2)
```

See Also

graph_complement , graph_union

Name

graph_union — union of two graphs

```
g2 = graph_union(g,g1 [,opt])
```

Parameters

g
: a graph_data_structure.

g1
: a graph data structure.

opt
: a boolean, with `%t` as default value.

g2
: graph data structure of the new graph

Description

- `graph_union(g,g1)` creates a new graph `g2`. The node set of `g2` is the union (in the usual sense) of the node sets of `g` and `g1`. If node names are given in both graph, nodes with equal names in `g1` and `g2` are considered as common nodes.

  `g2` has an edge for each edge of `g` and an edge for each edge of `g1`. The edges of `g` and `g1` having the same endpoints are kept and in this case `g2` has multiple edges.

- `graph_union(g,g1,%f)` creates a new graph `g2`. The node set of `g2` is the union (in the usual sense) of the node sets of t`g` and `g1`. In this case the function does not looks for common nodes.

  `g2` has an edge for each edge of `g` and an edge for each edge of `g1`. The edges of `g` and `g1` having the same endpoints are kept and in this case `g2` has multiple edges.

Examples

```
ta1=[1,2,3,4,4,4,4,5,6,7,7];he1=[2,3,1,1,5,7,7,6,7,3,4];
g1=make_graph('foo1',1,7,ta1,he1);
g1.nodes.graphics.x= [273,271,339,384,504,513,439];
g1.nodes.graphics.y= [221,324,432,141,209,319,428];
g1.nodes.graphics.display= 'name';
g1.nodes.graphics.name= ['A' 'B' 'C' 'D' 'E' 'F' 'G'];
w1=show_graph(g1);

ta2=[1,1,2,2,3,5,6,6,7,8];he2=[2,8,3,5,4,6,4,7,5,5];
g2=make_graph('foo2',1,8,ta2,he2);
g2.nodes.graphics.x= [233,113,114,114,223,221,289,334];
g2.nodes.graphics.y= [59, 133,227,319,221,324,432,141];
g2.nodes.graphics.name= ['H' 'I' 'J' 'K' 'A' 'B' 'C' 'D'];
g2.nodes.graphics.display= 'name';
w2=show_graph(g2,'new');


g=graph_union(g1,g2);
show_graph(g,'new');
```

```
g=graph_union(g1,g2,%f);
show_graph(g,'new');
```

See Also

supernode , subgraph

Name

hamilton — hamiltonian circuit of a graph

```
cir = hamilton(g)
```

Parameters

g
: a graph_data_structure.

cir
: integer row vector

Description

A Hamiltonian circuit, is a graph circuit through a graph that visits each node exactly once.

`hamilton` finds an hamiltonian circuit (if it exists) of the directed graph `g`.

Examples

```
ta=[2  1 3 2 2 4 4 5 6 7 8 8 9 10 10 10 10 11 12 13 13 14 15 16 16 17 17];
he=[1 10 2 5 7 3 2 4 5 8 6 9 7 7 11 13 15 12 13  9 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
show_graph(g);
cir=hamilton(g)
hilite_edges(cir);

ta=[1,1,2,5,4,3,14,13,12,11,10,9,8,7,6 ,15,17,16,18,20,19,13,6,19];
he=[3,2,5,4,3,14,15,12,11,10,9 ,8,7,6,15,17,16,18,20,19,17,14,1,13];
g=make_graph('foo',1,20,ta,he);
g.nodes.graphics.x=[-30,-182,161,64,-116,-25,-79,-128,-97,-94,-19,47,66,98,24,-
g.nodes.graphics.y=[198,94,90,-107,-96,144,103,78,13,-63,-48,-76,23,77,103,76,7
show_graph(g,'new');
cir=hamilton(g)
hilite_edges(cir);
```

Name

hilite_edges — highlights a set of edges
unhilite_edges — unhighlights a set of edges

```
hilite_edges(p)
unhilite_edges(p)
```

Parameters

p
: row vector of edge numbers

Description

`hilite_edges` highlights the set of edges `p` of the displayed graph in the current edit_graph window.

`unhilite_edges` un-highlights the set of edges `p` of the displayed graph in the current edit_graph window. If the edges are not hilited nothing is done.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
show_graph(g);
for i=1:edge_number(g), hilite_edges(i);xpause(3d5);unhilite_edges(i), end;
hilite_edges(1:3:edge_number(g))
```

See Also

hilite_nodes , show_nodes , show_arcs , netwindow , netwindows

Name

hilite_nodes — highlights a set of nodes
unhilite_nodes — unhighlights a set of nodes

```
hilite_nodes(p)
unhilite_nodes(p)
```

Parameters

p
: row vector of node numbers

Description

`hilite_nodes` highlights the set of nodes `p` of the displayed graph in the current edit_graph window.

`unhilite_nodes` un-highlights the set of nodes `p` of the displayed graph in the current edit_graph window. If the nodes are not hilited nothing is done.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
show_graph(g);
for i=2:3:node_number(g), hilite_nodes(i);xpause(3d5);unhilite_nodes(i), end;
hilite_nodes(1:3:node_number(g))
```

See Also

show_nodes , show_arcs , netwindow , netwindows

Name

index_from_tail_head — Computes the index of edges given by (tail,head) pairs

```
i = index_from_tail_head(g,t,h)
```

Parameters

g
: a graph_data_structure.

t
: a vector: edges tail node numbers.

h
: a vector: edges head node numbers.

i
: a row vector of edge numbers

Description

`i = index_from_tail_head(g,tail,head)` computes the index of edges given by `(tail(k),head(k))` pairs relative to the graph `g`.

Examples

```
ta=[27 27 3 12 11 12 27 26 26 25 25 24 23 23 21 22 21 20 19 18 18];
ta=[ta  16 15 15 14 12 9 10 6 9 17 8 17 10 20 11 23 23 12 18 28 14];
he=[ 1  2 2  4  5 11 13  1 25 22 24 22 22 19 13 13 14 16 16  9 16];
he=[he  10 10 11 12  2 6  5 5 7  8 7  9  6 11  4 18 13  3 28 17 12];
n=28;
g=make_graph('foo',1,n,ta,he);
xx=[46 120 207 286 366 453 543 544 473 387 300 206 136 250 346 408];
g.nodes.graphics.x=[xx 527 443 306 326 196 139 264  55  58  46 118 513];
yy=[36  34  37  40  38  40  35 102 102  98  93  96 167 172 101 179];
g.nodes.graphics.y=[yy 198 252 183 148 172 256 259 258 167 109 104 253];
g.nodes.graphics.display='number';
show_graph(g);

i = index_from_tail_head(g,[22 14 17],[13 12 9])
hilite_edges(i)
```

See Also

show_edges

Authors

Serge Steer
INRIA

Name

is_connex — connectivity test

```
res = is_connex(g)
```

Parameters

g
: a graph_data_structure.

res
: integer, result of the test

Description

`is_connex` returns 1 if the graph `g` is connected and 0 otherwise.

Examples

```
g=make_graph('foo',1,3,[1,2,3,1],[2,3,1,3]);
g.nodes.graphics.x=[26,85,-55];
g.nodes.graphics.y=[92,-18,-25];
show_graph(g)
is_connex(g)

g1=add_node(g,[26,-90]);
show_graph(g1)
is_connex(g1)
```

See Also

con_nodes , strong_connex

Name

knapsack — solves a 0-1 multiple knapsack problem

```
[earn,ind] = knapsack(profit,weight,capa,[bck])
```

Parameters

profit
: integer row vector

weight
: integer row vector

capa
: integer row vector

bck
: integer

earn
: integer

ind
: integer row vector

Description

The 0-1 multiple knapsack problem with n (n >= 2) items and m knapsacks (m >= 1) is defined as follow:

Maximize the global profit `E=profit*sum(X,1)` under the constraints:

```
X*weight <= capa
```

```
sum(X,1) <= 1 ; i=1,...,n
```

```
X(j,i)= 0 or 1
```

Where

profit
: is the vector of the "profits" of the n items. The entries must be positive integers.

weight
: is the vector of the corresponding "weights". The entries must be positive integers.

capa
: is the vector of the (integer) capacities of the m knapsacks.The entries must be positive integers.

X
: is a `m` by `n` matrix.

  est une matrice `m` par `n` à valeurs dans {0,1}.

`[earn,ind] = knapsack(profit,weight,capa)` solves the problem. It returns in

earn
: the value of the criterium `E` for the "optimal" solution if it has been found. In case of error, `earn` is assigned to a negative value:

-3
means that a knapsack cannot contain any item.

-4
means that an item cannot fit into any knapsack.

-5
means that a knapsack contains all the items.

ind
the integer vector of the knapsack number where item `i` is inserted and this value is 0 if the item i is not in the optimal solution. The matrix `X` can be derived from `ind` by

```
items=1:n;
items(ind==0)==[];
ind(ind==0)=[];
X=sparse([ind;items]',ones(n,1),[m,n])
```

bck
is an optional integer: the maximum number of backtrackings to be performed if heuristic solution is required. If the exact solution is required `bck` must be omitted or assigned to a negative value.

Examples

```
weight=ones(1,15).*.[1:4];
profit=ones(1,60);
capa=[15 40 30 60];
[earn,ind]=knapsack(profit,weight,capa)

items=1:60;
items(ind==0)=[];
ind(ind==0)=[];
X=full(sparse([ind;items]',ones(ind),[4,60])) //one row per sacks
X*weight' //sack weights
x=sum(X,1);
and(x<=1) //contraints check
profit*x'==earn
```

See Also

qassign

Bibliography

Coppersmith, D. "Knapsack Used in Factoring." §4.6 in Open Problems in Communication and Computation (Ed. T. M. Cover and B. Gopinath). New York: Springer-Verlag, pp. 117-119, 1987.

Honsberger, R. Mathematical Gems III. Washington, DC: Math. Assoc. Amer., pp. 163-166, 1985.

Name

line_graph — graph with nodes corresponding to edges

```
g1 = line_graph(g)
```

Parameters

g
: a graph_data_structure.

g1
: graph data structure of the new graph

Description

`line_graph` returns the graph `g1` with the nodes corresponding to the edges of the graph `g`. `g1` is defined in the following way: - its nodes correspond to the edges of `g` - 2 nodes of the new graph are adjacent if and only if the corresponding edges of the graph `g` are adjacent.

The coordinates of the nodes of `g1` are given by the middle points of the corresponding edges of `g`.

Examples

```
ta=[1 1 2 4 4 5 6 7 2 3 5 1];
he=[2 6 3 6 7 8 8 8 4 7 3 5];
g=make_graph('foo',0,8,ta,he);
g.nodes.graphics.x=[281  284  360  185  405  182  118  45];
g.nodes.graphics.y=[262  179  130  154  368  248  64  309];
show_graph(g);
g1=line_graph(g);show_graph(g,'new');

g1.nodes.graphics.colors(2,:)=color('red')
g1.edges.graphics.foreground(:)=color('red')

show_graph(graph_union(g,g1,%f),'new')
```

See Also

arc_graph

Name

load_graph — loads a graph from a file

```
[g,modified,mgs] = load_graph(name)
```

Parameters

name
: string, the path of the file to load

g
: a graph_data_structure.

modified
: a boolean, if true it indicates that the graph list has been updated to the current version

msg
: a string. if this argument is required the `load_graph` function returns on errors instead of calling the error manager. In such a case msg contains the error message. If no error occurs `msg` is an empty matrix.

Description

`load_graph` loads a graph from a file. This file may be either an ascii file (scilab4.x versions) or a binary file compatible with the `load` and `save` functions.

Examples

```
g=load_graph(metanet_module_path()+'/demos/mesh100.graph');
show_graph(g);
g=load_graph(metanet_module_path()+'/demos/colored.graph');
show_graph(g,'new');
```

See Also

save_graph , graph_data_structure , save , load

Name

make_graph — makes a graph list

```
g = make_graph(name,directed,n,tail,head)
```

Parameters

name
: string, the name of the graph

directed
: integer, 0 (undirected graph) or 1 (directed graph)

n
: integer, the number of nodes of the graph

tail
: row vector of the numbers of the tail nodes of the graph (its size is the number of edges of the graph)

head
: row vector of the numbers of the head nodes of the graph (its size is the number of edges of the graph)

g
: a graph_data_structure.

Description

`make_graph` makes a graph list according to its arguments which are respectively the name of the graph, a flag for directed or undirected, the number of nodes and the row vectors tail and head. These are the minimal data needed for a graph.

If `n` is a positive number, graph `g` has `n` nodes; this number must be greater than or equal to `max(max(tail),max(head))`. If it is greater than this number,graph `g` has isolated nodes. The nodes names are taken as the nodes numbers.

If `n` is equal to 0, graph `g` has no isolated node and the number of nodes is computed from `tail` and `head`. The nodes names are taken from the numbers in `tail` and `head`.

Examples

```
// creating a directed graph with 3 nodes and 4 arcs.
g=make_graph('foo',1,3,[1,2,3,1],[2,3,1,3]);

// creating a directed graph with 13 nodes and 14 arcs.
ta=[1  1 2 7 8 9 10 10 10 10 11 12 13 13];
he=[2 10 7 8 9 7  7 11 13 13 12 13  9 10];
g=make_graph('foo',1,13,ta,he);
g.nodes.graphics.x=[40,33,29,63,146,233,75,42,114,156,237,260,159]
g.nodes.graphics.y=[7,61,103,142,145,143,43,120,145,18,36,107,107]
show_graph(g)

// creating same graph without isolated node and 14 arcs.
g=make_graph('foo',1,0,ta,he);
```

```
g.nodes.graphics.x=[40,33,75,42,114,156,237,260,159];
g.nodes.graphics.y=[7,61,43,120,145,18,36,107,107];
show_graph(g,'new')
```

See Also

graph_data_structure

Name

mat_2_graph — graph from node-arc or node-node incidence matrix

```
g = mat_2_graph(a,directed,[mat])
```

Parameters

a
: sparse node-arc or node-node incidence matrix

directed
: integer, 0 (undirected graph) or 1 (directed graph)

mat
: optional string, `'node-arc'` or `'node-node'` matrix

g
: a graph_data_structure.

Description

`mat_2_graph` computes the graph `g` corresponding to the node-arc or the node-node incidence matrix `a`. Note that a checking is made to insure that `a` is a sparse node-arc or node-node incidence matrix of a directed (`directed = 1`) or undirected (`directed = 0`) graph. If the optional argument `mat` is omitted or is the string `'node-arc'`, a must be a node-arc matrix. If `mat` is the string `'node-node'`, a must be a node-node matrix.

Examples

```
// creating a directed graph with 13 nodes and 14 arcs.
ta=[1  1 2 7 8 9 10 10 10 11 12 13 13];
he=[2 10 7 8 9 7  7 11 13 12 13  9 10];
g=make_graph('foo',1,13,ta,he);
g.nodes.graphics.x=[40,33,29,63,146,233,75,42,114,156,237,260,159]
g.nodes.graphics.y=[7,61,103,142,145,143,43,120,145,18,36,107,107]
show_graph(g)

a=graph_2_mat(g);
g1=mat_2_graph(a,1);
g1.nodes.graphics.x=g.nodes.graphics.x; g1.nodes.graphics.y=g.nodes.graphics.y;
show_graph(g1,'new');

a=graph_2_mat(g,'node-node');
g1=mat_2_graph(a,1,'node-node');
g1.nodes.graphics.x=g.nodes.graphics.x; g1.nodes.graphics.y=g.nodes.graphics.y;
show_graph(g1,'new');
```

See Also

adj_lists, chain_struct, graph_2_mat

Name

max_cap_path — maximum capacity path

```
[p,cap] = max_cap_path(i,j,g)
```

Parameters

i,j
: integers, node numbers

g
: a graph_data_structure.

p
: row vector of integer numbers of the arcs of the path if it exists

cap
: value of the capacity of the path

Description

`max_cap_path` returns the path with maximum capacity from node `i` to node `j` for the graph `g` if it exists and returns the empty vector `[]` otherwise.

The capacities of the edges are given by the field `max_cap` of the edges sub field of the graph data structure. If its value is not given (empty vector `[]`), `max_cap_path` returns the empty vector `[]`. The capacities must be strictly positive, i.e negative capacities are considered as equal to 0 (no capacity at all).

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[142,82,32,29,82,82,137,136,170,192,252,257,220,312,316,379,
g.nodes.graphics.y=[30,67,112,159,114,160,111,162,216,71,105,160,214,222,94,76,
g = add_edge_data(g,'max_cap',int(rand(1,edge_number(g))*16)+5)
g.edges.graphics.display='max_cap';
g.nodes.graphics.display='number';
show_graph(g);

hilite_nodes([1 14])
[p,cap]=max_cap_path(1,14,g);
hilite_edges(p);
```

Name

max_clique — maximum clique of a graph

```
[size,nodes] = max_clique(g,[ind])
```

Parameters

g
: a graph_data_structure.

ind
: integer (optional)

size
: integer

nodes
: integer row vector

Description

`max_clique` computes the maximum clique of the graph `g` i.e. the complete subgraph of maximum size. `ind` is a parameter for the choice of the method: if `ind` is 0 the method is a partial enumerative algorithm and if `ind` is 1 the algorithm is based on quadratic zero-one programming. The default is 0. The output `size` is the number of the nodes of the clique found by the algorithm and `nodes` is the vector of the corresponding nodes.

Examples

```
ta=[1 2 3 4 5 6 6 7 8  9 10 16 16 10 11 11 12 12 11 14 15 15 13 7 13 13];
he=[2 3 4 5 6 7 8 8 9 10 16  2  3 11 12 13  1 14 14 15  5  9 12 4 14 15];
g=make_graph('foo',0,16,ta,he);
g.nodes.graphics.x=[106 199 369 467 470 403 399 347 308 269 184 108 199 268 345
g.nodes.graphics.y=[341 420 422 321 180 212 286 246 193 244 243 209  59 134  51
show_graph(g);

[ns,no] = max_clique(g);
show_nodes(no);

g1=graph_complement(g);
[ns,no] = max_clique(g1);
show_nodes(no);
```

Name

max_flow — maximum flow between two nodes

```
[v,phi,flag] = max_flow(i,j,g)
```

Parameters

i
: integer, number of start node

j
: integer, number of end node

g
: a graph_data_structure.

v
: value of the maximum flow it is exists

phi
: row vector of the value of the flow on the arcs

flag
: feasible problem flag (0 or 1)

Description

`max_flow` returns the value of maximum flow `v` from node number `i` to node number `j` if it exists, and the value of the flow on each arc as a row vector `phi`. All the computations are made with integer numbers. The graph must be directed. If the problem is not feasible, `flag` is equal to 0, otherwise it is equal to 1.

The bounds of the flow are given by the `min_cap` and `max_cap` fields of the edges_data_structure. The value of the maximum capacity must be greater than or equal to the value of the minimum capacity. If the value of `min_cap` or `max_cap` is not given, they are assumed to be equal to 0 on each edge.

Examples

```
ta=[1 1 2 2 3 3 4 4 5 5 5 5 6 6 6 7 7 15 15 15 15 15 15 15 8 9 10 11 12 13 14];
he=[10 13 9 14 8 11 9 11 8 10 12 13 8 9 12 8 11 1 2 3 4 5 6 7 16 16 16 16 16 16
g=make_graph('foo',1,16,ta,he);
g.nodes.graphics.x=[53,430,202,374,116,250,325,176,373,34,330,233,114,429,208,2
g.nodes.graphics.y=[86,114,115,129,118,122,133,222,222,221,214,219,218,246,40,3
ma=edge_number(g);
g=add_edge_data(g,'max_cap',[2,4,3,3,3,2,3,3,5,2,3,1,2,1,4,1,2,2,2,2,4,1,5,3,2,
g=add_edge_data(g,'min_cap',ones(1,ma))
source=15;g.nodes.graphics.type(source)=2; //source node
sink=16;g.nodes.graphics.type(sink)=1; //sink node
show_graph(g);

[v,phi,ierr]=max_flow(source,sink,g);
g.edges.graphics.foreground(phi<>0)=11;
g=add_edge_data(g,'flow',phi)
g.edges.graphics.display='flow';
show_graph(g);
```

See Also

min_lcost_flow1, min_lcost_flow2, min_qcost_flow, edges_data_structure, add_edge_data

Name

mesh2d — triangulation of n points in the plane

```
[nutr,A] = mesh2d(x,y,[front])
```

Parameters

x
: real row array

y
: real row array

front
: integer row array

nutr
: integer matrix

A
: sparse 0-1 matrix

Description

The arrays `x` and `y` are the coordinates of n points in the plane. `mesh2d` returns a matrix `nutr(3,nbt)` of the numbers of the nodes of the `nbt` triangles of the triangulation of the points. It returns also a sparse matrix `A` representing the connections between the nodes (`A(i,j)=1` if `(i,j)` is a side of one of the triangles or `i=j`). In the case of 3 parameters `front` is the array defining the boundary: it is the array of the indices of the points located on the boundary . The boundary is defined such that the normal to the boundary is oriented towards outside. The boundary is given by its connected components: a component is the part `(i1,i2)` such that `front(i1)=front(i2)` (the external boundary is defined in the counterclockwise way, see the examples below). The error cases are the following: err = 0 if no errors were encountered; err = 3 all nodes are collinear.

If the boundary is given, the other error cases are: err = 2 some points are identical; err = 5 wrong boundary array; err = 6 crossed boundary; err = 7 wrong orientation of the boundary; err = 10 an interior point is on the boundary; err = 8 size limitation; err = 9 crossed boundary; err = 12 some points are identical or size limitation.

Examples

```
// FIRST CASE
theta=0.025*[1:40]*2.*%pi;
x=1+cos(theta);
y=1.+sin(theta);
theta=0.05*[1:20]*2.*%pi;
x1=1.3+0.4*cos(theta);
y1=1.+0.4*sin(theta);
theta=0.1*[1:10]*2.*%pi;
x2=0.5+0.2*cos(theta);
y2=1.+0.2*sin(theta);
x=[x x1 x2];
y=[y y1 y2];
//
nu=mesh2d(x,y);
nbt=size(nu,2);
```

```
jj=[nu(1,:)' nu(2,:)';nu(2,:)' nu(3,:)';nu(3,:)' nu(1,:)'];
as=sparse(jj,ones(size(jj,1),1));
ast=tril(as+abs(as'-as));
[jj,v,mn]=spget(ast);
n=size(x,2);
g=make_graph('foo',0,n,jj(:,1)',jj(:,2)');
g.nodes.graphics.x=300*x;
g.nodes.graphics.y=300*y;
g.nodes.graphics.defaults.diam=10;
show_graph(g)
// SECOND CASE !!! NEEDS x,y FROM FIRST CASE
x3=2.*rand(1:200);
y3=2.*rand(1:200);
wai=((x3-1).*(x3-1)+(y3-1).*(y3-1));
ii=find(wai >= .94);
x3(ii)=[];y3(ii)=[];
wai=((x3-0.5).*(x3-0.5)+(y3-1).*(y3-1));
ii=find(wai <= 0.055);
x3(ii)=[];y3(ii)=[];
wai=((x3-1.3).*(x3-1.3)+(y3-1).*(y3-1));
ii=find(wai <= 0.21);
x3(ii)=[];y3(ii)=[];
xnew=[x x3];ynew=[y y3];
fr1=[[1:40] 1];fr2=[[41:60] 41];fr2=fr2($:-1:1);
fr3=[[61:70] 61];fr3=fr3($:-1:1);
front=[fr1 fr2 fr3];
//
nu=mesh2d(xnew,ynew,front);
nbt=size(nu,2);
jj=[nu(1,:)' nu(2,:)';nu(2,:)' nu(3,:)';nu(3,:)' nu(1,:)'];
as=sparse(jj,ones(size(jj,1),1));
ast=tril(as+abs(as'-as));
[jj,v,mn]=spget(ast);
n=size(xnew,2);
g=make_graph('foo',0,n,jj(:,1)',jj(:,2)');
g.nodes.graphics.x=300*xnew;
g.nodes.graphics.y=300*ynew;
g.nodes.graphics.defaults.diam=10;
show_graph(g)
// REGULAR CASE !!! NEEDS PREVIOUS CASES FOR x,y,front
xx=0.1*[1:20];
yy=xx.*.ones(1,20);
zz= ones(1,20).*.xx;
x3=yy;y3=zz;
wai=((x3-1).*(x3-1)+(y3-1).*(y3-1));
ii=find(wai >= .94);
x3(ii)=[];y3(ii)=[];
wai=((x3-0.5).*(x3-0.5)+(y3-1).*(y3-1));
ii=find(wai <= 0.055);
x3(ii)=[];y3(ii)=[];
wai=((x3-1.3).*(x3-1.3)+(y3-1).*(y3-1));
ii=find(wai <= 0.21);
x3(ii)=[];y3(ii)=[];
xnew=[x x3];ynew=[y y3];
nu=mesh2d(xnew,ynew,front);
nbt=size(nu,2);
jj=[nu(1,:)' nu(2,:)';nu(2,:)' nu(3,:)';nu(3,:)' nu(1,:)'];
as=sparse(jj,ones(size(jj,1),1));
```

```
ast=tril(as+abs(as'-as));
[jj,v,mn]=spget(ast);
n=size(xnew,2);
g=make_graph('foo',0,n,jj(:,1)',jj(:,2)');
g.nodes.graphics.x=300*xnew;
g.nodes.graphics.y=300*ynew;
g.nodes.graphics.defaults.diam=3;
show_graph(g)

//An example with a random set of points
function []=test(X,Y)
  Tr=mesh2d(X,Y);
  plot2d(X,Y,[-1,-2,3]);
  [m,n]=size(Tr);
  xpols= matrix(X(Tr),m,n);
  ypols= matrix(Y(Tr),m,n);
  xset("colormap",rand(2*n,3));
  xfpolys(xpols,ypols,[n/4:n/4+n-1]);
endfunction
N=1000;xbasc();X=rand(1,N); Y=rand(1,N);
xset("wdim",700,700);
test(X,Y);
```

Name

metanet_module_path — Returns the path of the metanet module

```
p = metanet_module_path()
```

Parameters

p
: a string, The module path.

Description

`p=metanet_module_path()` return the path of the metanet module. This path may be useful to retreive Metanet data files.

Examples

```
g=load_graph(metanet_module_path()+'/demos/mesh100.graph');
show_graph(g);
```

Name

min_lcost_cflow — minimum linear cost constrained flow

```
[c,phi,v,flag] = min_lcost_cflow(i,j,cv,g)
```

Parameters

i
: integer, source node number

j
: integer, sink node number

cv
: scalar, value of constrained flow

g
: a graph_data_structure.

c
: value of cost

phi
: row vector of the values of flow on the arcs

v
: value of flow from source to sink

flag
: feasible constrained flow flag (0 or 1)

Description

`min_lcost_cflow` computes the minimum cost flow in the network `g`, with the value of the flow from source node `i` to sink node `j` constrained to be equal to `cv`.

`min_lcost_cflow` returns the total cost of the flows on the arcs `c`, the row vector of the flows on the arcs `phi` and the value of the flow `v` on the virtual arc from sink to source. If `v` is less than `cv`, a message is issued, but the computation is done: in this case `flag` is equal to 0, otherwise it is equal to 1.

The bounds of the flows are given by the `max_cap` fields of the edges_data_structure. The value of the maximum capacity must be non negative and must be integer numbers. If the values of `max_cap` is not given, they are assumed to be equal to 0 on each edge.

The costs on the edges are given by the `cost` field of the edges_data_structure. The costs must be non negative. If the value of `cost` is not given, it is assumed to be equal to 0 on each edge.

This function uses the algorithm of Busacker and Goven.

Examples

```
ta=[1 1 2 2 2 3 4 4 5 6 6 6 7 7 7 8 9 10 12 12 13 13 13 14 15 14 9 11 10];
he=[2 6 3 4 5 1 3 5 1 7 10 11 5 8 9 5 8 11 10 11 9 11 15 13 14 4 6 9 1];
g=make_graph('foo',1,15,ta,he);
```

```
g.nodes.graphics.x=[155,153,85,155,237,244,244,334,338,346,442,440,439,333,438]
g.nodes.graphics.y=[45,145,221,222,221,82,139,225,142,69,140,72,232,318,319];
source=15;g.nodes.graphics.type(source)=2; //source node
sink=1;g.nodes.graphics.type(sink)=1; //sink node
show_graph(g);

g=add_edge_data(g,'max_cap',[11,8,9,19,20,8,9,16,6,11,6,12,12,3,6,14,16,2,14,5,
g=add_edge_data(g,'cost',[9,6,11,7,11,2,8,5,7,10,2,9,10,7,7,9,2,7,2,8,4,6,11,8,
flow_constraint=5;
[c,phi,v,flag]=min_lcost_cflow(source,sink,flow_constraint,g);

g.edges.graphics.foreground(find(phi<>0))=color('red');
g=add_edge_data(g,'flow',phi)
g.edges.graphics.display='flow';
show_graph(g);
```

See Also

max_flow , min_lcost_flow1 , min_lcost_flow2 , min_qcost_flow , edges_data_structure , add_edge_data , nodes_data_structure , add_node_data

Name

min_lcost_flow1 — minimum linear cost flow

```
[c,phi,flag] = min_lcost_flow1(g)
```

Parameters

g
: a graph_data_structure.

c
: value of cost

phi
: row vector of the value of flow on the arcs

flag
: feasible problem flag (0 or 1)

Description

`min_lcost_flow1` computes the minimum linear cost flow in the network `g`. It returns the total cost of the flows on the arcs `c` and the row vector of the flows on the arcs `phi`. If the problem is not feasible (impossible to find a compatible flow for instance), `flag` is equal to 0, otherwise it is equal to 1.

The bounds of the flow are given by the `min_cap` and `max_cap` fields of the graph edges_data_structure. The value of the minimum capacity and of the maximum capacity must be non negative and must be integer numbers. The value of the maximum capacity must be greater than or equal to the value of the minimum capacity. If the value of `min_cap` or `max_cap` is not given it is assumed to be equal to 0 on each edge.

The costs on the edges are given by the element `cost` of the fields of the graph edges_data_structure. The costs must be non negative. If the value of `cost` is not given, it is assumed to be equal to 0 on each edge.

This function uses the out-of-kilter algorithm.

Examples

```
ta=[1,1,2,2,2,3,4,4,5,6,6,6,7,7,7,8,9,10,12,12,13,13,13,14,15,14,9,11,10,1,8];
he=[2,6,3,4,5,1,3,5,1,7,10,11,5,8,9,5,8,11,10,11,9,11,15,13,14,4,6,9,1,12,14];
g=make_graph('foo',1,15,ta,he);
g.nodes.graphics.x=[155,153,85,155,237,244,244,334,338,346,442,440,439,333,438]
g.nodes.graphics.y=[45,145,221,222,221,82,139,225,142,69,140,72,232,318,319];
show_graph(g);

g=add_edge_data(g,'min_cap',[3,7,1,3,6,15,11,3,5,3,12,5,0,5,15,6,2,3,16,7,7,2,8
g=add_edge_data(g,'max_cap',[44,55,49,42,42,63,50,46,38,52,55,46,51,52,67,..
                             57,51,38,57,46,45,52,46,42,52,49,47,58,66,38,46])
g=add_edge_data(g,'cost',[4,6,6,3,8,2,3,3,3,5,10,4,9,5,4,3,11,8,5,4,8,4,6,10,8,
[c,phi,flag]=min_lcost_flow1(g);flag

g.edges.graphics.foreground(find(phi<>0))=color('red');
g=add_edge_data(g,'flow',phi)
```

```
g.edges.graphics.display='flow';
show_graph(g);
```

See Also

min_lcost_cflow , min_lcost_flow2 , min_qcost_flow , edges_data_structure , add_edge_data

Name

min_lcost_flow2 — minimum linear cost flow

```
[c,phi,flag] = min_lcost_flow2(g)
```

Parameters

g
: a graph_data_structure.

c
: value of cost

phi
: row vector of the value of flow on the arcs

flag
: feasible problem flag (0 or 1)

Description

`min_lcost_flow2` computes the minimum linear cost flow in the network `g`. It returns the total cost of the flows on the arcs `c` and the row vector of the flows on the arcs `phi`. If the problem is not feasible (impossible to find a compatible flow for instance), `flag` is equal to 0, otherwise it is equal to 1.

The bounds of the flow are given by the `min_cap` and `max_cap` fields of the graph edges_data_structure. The value of the minimum capacity must be equal to zero. The values of the maximum capacity must be non negative and must be integer numbers. If the value of `min_cap` or `max_cap` are not given , it is assumed to be equal to 0 on each edge.

The costs on the edges are given by the `cost` field of the graph edges_data_structure. The costs must be non negative and must be integer numbers. If the value of `cost` is not given (empty row vector `[]`), it is assumed to be equal to 0 on each edge.

The demand on the nodes are given by the `demand` field of the graph nodes_data_structure. The demands must be integer numbers. Note that the sum of the demands must be equal to zero for the problem to be feasible. If the value of `demand` is not given (empty row vector `[]`), it is assumed to be equal to 0 on each node.

This functions uses a relaxation algorithm due to D. Bertsekas.

Examples

```
ta=[1 1 2 2 2 3 4 4 5 6 6 6 7 7 7 8 9 10 12 12 13 13 13 14 15 14 9 11 10 1 8];
he=[2 6 3 4 5 1 3 5 1 7 10 11 5 8 9 5 8 11 10 11 9 11 15 13 14 4 6 9 1 12 14];
g=make_graph('foo',1,15,ta,he);
g.nodes.graphics.x=[194 191 106 194 296 305 305 418 422 432 552 550 549 416 548
g.nodes.graphics.y=[56 221 316 318 316 143 214 321 217 126 215 80 330 437 439];
show_graph(g);

g=add_edge_data(g,'max_cap',[37,24,23,30,25,27,27,24,34,40,21,38,35,23,38,28,26
                        22,40,22,28,24,31,25,26,24,23,30,22,24,35]);
g=add_edge_data(g,'cost',[10,6,3,8,10,8,11,1,2,6,5,6,5,3,4,2,4,4,8,2,4,5,4,8,8,
g=add_node_data(g,'demand',[22,-29,18,-3,-16,20,-9,7,-6,17,21,-6,-8,-37,9]);
```

```
[c,phi,flag]=min_lcost_flow2(g);flag

g.edges.graphics.foreground(find(phi<>0))=color('red');
g=add_edge_data(g,'flow',phi)
g.edges.graphics.display='flow';
g.nodes.graphics.display='demand';

show_graph(g);
```

See Also

min_lcost_cflow , min_lcost_flow1 , min_qcost_flow , edges_data_structure , add_edge_data , nodes_data_structure , add_node_data

Name

min_qcost_flow — minimum quadratic cost flow

```
[c,phi,flag] = min_qcost_flow(eps,g)
```

Parameters

eps
: scalar, precision

g
: a graph_data_structure.

c
: value of cost

phi
: row vector of the value of flow on the arcs

flag
: feasible problem flag (0 or 1)

Description

`min_qcost_flow` computes the minimum quadratic cost flow in the network `g`. It returns the total cost of the flows on the arcs `c` and the row vector of the flows on the arcs `phi`. `eps` is the precision of the iterative algorithm. If the problem is not feasible (impossible to find a compatible flow for instance), `flag` is equal to 0, otherwise it is equal to 1.

The bounds of the flow are given by the`min_cap` and `max_cap` fielgs of the graph edges_data_structure. The value of the maximum capacity must be greater than or equal to the value of the minimum capacity. If the value of `min_cap` or `max_cap` is not given , it is assumed to be equal to 0 on each edge.

The costs on the edges are given by the elements `q_orig` and `q_weight` of the graph edges_data_structure. The cost on arc `u` is given by:

```
(1/2)*q_weight[u](phi[u]-q_orig[u])^2
```

The costs must be non negative. If the value of `q_orig` or `q_weight` is not given, it is assumed to be equal to 0 on each edge.

This function uses an algorithm due to M. Minoux.

Examples

```
ta=[1 1 2 2 2 3 4 4 5 6 6 6 7 7 7 8 9 10 12 12 13 13 13 14 15 14 9 11 10 1 8];
he=[2 6 3 4 5 1 3 5 1 7 10 11 5 8 9 5 8 11 10 11 9 11 15 13 14 4 6 9 1 12 14];
g=make_graph('foo',1,15,ta,he);
g.nodes.graphics.x=[155,153,85,155,237,244,244,334,338,346,442,440,439,333,438]
g.nodes.graphics.y=[45,177,253,254,253,114,171,257,174,101,172,64,264,350,351];
show_graph(g);

ma=edge_number(g)
g=add_edge_data(g,'min_cap',[0,1,5,5,0,3,3,5,4,1,4,3,3,1,3,1,1,4,1,3,1,4,5,4,4,
```

```
g=add_edge_data(g,'max_cap',[38,37,42,41,34,49,35,36,43,43,43,48,37,..
                                  36,42,48,44,36,30,31,30,41,32,42,34,48,32,36,3
g=add_edge_data(g,'q_weight',ones(1,ma));
[c,phi,flag]=min_qcost_flow(0.001,g);flag

g.edges.graphics.foreground(find(phi<>0))=color('red');
g=add_edge_data(g,'flow',phi)
g.edges.graphics.display='flow';
show_graph(g);
```

See Also

min_lcost_cflow , min_lcost_flow1 , min_lcost_flow2 , edges_data_structure , add_edge_data , nodes_data_structure , add_node_data

Name

min_weight_tree — minimum weight spanning tree

```
t = min_weight_tree([i],g)
```

Parameters

i
: integer, node number of the root of the tree

g
: a graph_data_structure.

t
: row vector of integer numbers of the arcs of the tree if it exists

Description

`min_weight_tree` tries to find a minimum weight spanning tree for the graph `g`. The optional argument `i` is the number of the root node of the tree; its default value is node number 1. This node is meaningless for an undirected graph.

The weights are given by the `weight` field of the graph edges_data_structure. If its value is not given, it is assumed to be equal to 0 on each edge. Weigths can be positive, equal to 0 or negative. To compute a spanning tree without dealing with weights, give to weights a value of 0 on each edge.

`min_weight_tree` returns the tree `t` as a row vector of the arc numbers (directed graph) or edge numbers (undirected graph) if it exists or the empty vector `[]` otherwise. If the tree exists, the dimension of `t` is the number of nodes less 1. If `t(i)` is the root of the tree: - for j < i, `t(j)` is the number of the arc in the tree after node `t(j)` - for j > i, `t(j)` is the number of the arc in the tree before node `t(j)`

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[117,57,7,4,57,57,112,111,145,167,227,232,195,287,291,354,29
g.nodes.graphics.y=[5,42,87,134,89,135,86,137,191,46,80,135,189,197,69,51,126];
g.nodes.graphics.type(1)=2;
show_graph(g);
t=min_weight_tree(1,g);

g.edges.graphics.foreground(t)=color('red');
show_graph(g);
```

Name

neighbors — nodes connected to a node

```
a = neighbors(i,g)
```

Parameters

i
: integer

g
: a graph_data_structure.

a
: vector of integers

Description

`neighbors` returns the numbers of the nodes connected with node i for graph `g` (directed or not).

Examples

```
ta=[1  6  2  4  7  5  6  8  4  3  5  1];
he=[2  1  3  6  4  8  8  7  2  7  3  5];
g=make_graph('foo',1,8,ta,he);
g.nodes.graphics.x=[285  284  335  160  405  189  118  45]/2;
g.nodes.graphics.y=[266  179   83  176  368  252  64  309]/2;
show_graph(g);
a=neighbors(6,g)
hilite_nodes(a);
```

See Also

predecessors , successors

Name

netclose — closes an edit_graph window

```
netclose(window)
```

Parameters

window
: integer, window number

Description

Each `editgraph` window has a window number returned by `edit_graph` and `show_graph` functions. This function is used to close the `edit_graph` window with number `window`.

See Also

edit_graph , netwindow , netwindows , show_graph

Name

netwindow — selects the current edit_graph window

```
netwindow(window)
```

Parameters

window
: integer, window number

Description

This function is used to change the current edit_graph window to those given by the `window` argument. Current edit_graph window is used by show_arcs, show_nodes,.. functions. The numbers of existing windows are given by the function `netwindows`.

See Also

edit_graph , netclose , netwindows , show_graph

Name

netwindows — gets the numbers of edit_graph windows

```
l = netwindows()
```

Parameters

l
: list

Description

This function returns a list `l`. Its first element is the row vector of all the edit_graph windows and the second element is the number of the current edit_graph window. This number is equal to 0 if no current Metanet window exists.

See Also

edit_graph , netclose , netwindow , show_graph

Name

ngraphic_data_structure — data structure representing the graphic properties used for nodes graphical display

Description

A data structure represented by a Scilab `mlist` with type `ngraphic` and 10 fields:

- `display` a string. Gives the information that is displayed with the nodes. The possible values are `'nothing'`, `'number'`, `'name'` or any node data field name as given by the `nodedatafields` function. Of course if `display` is set to `'nothing'` no information is displayed.
- `defaults` A Scilab tlist data structure of type `nodedefs`. Contains the default values for `'type'`, `'diam'`,`'border'`, `'font'`, `'colors'` properties.
- `name` A row vector of strings. The name associated with each node.
- `x` A row vector which gives the abscissae of each node.
- `y` A row vector which gives the ordinate of each node.
- `type` A row vector with integer values which stores the type index for each node. A zero value stands for the default value.
- `diam` A row vector which stores the diameter of each node. A zero value stands for the default value.
- `border`a row vector. The thickness of the polyline used to draw the border of each node. A zero value stands for the default value.
- `colors`a matrix with to rows. The first row contains the color index of the node border, the second row contains the index of the node's background color. A zero value of either color index stands for the corresponding default value.
- `font`a matrix with 3 rows: `font(1,i)` is the font size, `font(2,i)` is the font style,`font(2,i)` is the font color used to draw information on the i th node.A zero value of either entry stands for the corresponding default value.

Examples

```
//create a simple graph
ta=[1  1 2 7 8 9 10 10 10 10 11 12 13 13 4];
he=[2 10 7 8 9 7  7 11 13 13 12 13  9 10 4];
g=make_graph('simple',1,13,ta,he);
g.nodes.graphics.x=[40,33,29,63,146,233,75,42,114,156,237,260,159];
g.nodes.graphics.y=[7,61,103,142,145,143,43,120,145,18,36,107,107];
show_graph(g,'new')

g.nodes.graphics.defaults.border=2;
g.nodes.graphics.defaults.diam=22;
g.nodes.graphics.defaults.colors=[color('red');color('blue')];
show_graph(g)

g.nodes.graphics.border(1:5)=1;
g.nodes.graphics.diam(6)=40;
g.nodes.graphics.type(11)=1;
```

```
show_graph(g)

g.nodes.graphics.display='number';
show_graph(g)

g.nodes.graphics
```

See Also

graph_data_structure

Name

node_number — number of nodes of a graph

```
n = node_number(g)
```

Parameters

g
: graph list

n
: integer, number of nodes

Description

`node_number` returns the number `n` of nodes of the graph.

See Also

arc_number , edge_number

Name

nodedatafields — returns the vector of node data fields names

```
F = nodedatafields(g)
```

Parameters

g
: a graph data structure (see graph_data_structure)

F
: a row vector of strings. Each element is a field name of the nodes data data structure.

Description

It is possible to associate data to the nodes of a graph. This can be done with the add_node_data function. the `nodedatafields` function allows to retreive the field names of these data. A given node data can be referenced by its field name `g.nodes.data(field_name)`.

Examples

```
//create a simple graph
ta=[1  1 2 7 8 9 10 10 10 10 11 12 13 13];
he=[2 10 7 8 9 7  7 11 13 13 12 13  9 10];
g=make_graph('simple',1,13,ta,he);
g.nodes.graphics.x=[40,33,29,63,146,233,75,42,114,156,237,260,159];
g.nodes.graphics.y=[7,61,103,142,145,143,43,120,145,18,36,107,107];
show_graph(g,'new')
nodedatafields(g)


g=add_node_data(g,'demand',round(10*rand(1,13,'u')));
g=add_node_data(g,'label','e'+string(1:13));
nodedatafields(g)
g.nodes.data.label
```

See Also

graph_data_structure , add_node_data

Name

nodes_2_path — path from a set of nodes

```
p = nodes_2_path(ns,g)
```

Parameters

ns
: row vector of integer numbers of the set of nodes

g
: a graph_data_structure.

p
: row vector of integer numbers of the arcs of the path if it exists

Description

`nodes_2_path` returns the path `p` corresponding to the node sequence `ns` given by its node numbers if it exists ; it returns the empty vector `[]` otherwise.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
show_graph(g);
ns=[1 10 15 16 17 14 11 12 13 9 7 8 6];
g.nodes.graphics.colors(2,ns)=color('red');
show_graph(g);

p=nodes_2_path(ns,g);
g.edges.graphics.foreground(p)=color('red');
show_graph(g);
```

See Also

path_2_nodes

Name

nodes_data_structure — description of the data structure representing the nodes of a graph

Description

A nodes data structure is represented by a Scilab `mlist` with type `nodes` and 3 fields:

- `number` the number of nodes.
- `graphics`> A Scilab `mlist` data structure of type `ngraphic` which stores the information relative to nodes display (see ngraphic_data_structure
- `data` A Scilab `mlist` data structure of type `nodedata`. which stores the data associated with nodes. By defaut this data structure is empty. User can add its own fields using the `add_node_data` function..

  For a given field the associated data should be a row vector or a matrix. In the matrix case a column is associated to a node.

Examples

```
//create a simple graph
ta=[1  1 2 7 8 9 10 10 10 10 11 12 13 13];
he=[2 10 7 8 9 7  7 11 13 13 12 13  9 10];
g=make_graph('simple',1,13,ta,he);
g.nodes.graphics.x=[40,33,29,63,146,233,75,42,114,156,237,260,159];
g.nodes.graphics.y=[7,61,103,142,145,143,43,120,145,18,36,107,107];
show_graph(g,'new')

g.nodes.number
g.nodes

g=add_node_data(g,'Size',rand(1,g.nodes.number,'u'));
g.nodes(1:10)

nodedatafields(g)
```

See Also

graph_data_structure , add_node , delete_nodes , nodedatafields , add_node_data

Name

nodes_degrees — degrees of the nodes of a graph

```
[outdegree,indegree] = graph_degree(g)
```

Parameters

g
: a graph_data_structure.

outdegree
: row vector of the out degrees of the nodes

indegree
: row vector of the in degrees of the nodes

Description

`nodes_degrees` returns the 2 row vectors of the out and in degrees of the nodes of the graph `g`.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
show_graph(g);
[outdegree,indegree]=nodes_degrees(g)

g=add_node_data(g,'outdegree',outdegree)
g.nodes.graphics.display='outdegree';
show_graph(g);

g=add_node_data(g,'indegree',indegree)
g.nodes.graphics.display='indegree';
show_graph(g,'new');
```

See Also

adj_lists

Name

path_2_nodes — set of nodes from a path

```
ns = path_2_nodes(p,g)
```

Parameters

p
: row vector of integer numbers of the arcs of the path

g
: a graph_data_structure.

ns
: row vector of integer numbers of the set of nodes

Description

`path_2_nodes` returns the set of nodes `ns` corresponding to the path `p` given by its arc numbers ; if `p` is not a path, the empty vector `[]` is returned.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
p=[2 16 23 25 26 22 17 18 19 13 10 11];
g.edges.graphics.foreground(p)=color('red')
show_graph(g);

ns=path_2_nodes(p,g);
hilite_nodes(ns);
```

See Also

nodes_2_path

Name

perfect_match — min-cost perfect matching

```
[cst,nmatch] = perfect_match(g,arcost)
```

Parameters

g
: :a undirected graph (see graph_data_structure).

arcost
: integer row vector

cst
: integer

nmatch
: integer row vector

Description

`perfect_match` finds a perfect min-cost matching for the graph `g`. `g` must be an undirected graph with an even number of nodes. `arcost` is the vector of the (integer) costs of the arcs (the dimension of `arcost` is twice the number of edges of the graph). The output is the vector `nmatch` of the perfect matching and the corresponding cost `cst`.

Examples

```
ta=[27 27 3 12 11 12 27 26 26 25 25 24 23 23 21 22 21 20 19 18 18];
ta=[ta  16 15 15 14 12 9 10 6 9 17 8 17 10 20 11 23 23 12 18 28];
he=[ 1  2 2  4  5 11 13  1 25 22 24 22 22 19 13 13 14 16 16  9 16];
he=[he  10 10 11 12  2 6  5 5 7  8 7  9  6 11  4 18 13  3 28 17];
n=28;
g=make_graph('foo',0,n,ta,he);
xx=[46 120 207 286 366 453 543 544 473 387 300 206 136 250 346 408];
g.nodes.graphics.x=[xx 527 443 306 326 196 139 264  55  58  46 118 513];
yy=[36  34  37  40  38  40  35 102 102  98  93  96 167 172 101 179];
g.nodes.graphics.y=[yy 198 252 183 148 172 256 259 258 167 109 104 253];
show_graph(g);

m2=2*size(ta,2);
arcost=round(100.*rand(1,m2));
[cst,nmatch] = perfect_match(g,arcost);
v=index_from_tail_head(g,1:n,nmatch)
hilite_edges(v);
```

See Also

best_match

Bibliography

U. Derigs "Solving non-bipartite matching problems via shortest path techniques" Annals of operations research 7, 1988.

Name

pipe_network — solves the pipe network problem

```
[x,pi] = pipe_network(g)
```

Parameters

g
: a graph_data_structure.

x
: row vector of the value of the flow on the arcs

pi
: row vector of the value of the potential on the nodes

Description

`pipe_network` returns the value of the flows and of the potentials for the pipe network problem: flow problem with two Kirchhoff laws. The graph must be directed. The problem must be feasible (the sum of the node demands must be equal to 0). The resistances on the arcs must be strictly positive and are given as the values of the weight field of the graph edges_data_structure.

The problem is solved by using sparse matrices LU factorization.

Examples

```
ta=[1 1 2 2 3 3 4 4 5 5 5 5 6 6 6 7 7 15 15 15 15 15 15 15 8 9 10 11 12 13 14];
he=[10 13 9 14 8 11 9 11 8 10 12 13 8 9 12 8 11 1 2 3 4 5 6 7 16 16 16 16 16 16
g=make_graph('foo',1,16,ta,he);
g.nodes.graphics.x=[42 615 231 505 145 312 403 233 506 34 400 312 142 614 260 2
g.nodes.graphics.y=[143 145 154 154 147 152 157 270 273 279 269 273 273 274 50
g.nodes.graphics.diam(15:16)=30;
g=add_node_data(g,'demand',[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -100 100]);
w = [1 3 2 6 4 7 8 1 2 2 2 4 7 8 9 2 3 5 7 3 2 5 8 2 5 8  6 4 3 5 6];
g=add_edge_data(g,'weight',w);
g.nodes.graphics.display='demand';
g.edges.graphics.display='weight';
show_graph(g);

[x,pi] = pipe_network(g)
g=add_edge_data(g,'flow',round(100*x)/100);
g.edges.graphics.display='flow';

show_graph(g);
```

Name

plot_graph — general plot of a graph (obsolete)

```
plot_graph(g,[rep,rep1])
```

Parameters

g
: a graph_data_structure.

rep
: row vector of 13 values for the parameters of the plot

rep1
: row vector of 4 values defining the plotting rectangle

Description

This function is osolete, use show_graph instead

`plot_graph` plots graph `g` in a Scilab graphical window. The optional arguments `rep` and `rep1` define the parameters of the plot. If there are not given, a dialog box for the definition of these parameters is opened.

`rep` must be a row vector with 13 integer numbers which must be 1 or 2. The meaning of the values of `rep` are:

Frame definition: 1 = Automatic 2 = Given (see below)

Plotting arrows: 1 = yes, 2 = no

Plotting sink and source nodes: 1 = yes, 2 = no

Plotting node names: 1 = yes, 2 = no

Plotting node labels: 1 = yes, 2 = no

Plotting arc names : 1 = yes, 2 = no

Plotting arc labels: 1 = yes, 2 = no

Plotting node demand: 1 = yes, 2 = no

Plotting edge length: 1 = yes, 2 = no

Plotting edge cost: 1 = yes, 2 = no

Plotting edge min cap: 1 = yes, 2 = no

Plotting edge max cap: 1 = yes, 2 = no

Plotting edge weight: 1 = yes, 2 = no

If `rep(1)` is 2, the frame definition must be given by `rep1`. Otherwise, `rep1`can be omitted. `rep1` must be a row vector `[orx,ory,w,h]` giving respectively the coordinates of the upper-left point, the width and the height of the plotting rectangle.

Examples

```
// simple graph with different choices for the plot
ta=[2 2 1 1 2 4 3 3 4];
he=[2 2 3 2 3 2 1 2 1];
g=make_graph('foo',1,4,ta,he);
g.nodes.graphics.type=[1 1 1 2];
g.nodes.graphics.name=string([1:4]);
g.nodes.graphics.x=[73 737 381 391];
g.nodes.graphics.y=[283 337 458 142];
g.nodes.graphics.colors(1,:)=[3 3 3 11];
g.nodes.graphics.diam=[30 30 30 60];
g.edges.graphics.foreground=[10 0 2 6 11 11 0 0 11];
rep=[2 2 1 1 2 2 2 2 2 2 2];
rep1=[0 0 850 500];
clf(); plot_graph(g,rep,rep1);

rep=[2 1 1 1 2 2 2 2 2 2 2 2];
clf(); plot_graph(g,rep,rep1);
// plotting using dialogs
clf(); plot_graph(g);
xset("thickness",4);
clf();
plot_graph(g);
```

See Also

show_graph

Name

predecessors — tail nodes of incoming arcs of a node

```
a = predecessors(i,g)
```

Parameters

i
: integer

g
: a graph_data_structure.

a
: row vector of integers

Description

`predecessors` returns the row vector of the numbers of the tail nodes of the incoming arcs to node `i` for a directed graph `g` .

Examples

```
ta=[1  6  2  4  7  5  6  8  4  3  5 1];
he=[2  1  3  6  4  8  8  7  2  7  3 5];
g=make_graph('foo',1,8,ta,he);
g.nodes.graphics.x=[285  284  335  160  405  189  118  45];
g.nodes.graphics.y=[266  179   83  176  368  252  64  309];
g.nodes.graphics.colors(2,8)=color('red');

show_graph(g);
a=predecessors(8,g)
hilite_nodes(a);
```

See Also

neighbors , successors

Name

qassign — solves a quadratic assignment problem

```
[crit,order] = qassign(c,f,d)
```

Parameters

c
: real matrix

f
: real matrix

d
: real matrix

crit
: real scalar

order
: integer row vector

Description

`qassign` solves the quadratic assignment problem i.e. minimize the global criterium: `crit = e(1)+...+e(n)` where `e(i) = c(i,l(i))+ fd(i)` where `fd(i) = f(i,1)*d(l(i),l(1))+...+f(i,n)*d(l(i),l(n))`

`c`, `f` and `d` are n x n real arrays; their diagonal entries are zero.

Examples

```
n=15;
d=100*rand(15,15);
d=d-diag(diag(d));
c=zeros(n,n);f=c;
f(2:n,1)=ones(1:n-1)';
[crit,order]=qassign(c,f,d)
```

See Also

knapsack

Name

salesman — solves the travelling salesman problem

```
cir = salesman(g,[nstac])
```

Parameters

g
: a graph_data_structure.

nstac
: integer

cir
: integer row vector

Description

`salesman` solves the travelling salesman problem. `g` is a directed graph; `nstac` is an optional integer which is a given bound for the allowed memory size for solving this problem. Its value is 100*n*n by default where n is the number of nodes.

Examples

```
ta=[2  1 3 2 2 4 4 5 6 7 8 8 9 10 10 10 10 11 12 13 13 14 15 16 16 17 17];
he=[1 10 2 5 7 3 2 4 5 8 6 9 7 7 11 13 15 12 13  9 14 11 16 1 17 14 15];
g=make_graph('foo',0,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
show_graph(g);

//replace edges by a couple of arcs
g1=make_graph('foo1',1,17,[ta he],[he ta]);
m=arc_number(g1);
g1=add_edge_data(g1,'length',5+round(30*rand(1,m)));
cir = salesman(g1);

ii=find(cir > edge_number(g));
if(ii <> []) then cir(ii)=cir(ii)-edge_number(g);end;
hilite_edges(cir);
```

Bibliography

Applegate, D.; Bixby, R.; Chvatal, V.; and Cook, W. "Solving Traveling Salesman Problems." http://www.tsp.gatech.edu/.

Hoffman, P. The Man Who Loved Only Numbers: The Story of Paul Erdos and the Search for Mathematical Truth. New York: Hyperion, pp. 168-169, 1998.

Kruskal, J. B. "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem." Proc. Amer. Math. Soc. 7, 48-50, 1956.

Lawler, E.; Lenstra, J.; Rinnooy Kan, A.; and Shmoys, D. The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. New York: Wiley, 1985.

Name

save_graph — saves a graph in a file

```
save_graph(g,path)
```

Parameters

g
: a graph_data_structure.

path
: string, the path of the graph to save

Description

`save_graph` saves the graph `g` in a graph file. `path` is the name of the graph file where the graph will be saved. `path` can be the name or the pathname of the file; if the `"graph"` extension is missing in `path`, it is assumed. If `path` is the name of a directory, the name of the graph is used as the name of the file.

Standard save function may also be used to save a graph in a file. In this case take care to save only a single graph data structure in the file (without any other variable) if you want to reload this file with load_graph or edit_graph.

Examples

```
g=load_graph(metanet_module_path()+'/demos/mesh100.graph');
show_graph(g);
save_graph(g,'mymesh100.graph');
g=load_graph('mymesh100.graph');
show_graph(g,'new');
```

See Also

load_graph , edit_graph , graph_data_structure , save , load

Name

set_nodes_id — displays labels near selected nodes in a graph display.

```
set_nodes_id(nodes,Id,loc)
```

Parameters

nodes
: vector of integers, the selected nodes

Id
: vector of strings: the labels to be drawn with the nodes;

loc
: string, with possible values : `"center"`, `"right"`, `"left"`, `"up"` and `"downn`. Specify where the labels will be drawn.

Description

`set_nodes_id(nodes,Id,loc)` allows to display labels near selected nodes of the displayed graph in the current `edit_graph` window (see netwindow). Note that these labels are not stored in the corresponding graph data structure.

Examples

```
 ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
show_graph(g);
set_nodes_id(1:3,'n'+string(1:3),'right')
```

See Also

edit_graph , hilite_nodes , unhilite_nodes , ngraphic_data_structure , show_nodes , netwindow , netwindows

Name

shortest_path — shortest path

```
[p,lp] = shortest_path(i,j,g,[typ])
```

Parameters

i
: integer, number of start node

j
: integer, number of end node

g
: a graph_data_structure.

typ
: string, type of shortest path

p
: row vector of integer numbers of the arcs of the shortest path if it exists

lp
: length of shortest path

Description

`shortest_path` returns the shortest path `p` from node `i` to node `j` if it exists, and the empty vector `[]` otherwise. The optional argument `typ` is a string which defines the type of shortest path, 'arc' for the shortest path with respect to the number of arcs and 'length' for the shortest path with respect to the length of the edges `edge_length`.

For the shortest path with respect to the length of the edges, the lengths are given by the element `edge_length` of the graph list. If its value is not given (empty vector `[]`), it is assumed to be equal to 0 on each edge. Lengths can be positive, equal to 0 or negative.

When a shortest path exists, `lp` is the length of this path.

Examples

```
rand('uniform');

ta=[1 1 2 2 2 3 4 4 5 6 6 6 7 7 7 8 9 10 12 12 13 13 13 14 15 14 9 11 10];
he=[2 6 3 4 5 1 3 5 1 7 10 11 5 8 9 5 8 11 10 11 9 11 15 13 14 4 6 9 1];
g=make_graph('foo',1,15,ta,he);
g.nodes.graphics.x=[194 191 106 194 296 305 305 418 422 432 552 550 549 416 548
g.nodes.graphics.y=[56 181 276 278 276 103 174 281 177 86 175 90 290 397 399]*0
g=add_edge_data(g,'length',int(20*rand(ta)));
g.edges.graphics.display='length';
show_graph(g);
[p,lp]=shortest_path(13,1,g,'length');p
hilite_edges(p);
```

See Also

find_path , nodes_2_path

Name

show_arcs — highlights a set of arcs

```
show_arcs(p,[sup])
```

Parameters

p
: row vector of arc numbers (directed graph) or edge numbers (undirected graph)

sup
: string, superposition flag

Description

`show_arcs` highlights the set of arcs or edges `p` of the displayed graph in the current edit_graph window. If the optional argument `sup` is equal to the string 'sup', the highlighting is superposed on the previous one.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642]
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301
show_graph(g);
t=min_weight_tree(1,g); g1=g; ma=edge_number(g1);
edgecolor=1*ones(1,ma); g1('edge_color')=edgecolor;
edgewidth=1*ones(1,ma); edgewidth(t)=4*ones(t); g1('edge_width')=edgewidth;
for i=8:12,
 edgecolor(t)=i*ones(t); g1('edge_color')=edgecolor;
 xpause(3d5); show_graph(g1);
 show_arcs(t);
end;
```

See Also

edit_graph , show_nodes , netwindow , netwindows

Name

show_edges — highlights a set of edges

```
show_edges(p [,sup])
show_edges(p ,sup=value)
show_edges(p ,leg=value)
show_edges(p ,sup=value,leg=value)
```

Parameters

p
: vector of arc numbers (directed graph) or edge numbers (undirected graph)

sup
: string, superposition flag. The default value is `'no'`.

leg
: string, data field to be displayed. The default value is `'nothing'`.

Description

`show_edges` highlights the set of arcs or edges `p` of the displayed graph in the current edit_graph window (see netwindow). If the optional argument `sup` is equal to the string 'sup', the highlighting is superposed on the previous one.

If leg is equal to `'number'` the edge numbers are also drawn.

If leg is equal to `'name'` the edge names are also drawn.

If leg is equal to one of the edges data fields `'name'` the corresponding values are also drawn.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
show_graph(g);

t=min_weight_tree(1,g);
show_edges(t);
show_edges(t,leg='number');
```

See Also

edit_graph , hilite_edges , unhilite_edges , show_nodes , netwindow , netwindows

Name

show_graph — displays a graph

```
nw = show_graph(g,[smode,scale])
nw = show_graph(g,[scale,winsize])
```

Parameters

g
: a graph_data_structure.

smode
: string, mode value

winsize
: row vector defining the size of edit_graph window

scale
: real value, scale factor

nw
: integer

Description

`show_graph` displays the graph `g` in the current edit_graph window. If there is no current edit_graph window, a edit_graph window is created. The return value `nw` is the number of the edit_graph window where the graph is displayed.

If the optional argument `smode` is equal to the string 'rep' or is not given and if there is already a graph displayed in the current edit_graph window, the new graph is displayed instead.

If the optional argument `smode` is equal to the string 'new', a new edit_graph window is created. In this case, if the optional argument `winsize` is given as a row vector `[width height]`, it is the size in pixels of edit_graph window. The default is `[600,400]`.

The optional argument `scale` is the value of the scale factor when drawing the graph. The default value is 1.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
show_graph(g,2);
show_graph(g,0.5);
show_graph(g,1);
```

See Also

edit_graph , netwindow , netwindows

Name

show_nodes — highlights a set of nodes

```
show_nodes(nodes [,sup])
show_nodes(nodes ,sup=value)
show_nodes(nodes ,leg=value)
show_nodes(nodes ,sup=value,leg=value)
```

Parameters

nodes
: row vector of node numbers

sup
: string, superposition flag. The default value is `'no'`.

leg
: string, data field to be displayed. The default value is `'nothing'`.

Description

`show_nodes` highlights the set of nodes `nodes` of the displayed graph in the current `edit_graph` window (see netwindow). If the optional argument `sup` is equal to the string 'sup', the highlighting is superposed on the previous one.

If leg is equal to `'number'` the node numbers are also drawn.

If leg is equal to `'name'` the node names are also drawn.

If leg is equal to one of the node data fields the corresponding values are also drawn.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
show_graph(g);
for i=2:3:g.nodes.number, show_nodes(i); end;
for i=1:3:g.nodes.number, show_nodes(i,'sup'); end;
show_nodes(1:3:g.nodes.number,leg='number')
```

See Also

edit_graph , hilite_nodes , unhilite_nodes , show_arcs , netwindow , netwindows

Name

split_edge — splits an edge by inserting a node

```
g1 = split_edge(i,j,g,name)
```

Parameters

i
: integer, number of start node of edge

j
: integer, number of end node of edge

g
: a graph_data_structure.

name
: optional name of the added node

g1
: graph data structure of the new graph

Description

`split_edge` returns the graph `g1`, the edge from node number `i` to node number `j` being splitted: a new node is created and located at the middle point between the 2 previous nodes. This new node is linked with the 2 nodes `i` and `j`. If `name` is given, it is the name of the new node, otherwise the number of nodes plus 1 is taken as the name of the new node.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 13 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
g.edges.graphics.foreground(index_from_tail_head(g,1,2))=5;
g.nodes.graphics.display='number';
show_graph(g);

gt=split_edge(1,2,g);
gt.nodes.graphics.colors(2,$)=color('red')
gt.edges.graphics.foreground($-1:$)=color('red')
show_graph(gt,'new');
```

See Also

add_edge , add_node , delete_arcs , delete_nodes

Name

strong_con_nodes — set of nodes of a strong connected component

```
ns = strong_con_nodes(i,g)
```

Parameters

i
: integer, number of the strong connected component

g
: a graph_data_structure.

ns
: row vector, node numbers of the strong connected component

Description

`strong_con_nodes` returns the row vector `ns` of the numbers of the nodes which belong to the strong connected component number `i`.

Examples

```
ta=[1 1 2 2 2 3 4 4 5 6 6 6 7 7 7 8 9 10 12 12 13 13 13 14 15];
he=[2 6 3 4 5 1 3 5 1 7 10 11 5 8 9 5 8 11 10 11 9 11 15 13 14];
g=make_graph('foo',1,15,ta,he);
g.nodes.graphics.x=[197 191 106 194 296 305 305 418 422 432 552 550 549 416 548
g.nodes.graphics.y=[76 181 276 278 276 83 174 281 177 86 175 90 290 397 399];
show_graph(g);


ncomp=strong_con_nodes(3,g);
g.nodes.graphics.colors(2,ncomp)=color('red');
show_graph(g);
```

See Also

connex , con_nodes , strong_connex

Name

strong_connex — strong connected components

```
[nc,ncomp] = strong_connex(g)
```

Parameters

g
: a graph_data_structure.

nc
: integer, number of strong connected components

ncomp
: row vector of strong connected components

Description

`strong_connex` returns the number `nc` of strong connected components for the graph `g` and a row vector `ncomp` giving the number of the strong connected component for each node. For instance, if `i` is a node number, `ncomp(i)` is the number of the strong connected component to which node `i` belongs.

Examples

```
ta=[1 1 2 2 2 3 4 4 5 6 6 6 7 7 7 8 9 10 12 12 13 13 13 14 15];
he=[2 6 3 4 5 1 3 5 1 7 10 11 5 8 9 5 8 11 10 11 9 11 15 13 14];
g=make_graph('foo',1,15,ta,he);
g.nodes.graphics.x=[197 191 106 194 296 305 305 418 422 432 552 550 549 416 548
g.nodes.graphics.y=[76 181 276 278 276 83 174 281 177 86 175 90 290 397 399];
show_graph(g);

[nc,ncomp]=strong_connex(g);
colors=[2 7 5 29 24 1];
g=add_node_data(g,'component_number',ncomp)
g.nodes.graphics.colors(1,:)=colors(ncomp);
g.nodes.graphics.display='component_number';

show_graph(g);
```

See Also

connex , con_nodes , strong_con_nodes

Name

subgraph — subgraph of a graph

```
g1 = subgraph(v,ind,g)
```

Parameters

v
: row vector, numbers of nodes or edges

ind
: string, 'nodes' or 'edges'

g
: a graph_data_structure.

g1
: graph data structure of the new graph

Description

`subgraph` returns the graph `g1`, built with the numbers given by the the row vector `v`. If `ind` is the string `'nodes'`, `g1` is built with the node numbers given by `v` and the connected edges of these nodes in `g`. If `ind` is the string `'edges'`, `g1` is built with the edge numbers given by `v` and the tail-head nodes of these edges in `g`.

All the characteristics of the old nodes and edges of `g` are preserved.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 13 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
w=show_graph(g);

v=[2 3 4 5 17 13 10];
hilite_nodes(v);
g1=subgraph(v,'nodes',g);
w1=show_graph(g1,'new');


v=[10 13 12 16 20 19];
netwindow(w)
show_graph(g);
hilite_edges(v);
g1=subgraph(v,'edges',g);
netwindow(w1);
show_graph(g1);
```

See Also

add_edge , add_node , delete_arcs , delete_nodes , supernode

Name

successors — head nodes of outgoing arcs of a node

```
a = successors(i,g)
```

Parameters

i
: integer

g
: a graph_data_structure.

a
: row vector of integers

Description

`successors` returns the row vector of the numbers of the head nodes of the outgoing arcs from node i for a directed graph `g` .

Examples

```
ta=[1  6  2  4  7  5  6  8  4  3  5  1];
he=[2  1  3  6  4  8  8  7  2  7  3  5];
g=make_graph('foo',1,8,ta,he);
g.nodes.graphics.x=[285  284  335  160  405  189  118  45];
g.nodes.graphics.y=[266  179   83  176  368  252  64  309];
g.nodes.graphics.colors(2,6)=5;
show_graph(g);
a=successors(6,g)
hilite_nodes(a);
```

See Also

neighbors , predecessors

Name

supernode — replaces a group of nodes with a single node

```
g1 = supernode(v,g)
```

Parameters

v
: row vector, nodes numbers

g
: a graph_data_structure.

g1
: graph data structure of the new graph

Description

`supernode` returns the graph `g1` with the nodes with numbers given by the vector `v` being contracted in a single node. The number of the supernode is the lowest number in `v`. The characteristics of the old nodes and edges are preserved. The supernode is located at the mean center of `v`. Its diameter and border are twice the previous of the replaced node.

The demand of the new node, if it exists, is the sum of the demands of the shrunken nodes.

Examples

```
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 13 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g.nodes.graphics.x=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 7
g.nodes.graphics.y=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 
show_graph(g);

v=[7 10 13 9];
hilite_nodes(v);
g1=supernode(v,g);

show_graph(g1,'new');

//verify by superimposing the two graphs
g1.nodes.graphics.diam(:)=g2.nodes.graphics.defaults.diam/2
g1.nodes.graphics.colors(2,:)=5;
g1.edges.graphics.foreground(:)=5;
g.edges.graphics.width(:)=4;
show_graph(graph_union(g,g1,%f))
```

See Also

add_edge , add_node , delete_arcs , delete_nodes

Name

trans_closure — transitive closure

```
g1 = trans_closure(g)
```

Parameters

g
: a graph_data_structure.

g1
: a graph data structure

Description

The transitive closure of a graph is a graph which contains an edge from node u to node v whenever there is a directed path from u to v.

`trans_closure` returns as a new graph `g1` the transitive closure of the graph `g`. This graph must be directed and connected. If `<name>` if the name of graph `g`, `<name>_trans_closure` is the name of the transitive closure.

Examples

```
ta=[2 3 3 5 3 4 4 5 8];
he=[1 2 4 2 6 6 7 7 4];
g=make_graph('foo',1,8,ta,he);
g.nodes.graphics.x=[129 200 283 281 128 366 122 333];
g.nodes.graphics.y=[61 125 129 189 173 135 236 249];
show_graph(g);
g1=trans_closure(g);

vv=1*ones(ta); aa=sparse([ta' he'],vv');
ta1=g1.edges.tail; he1=g1.edges.head;
ww=1*ones(ta1); bb=sparse([ta1' he1'],ww');
dif=bb-aa; lim=size(ta1); edgecolor=0*ones(ta1);
for i=1:lim(2)
 if dif(ta1(i),he1(i))==1 then edgecolor(i)=11; end;
end;
g1.edges.graphics.foreground=edgecolor;
show_graph(g1);
```

Name

update_graph — converts an old graph data structure to the current one.

```
Gnew = update_graph(G)
```

Parameters

G
: Scilab-4.x graph data structure.

Gnew
: new graph data structure (see graph_data_structure).

Description

Converts a Scilab-4.x graph data structure to the new one.

In the future this function will be used to adapt graph data structures according to their versions.

See Also

graph_data_structure , graph-list

Online help management

Name

add_help_chapter — Add an entry in the helps list

```
add_help_chapter(title,path[,mode])
```

Parameters

title
: a character string, the help chapter title

path
: a character string, the path of the directory containing the help files.

mode
: a boolean, %T if the help directory belongs to scilab modules list, %F else (toolboxes). The default value is %F.

Description

This function adds a new entry in the helps list. The help chapter files are to be located in a single directory. If the given `title` already exists in the helps list associated with the same path nothing is done. The function checks if the directory exist.

See Also

help , add_demo

Authors

Serge Steer , INRIA

Name

apropos — searches keywords in Scilab help

```
apropos(key)
apropos(regexp)
```

Parameters

key
: character string. give the sequence of characters to be found

regexp
: character string. give the regular expression to be found (only with "Scilab Browser")

Description

`apropos(key)` looks for Scilab help files containing keywords `key`in their short description section.

`apropos(regexp)` looks for Scilab help files containing regular expression `regexp`in their short description section.

Examples

```
apropos('ode')
apropos ode
apropos "list of"
apropos "sin.*hyperbolic"
apropos "^ab"  //search help beginning the two characters "ab"
apropos "quadratic.*solver"
```

See Also

help , man

Name

foo — foo short description

```
[y] = foo(x)
```

Parameters

x
: what may be x

y
: what may be y

Description

A first paragraph which explains what computes the foo function. If you want to emphasis a parameter name then you use the following tag `x`, if you want to emphasis a part of text *inclose it inside theses tags* and use theses ones **to have a bold font** and finally `for a type writer style`.

A second paragraph... Here is an example of a link to another page : man.

first
: simple paragraph

second
: toto is the french foo...

Examples

```
deff("y=foo(x)","y=x"); // define the foo function as the identity function
foo("toto")
```

See Also

man , apropos

Authors

B. P.

Name

help — on-line help command

```
help(key)
help
```

Parameters

key
: character string. Gives the help page to be found

Description

`help` without argument displays the hypertext page containing the list of help chapters.

`help(key)` displays the Scilab help file associated with the given `key`. If no help file is found, `help(key)` automatically call `apropos(key)`.

See man for more explanation on how to write new help pages .

See Also

apropos , man

Name

help_from_sci — Generate help files and demo files from the head comments section of a .sci source file.

```
help_from_sci() // generate an empty function template

help_from_sci(funname,helpdir) // generate helpdir/funname.xml from funname.sci

help_from_sci(dirname,helpdir) // process dirname/*.sci and create helpdir/*.xm

help_from_sci(dirname,helpdir,helpdir) // as above but also creating helpdir/*.

[helptxt,demotxt]=help_from_sci(funname) // return funname.xml and funname.dem.
```

Parameters

funname:
: the name of a single .sci source file to be processed.

dirname:
: directory name where all .sci files will be processed.

helpdir:
: optional path where the .xml help file will be created.

demodir:
: optional path where .dem.sce demo files will be created based on code from the Examples section.

helptxt:
: returns the XML help code if helpdir is empty, or the path to the new .xml file.

demotxt:
: returns the demo code if demodir is empty, or the path to the new .dem.sc file.

Description

help_from_sci is a revised version of the help_skeleton function. Its objective is to generate .xml help files based on the head comments section of .sci source files. Optionally .dem.sce demo files can be generated based on code from the Examples section in the head comments section of .sci files.

In order for help_from_sci to format the .xml file properly the head comments section should comply with some simple formatting rules.

The first comment line following the function definition should contain a short description of the function.

The remaining comments are formatted according to the following (optional) headlines: "Calling Sequence", "Parameters", "Description", "Examples", "See also", "Used functions", "Authors" and "Bibliography".

The following guidelines should be used when writing the source code comments:

- `Calling Sequence` - one example pr. line.
- `Parameters` - separate parameter name and description by a ":". Keep the description of each parameter on the same line.
- `Description` - formatting of the text can be done using XML commands. Adding an empty comment line in the Description section is interpreted as the start of a new paragraph.

- `See also` - list one function name pr line.
- `Authors` - write one author on each line following the Authors headline. Use ";" to separate the authors name from any add additional information.
- `Bibliography` - write one reference pr line following the References headline.

Examples

```
help_from_sci()   // Open an empty source code template in the Scipad editor.
// Save this template as test_fun.sci in the current directory before running
// the next example commands.

help_from_sci('test_fun')        // return the xml skeleton as a text string

help_from_sci('test_fun','.')    // create the xml help file in the current dir

// create both a xml help file and a demo file in the current directory.
help_from_sci('test_fun','.','.')

// From a toolbox root directory a typical calling sequence would be:
// help_from_sci('macros','help\en_US','demos')
// This command would process all .sci files in the macros directory
// and use the head comments section to update all .xml help files in the
// help\en_US directory an rebuild the .dem.sce files in the demos\ directory.
```

See also

help, help_skeleton, xmltohtml

Authors

T. Pettersen
torbjorn.pettersen@broadpark.no

Name

help_skeleton — build the skeleton of the xml help file associated to a Scilab function

```
txt = help_skeleton(funname [,path [,language]])
```

Parameters

funname
: character string : the name of the function

path
: character string : the path where the file will be create if required. If this argument is not given the skeleton is returned as a string.

language
: character string :with possible value "fr_FR" or "en_US" the defaultis "en_US"

txt
: the XML code or the complete xml file path

Description

`txt = help_skeleton(funname)` generates a vector of strings containing the skeleton of the XML code describing the help of the function `funname`.

`fullpath = help_skeleton(funname,dirpath)` generates the XML code describing the help of the function `funname` in a file named `funname.xml` in the directory specified by the path `dirpath`. In this case the function returns the file path.

Examples

```
function [y,z]=foo(a,b),y=a+b,z=1,endfunction
 p=help_skeleton('foo',TMPDIR)
 scipad(p)
```

See Also

help

Authors

Serge Steer, INRIA

Name

make_index — creates a new index file for on-line help

```
make_index()
```

Description

The on-line help reads first the `index.html` file which contains the list of the chapters. This file comes with Scilab and is in the directory `SCIDIR/man/<language>` (see man). It is possible to change this index file while interactively adding new chapters. For that, modify the %helps variable and then use the `make_index` function.

See Also

%helps , man

Name

man — on line help XML file description format

Description

The on line help source files are written in XML.

Source files (with extension .xml) can be found in the <SCIDIR>/man/<language>/* directories. The file name is usually associated to a keyword (corresponding to a function name most of the cases) it describes.

A few words about XML

An XML file resembles to an HTML file but with both a more rigid and free syntax. Free because you may build your own tags: the set of tags together with its rules must be described somewhere, generally in another file (<SCIDIR>/man/manrev.dtd for scilab), and rigid because, once the tags and rules are defined (which are called the Definition Type Document: DTD) , you must respect its (in particular to every open tags <MY_TAG> must correspond a closed </MY_TAG>).

The DTD manrev.dtd is written in SGML and precises the exact syntax required by a scilab XML help page. So if you know this language you may red this file. The following annotated example (see the next section) shows you some possibilities offered by this DTD and may be enough to write simple help pages.

Once an XML page is written and conforms to the DTD, it may be transformed in HTML to be red by your favorite browser or by the tcltk scilab browser (see section browser choice in this page). The XML -> HTML translation is controled by a set of rules written in the (XML) file <SCIDIR>/man/language/html.xsl. Those rules are currently more or less restricted to fit the tcltk scilab browser features (which may display correctly only basic HTML): if you use a real HTML browser and want a better appearance you have to modify this file.

How to write a simple xml scilab help page: the lazzy way

If one want to write the xml file associated to a new scilab function he or she may use the Scilab function help_skeleton to produce the skeleton of the xml file. In most cases the user will not be required to know xml syntax.

How to write a simple xml scilab help page: an example

Here is a simple annotated XML scilab help page which describes an hypothetic foo scilab function. In the following, the XML file is displayed in a type writer font and cut-out in several parts, each part being preceded by some associated explanations. The entire XML file foo.xml is in the <SCIDIR>/man/eng/utility directory and the result may be displayed by clicking on foo. (you may found others examples in the <SCIDIR>/examples/man-examples-xml directory). **Finally** note that some tag pairs <TAG>, </TAG> have been renamed here <ATAG>, </ATAG>. This is because some scilab scripts which do some work on or from the xml files don't verify if a tag is inside a VERBATIM entry.

The 3 first lines of the file are mandatory, the second precises the path to the DTD file and the third, formed by the <MAN> tag, begin the hierarchical description (the file must finish with the </MAN> tag). The 4 followings entries : LANGUAGE, TITLE, TYPE and DATE, are also mandatory (in this order) the text corresponding to <TYPE> being generally 'Scilab function' (most of the cases) but may be simply 'Scilab keyword' or 'Scilab data type', ..., depending of what explains the help page.

```
<!DOCTYPE MAN SYSTEM "<SCIDIR>/man/manrev.dtd">
<MAN>
   <LANGUAGE>eng</LANGUAGE>
   <TITLE>foo</TITLE>
   <TYPE>Scilab function</TYPE>
   <DATE>$LastChangedDate$</DATE>
```

The first of these 2 following entries (SHORT_DESCRIPTION) is mandatory and important since the words of the short description text, are used by the apropos command to search help pages from a keyword: the short description is used to build the whatis.html file corresponding to your toolbox and the apropos keyword command looks in all the whatis files and then proposes the links to every page containing the word keyword in its short description (in fact the actual associated tags are <SHORT_DESCRIPTION> and </SHORT_DESCRIPTION> and not <ASHORT_DESCRIPTION> and </ASHORT_DESCRIPTION>). The next entry (CALLING_SEQUENCE) must be used if you describe a function (but is not strictly mandatory). If your function have several calling sequences use several CALLING_SEQUENCE_ITEM entries.

```
   <ASHORT_DESCRIPTION name="foo">foo short description</ASHORT_DESCRIPT
   <CALLING_SEQUENCE>
       <CALLING_SEQUENCE_ITEM>[y] = foo(x)</CALLING_SEQUENCE_ITEM>
   </CALLING_SEQUENCE>
```

The following entry (PARAM) is not strictly mandatory but is the good one to describe each parameters (input and output) in case of a function.

```
   <PARAM>
      <PARAM_INDENT>
         <PARAM_ITEM>
            <PARAM_NAME>x</PARAM_NAME>
            <PARAM_DESCRIPTION>
               <SP>: what may be x</SP>
            </PARAM_DESCRIPTION>
         </PARAM_ITEM>
         <PARAM_ITEM>
            <PARAM_NAME>y</PARAM_NAME>
            <PARAM_DESCRIPTION>
               <SP>: what may be y</SP>
            </PARAM_DESCRIPTION>
         </PARAM_ITEM>
      </PARAM_INDENT>
   </PARAM>
```

The DESCRIPTION entry is perhaps the most significant one (but not strictly mandatory) and may be more sophisticated than in this example (for instance you may have DESCRIPTION_ITEM sub-entries). Here you see how to write several paragraphs (each one enclosed between the <P> and </P> tags), how to emphasis a variable or a function name (by enclosing it between the <VERB> and </VERB> tags), how to emphasis a part of text (<EM> or <BD> and <TT> to put it in a type writer font)),

and finally, how to put a link onto another help page (in fact the actual associated tags are <LINK> and </LINK> and not <ALINK> and </ALINK>).

```
    <DESCRIPTION>
        <P>
           A first paragraph which explains what computes the foo function
           If you want to emphasis a parameter name then you use the follo
           tag <VERB>x</VERB>, if you want to emphasis a part of text
           <EM>inclose it inside theses tags</EM> and use theses ones
           <BD>to have a bold font</BD> and finally <TT>for a type writer
        </P>
        <P>
           A second paragraph... Here is an example of a link to another pa
           <ALINK>man</ALINK>.
        </P>
    </DESCRIPTION>
```

Here is how to write your own entry, for instance to describe some outside remarks and/or notes about your wonderful function.

```
    <SECTION label='Notes'>
        <P>
           Here is a list of notes :
        </P>
        <ITEM label='first'><SP>blablabla...</SP></ITEM>
        <ITEM label='second'><SP>toto is the french foo...</SP></ITEM>
    </SECTION>
```

An important entry is the EXAMPLE one which is reserved to show scilab uses of your function (begin with simple ones !). Note that you must close this entry with]]></EXAMPLE> and not like here with }}></EXAMPLE> (once again this is a bad trick to avoid some interpretation problems).

```
    <EXAMPLE><![CDATA[
        deff("y=foo(x)","y=x"); // define the foo function as the identity
        foo("toto")
 }}></EXAMPLE>
```

This last part explains how to put the links onto others related help pages (as said before the good tags are in fact <LINK> and </LINK> and not <ALINK> and </ALINK>) and finally how to reveal your name if you want (use one AUTHOR_ITEM entry by author). Perhaps it is a good idea to put an email adress if you look for bug reports !

```
    <SEE_ALSO>
      <SEE_ALSO_ITEM> <ALINK>man</ALINK> </SEE_ALSO_ITEM>
      <SEE_ALSO_ITEM> <ALINK>apropos</ALINK> </SEE_ALSO_ITEM>
```

```
  </SEE_ALSO>

  <AUTHOR>
    <AUTHOR_ITEM>B. P.</AUTHOR_ITEM>
  </AUTHOR>
</MAN>
```

How to create an help chapter

Create a directory and write down a set of xml files build as described above. Then start Scilab and execute `xmltohtml(dir)`, where `dir` is a character string giving the path of the directory (see xmltohtml for more details) .

How to make Scilab know a new help chapter

This can be done by the function add_help_chapter.

Examples

```
function y=foo(a,b,c),y=a+2*b+c,endfunction
path=help_skeleton('foo',TMPDIR)
scipad(path)
```

See Also

apropos , help , help_skeleton

Name

manedit — editing a manual item

```
manedit(manitem)
```

Parameters

manitem
: character string (usually, name of a function)

Description

`edit(manitem)` opens the xml file associated to `manitem` in the scipad editor.

If there is no xml file associated with `manitem` and `manitem` is the name of a Scilab function scipad opens with the skeleton of the xml file produced by help_skeleton. This file is located in TMPDIR.

Examples

```
manedit('manedit')

function [x,y,z]=foo123(a,b),
x=a+b,y=a-b,z=a==b
endfunction
manedit foo123
```

See Also

help , help_skeleton

Name

%helps — Variable defining the path of help directories

Description

The global variable `%helps` is an N x 2 matrix of strings. The kth row of `%helps`, `%helps(k,:)` represents the kth chapter of the manual and is made of two strings:

`%helps(k,1)` is the absolute pathname for a help directory.

`%helps(k,2)` is a title for this help directory. For instance, for k=2, we have the graphics chapter `%helps(2,:)`.

The variable `%helps` is defined in the Scilab startup file `SCI+"/scilab.start"`.

To add a new help directory, the user should add a row to the variable `%helps`. (One row for each directory).

For instance, `%helps=[%helps; "Path-Of-My-Help-Dir","My-Title"];` enables the Scilab help browser to look for help manual items in the directory with pathname "Path-Of-My-Help-Dir".

"My-Title" is then the title of a new help chapter.

A valid help directory must contain:

1- A set of `.html` files (e.g. `item1.html, item2.html` etc). The `.html` files are usually built from XML files.

2- A `whatis.html` file, which must have a special format. Each row of the `whatis` must be as follows:

```
<BR><A HREF="item.html">item</A> - quick description
```

`item` is the item of the help, i.e. the command `help item` displays the contents of the file `item.html`.

The command `apropos keyword` returns the row(s) of all the `whatis.html` file(s) in which the keyword appears.

On Linux platforms Scilab provides a Makefile for transforming `.xml` pages into `.html` pages (see SCIDIR/examples/man-examples).

See Also

apropos , help , man

Name

xmltohtml — converts xml Scilab help files to HTML format

```
xmltohtml(dirs [,titles [,dir_language [default_language]]]])
```

Parameters

dirs
: vector of strings: a set of directory paths for which html manuals are to be generated or `[]`

titles
: vector of strings: titles associated to directory paths or []

dir_language
: vector of strings: languages associated to directory paths or []

default_language
: vector of strings: default languages associated to directory paths or []. If an XML file is missing in the dir_language, it's copied from the default_language.

Description

converts xml Scilab help files contained in a set of directories into HTML files.

Examples

```
// example_1/
// `-- help
//     |-- en_US
//     |   |-- example_1_function_1.xml
//     |   |-- example_1_function_2.xml
//     |   `-- example_1_function_3.xml
//     `-- fr_FR
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml
//     `-- zh_TW
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml

my_module_path = pathconvert(SCI+'/modules/helptools/examples/example_1',%f,%f)

// Build the french help
// ============================================================================
my_french_help_dir    = my_module_path+'/help/fr_FR';
my_french_help_title  = 'Example 1 [fr_FR]';
my_french_html_dir    = xmltohtml(my_french_help_dir,my_french_help_title,'fr_F


// Build the english help
// ============================================================================
my_english_help_dir   = my_module_path+'/help/en_US';
```

```
my_english_help_title = 'Example 1 [en_US]';
my_english_html_dir   = xmltohtml(my_english_help_dir,my_english_help_title,'en


// Build the chinese help
// ======================================================================
my_chinese_help_dir   = my_module_path+'/help/zh_TW';
my_chinese_help_title = 'Example 1 [zh_TW]';
my_chinese_html_dir   = xmltohtml(my_chinese_help_dir,my_chinese_help_title,'zh
```

See Also

help , add_help_chapter

Name

xmltojar — converts xml Scilab help files to javaHelp format

```
xmltojar(dirs [,titles [,dir_language [default_language]]]])
```

Parameters

dirs
: vector of strings: a set of directory paths for which html manuals are to be generated or `[]`

titles
: vector of strings: titles associated to directory paths or []

dir_language
: vector of strings: languages associated to directory paths or []

default_language
: vector of strings: default languages associated to directory paths or []. If an XML file is missing in the dir_language, it's copied from the default_language.

Description

converts xml Scilab help files contained in a set of directories into jar files.

Examples

```
// example_1/
// `-- help
//     |-- en_US
//     |   |-- example_1_function_1.xml
//     |   |-- example_1_function_2.xml
//     |   `-- example_1_function_3.xml
//     `-- fr_FR
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml
//     `-- zh_TW
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml

my_module_path = pathconvert(SCI+'/modules/helptools/examples/example_1',%f,%f)

// Build the french help
// ============================================================================
my_french_help_dir    =  my_module_path+'/help/fr_FR';
my_french_help_title  =  'Example 1 [fr_FR]';
xmltojar(my_french_help_dir,my_french_help_title,'fr_FR');


// Build the english help
// ============================================================================
my_english_help_dir   =  my_module_path+'/help/en_US';
```

```
my_english_help_title =  'Example 1 [en_US]';
xmltojar(my_english_help_dir,my_english_help_title,'en_US');


// Build the chinese help
// ========================================================================
my_chinese_help_dir   =  my_module_path+'/help/zh_TW';
my_chinese_help_title =  'Example 1 [zh_TW]';
xmltojar(my_chinese_help_dir,my_chinese_help_title,'zh_TW');


// Add french, english or chinese help chapters
// ========================================================================

if getlanguage() == 'fr_FR' then
  add_help_chapter(my_french_help_title,my_module_path+"/jar");

elseif getlanguage() == 'zh_TW' then
  add_help_chapter(my_chinese_help_title,my_module_path+"/jar");

else
  add_help_chapter(my_english_help_title,my_module_path+"/jar");
end

// See the result in the help browser
// ========================================================================
help();

// Del french and english help chapters
// ========================================================================
if getlanguage() == 'fr_FR' then
  del_help_chapter(my_french_help_title);
else
  del_help_chapter(my_english_help_title);
end
```

See Also

help , add_help_chapter

Name

xmltopdf — converts xml Scilab help files to pdf format

```
xmltopdf(dirs [,titles [,dir_language [default_language]]]])
```

Parameters

dirs
: vector of strings: a set of directory paths for which pdf manuals are to be generated or `[]`

titles
: vector of strings: titles associated to directory paths or []

dir_language
: vector of strings: languages associated to directory paths or []

default_language
: vector of strings: default languages associated to directory paths or []. If an XML file is missing in the dir_language, it's copied from the default_language.

Description

converts xml Scilab help files contained in a set of directories into pdf files.

Examples

```
// example_1/
// `-- help
//     |-- en_US
//     |   |-- example_1_function_1.xml
//     |   |-- example_1_function_2.xml
//     |   `-- example_1_function_3.xml
//     `-- fr_FR
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml
//     `-- zh_TW
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml

my_module_path = pathconvert(SCI+'/modules/helptools/examples/example_1',%f,%f)

// Build the french help
// ============================================================================
my_french_help_dir    = my_module_path+'/help/fr_FR';
my_french_help_title  = 'Example 1 [fr_FR]';
my_french_pdf         = xmltopdf(my_french_help_dir,my_french_help_title,'fr_FR


// Build the english help
// ============================================================================
my_english_help_dir   = my_module_path+'/help/en_US';
```

```
my_english_help_title = 'Example 1 [en_US]';
my_english_pdf        = xmltopdf(my_english_help_dir,my_english_help_title,'en_

// Build the chinese help
// ========================================================================
my_chinese_help_dir   = my_module_path+'/help/zh_TW';
my_chinese_help_title = 'Example 1 [zh_TW]';
my_chinese_pdf        = xmltopdf(my_chinese_help_dir,my_chinese_help_title,'zh_
```

See Also

help , add_help_chapter

Name

xmltops — converts xml Scilab help files to postscript format

```
xmltops(dirs [,titles [,dir_language [default_language]]]])
```

Parameters

dirs
: vector of strings: a set of directory paths for which postscript manuals are to be generated or `[]`

titles
: vector of strings: titles associated to directory paths or []

dir_language
: vector of strings: languages associated to directory paths or []

default_language
: vector of strings: default languages associated to directory paths or []. If an XML file is missing in the dir_language, it's copied from the default_language.

Description

converts xml Scilab help files contained in a set of directories into ps files.

Examples

```
// example_1/
// `-- help
//     |-- en_US
//     |   |-- example_1_function_1.xml
//     |   |-- example_1_function_2.xml
//     |   `-- example_1_function_3.xml
//     `-- fr_FR
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml
//     `-- zh_TW
//         |-- example_1_function_1.xml
//         |-- example_1_function_2.xml
//         `-- example_1_function_3.xml



my_module_path = pathconvert(SCI+'/modules/helptools/examples/example_1',%f,%f)

// Build the french help
// ============================================================================
my_french_help_dir    = my_module_path+'/help/fr_FR';
my_french_help_title  = 'Example 1 [fr_FR]';
my_french_ps          = xmltops(my_french_help_dir,my_french_help_title,'fr_FR'


// Build the english help
```

```
// ============================================================================
my_english_help_dir   = my_module_path+'/help/en_US';
my_english_help_title = 'Example 1 [EN_US]';
my_english_ps         = xmltops(my_english_help_dir,my_english_help_title,'en_US

// Build the chinese help
// ============================================================================
my_chinese_help_dir   = my_module_path+'/help/zh_TW';
my_chinese_help_title = 'Example 1 [zh_TW]';
my_chinese_ps         = xmltops(my_chinese_help_dir,my_chinese_help_title,'zh_TW
```

See Also

help , add_help_chapter

Optimization and Simulation

Name

NDcost — generic external for optim computing gradient using finite differences

```
[f,g,ind]=NDcost(x,ind,fun,varargin)
```

Parameters

x
: real vector or matrix

ind
: integer parameter (see optim)

fun
: Scilab function with calling sequence `F=fun(x,varargin)` varargin may be use to pass parameters `p1,...pn`

f
: criterion value at point `x` (see optim)

g
: gradient value at point `x` (see optim)

Description

This function can be used as an external for `optim` to minimize problem where gradient is too complicated to be programmed. only the function `fun` which computes the criterion is required.

This function should be used as follow: `[f,xopt,gopt]=optim(list(NDcost,fun,p1,...pn),x0,...)`

Examples

```
// example #1 (a simple one)
//function to minimize
function f=rosenbrock(x,varargin)
   p=varargin(1)
   f=1+sum( p*(x(2:$)-x(1:$-1)^2)^2 + (1-x(2:$))^2)
endfunction

x0=[1;2;3;4];
[f,xopt,gopt]=optim(list(NDcost,rosenbrock,200),x0)

// example #2: This example (by Rainer von Seggern) shows a quick (*) way to
//             identify the parameters of a linear differential equation with
//             the help of scilab.
//             The model is a simple damped (linear) oscillator:
//
//               x''(t) + c x'(t) + k x(t) = 0 ,
//
// and we write it as a system of two differential equations of first
// order with y(1) = x, and y(2) = x':
//
//     dy1/dt = y(2)
//     dy2/dt = -c*y(2) -k*y(1).
//
// We suppose to have m measurements of x (that is y(1)) at different times
```

```
// t_obs(1), ..., t_obs(m) called x_obs(1), ..., x_obs(m)  (in this example
// these measuresments will be simulated), and we want to identify the paramete
// c and k by minimizing the sum of squared errors between x_obs and y1(t_obs,p
//
// (*) This method is not the most efficient but it is easy to implement.
//
function dy = DEQ(t,y,p)
   // The rhs of our first order differential equation system.
   c =p(1);k=p(2)
   dy=[y(2);-c*y(2)-k*y(1)]
endfunction

function y=uN(p, t, t0, y0)
   // Numerical solution obtained with ode. (In this linear case an exact analy
   // solution can easily be found, but ode would also work for "any" system.)
   // Note: the ode output must be an approximation of the solution at
   //       times given in the vector t=[t(1),...,t($)]
   y = ode(y0,t0,t,list(DEQ,p))
endfunction

function r = cost_func(p, t_obs, x_obs, t0, y0)
   // This is the function to be minimized, that is the sum of the squared
   // errors between what gives the model and the measuments.
   sol = uN(p, t_obs, t0, y0)
   e = sol(1,:) - x_obs
   r = sum(e.*e)
endfunction

// Data
y0 = [10;0]; t0 = 0; // Initial conditions y0 for initial time t0.
T = 30;  // Final time for the measurements.

// Here we simulate experimental data, (from which the parameters
// should be identified).
pe = [0.2;3];  // Exact parameters
m = 80;  t_obs = linspace(t0+2,T,m); // Observation times
// Noise: each measurement is supposed to have a (gaussian) random error
// of mean 0 and std deviation proportional to the magnitude
// of the value (sigma*|x_exact(t_obs(i))|).
sigma = 0.1;
y_exact = uN(pe, t_obs, t0, y0);
x_obs = y_exact(1,:) + grand(1,m,"nor",0, sigma).*abs(y_exact(1,:));

// Initial guess parameters
p0 = [0.5 ; 5];

// The value of the cost function before optimization:
cost0 = cost_func(p0, t_obs, x_obs, t0, y0);
mprintf("\n\r The value of the cost function before optimization = %g \n\r",...
                                                                    cost0)


// Solution with optim
[costopt,popt]=optim(list(NDcost,cost_func, t_obs, x_obs, t0, y0),p0,...
                                                    'ar',40,40,1e-3);

mprintf("\n\r The value of the cost function after optimization  = %g",costopt)
mprintf("\n\r The identified values of the parameters: c = %g, k = %g \n\r",...
                                                          popt(1),popt(2))
```

```
// A small plot:
t = linspace(0,T,400);
y = uN(popt, t, t0, y0);
clf();
plot2d(t',y(1,:)',style=5)
plot2d(t_obs',x_obs(1,:)',style=-5)
legend(["model","measurements"]);
xtitle("Least square fit to identify ode parameters")
```

See Also

optim, external, derivative

Name

bvode — boundary value problems for ODE

```
[z]=bvode(points,ncomp,m,aleft,aright,zeta,ipar,ltol,tol,fixpnt,...
fsub1,dfsub1,gsub1,dgsub1,guess1)
```

Parameters

z
: The solution of the ode evaluated on the mesh given by points

points
: an array which gives the points for which we want the solution

ncomp
: number of differential equations (ncomp <= 20)

m
: a vector of size `ncomp`. `m(j)` gives the order of the j-th differential equation

aleft
: left end of interval

aright
: right end of interval

zeta
: `zeta(j)` gives j-th side condition point (boundary point). must have

  ```
  zeta(j) <= zeta(j+1)
  ```

  all side condition points must be mesh points in all meshes used, see description of `ipar(11)` and `fixpnt` below.

ipar
: an integer array dimensioned at least 11. a list of the parameters in `ipar` and their meaning follows some parameters are renamed in bvode; these new names are given in parentheses.

  ipar(1)
  : 0 if the problem is linear, 1 if the problem is nonlinear

  ipar(2)
  : = number of collocation points per subinterval (= k) where

    ```
    max m(i) <= k <= 7 .
    ```

    if `ipar(2)=0` then bvode sets

    ```
    k = max ( max m(i)+1, 5-max m(i) )
    ```

  ipar(3)
  : = number of subintervals in the initial mesh (= n). if `ipar(3) = 0` then bvode arbitrarily sets `n = 5`.

  ipar(4)
  : = number of solution and derivative tolerances. (= ntol) we require

    ```
    0 < ntol <= mstar.
    ```

ipar(5)
= dimension of `fspace` (= ndimf) a real work array. its size provides a constraint on nmax. choose ipar(5) according to the formula:

```
ipar(5)>=nmax*nsizef
```

where

`nsizef=4+3*mstar+(5+kd)*kdm+(2*mstar-nrec)*2*mstar`.

ipar(6)
= dimension of ispace (= ndimi)an integer work array. its size provides a constraint on nmax, the maximum number of subintervals. choose `ipar(6)` according to the formula:

```
ipar(6)>=nmax*nsizei
```

where

`nsizei=3 + kdm` with `kdm=kd+mstar` ; `kd=k*ncomp` ; `nrec`=number of right end boundary conditions.

ipar(7)
output control (= iprint)

= -1
for full diagnostic printout

= 0
for selected printout

= 1
for no printout

ipar(8)
(= iread)

= 0
causes bvode to generate a uniform initial mesh.

= xx
Other values are not implemented yet in Scilab

= 1
if the initial mesh is provided by the user. it is defined in fspace as follows: the mesh

will occupy `fspace(1), ..., fspace(n+1)`. the user needs to supply only the interior mesh points `fspace(j) = x(j), j = 2, ..., n.`

= 2 if the initial mesh is supplied by the user
as with `ipar(8)=1`, and in addition no adaptive mesh selection is to be done.

ipar(9)
(= iguess)

= 0
if no initial guess for the solution is provided.

= 1
if an initial guess is provided by the user in subroutine `guess`.

= 2
if an initial mesh and approximate solution coefficients are provided by the user in fspace. (the former and new mesh are the same).

= 3
: if a former mesh and approximate solution coefficients are provided by the user in fspace, and the new mesh is to be taken twice as coarse; i.e.,every second point from the former mesh.

= 4
: if in addition to a former initial mesh and approximate solution coefficients, a new mesh is provided in fspace as well. (see description of output for further details on iguess = 2, 3, and 4.)

ipar(10)

= 0
: if the problem is regular

= 1
: if the first relax factor is =rstart, and the nonlinear iteration does not rely on past covergence (use for an extra sensitive nonlinear problem only).

= 2
: if we are to return immediately upon (a) two successive nonconvergences, or (b) after obtaining error estimate for the first time.

ipar(11)
: = number of fixed points in the mesh other than `aleft` and `aright`. (= nfxpnt , the dimension of `fixpnt`) the code requires that all side condition points other than `aleft` and `aright` (see description of zeta) be included as fixed points in `fixpnt`.

ltol
: an array of dimension `ipar(4)`. `ltol(j) = l` specifies that the j-th tolerance in tol controls the error in the l-th component of `z(u)`. also require that:

```
1 <= ltol(1) < ltol(2) < ... < ltol(ntol) <= mstar
```

tol
: an array of dimension `ipar(4)`. `tol(j)` is the error tolerance on the `ltol(j)` -th component of `z(u)`. thus, the code attempts to satisfy for `j=1:ntol` on each subinterval

$$\left|(z(v)-z(u))_{\mathrm{ltol}(j)}\right| \le \mathrm{tol}(j)\cdot\left|(z(u))_{\mathrm{ltol}(j)}\right|+\mathrm{tol}(j)$$

: if `v(x)` is the approximate solution vector.

fixpnt
: an array of dimension `ipar(11)`. it contains the points, other than `aleft` and `aright`, which are to be included in every mesh.

externals
: The function `fsub,dfsub,gsub,dgsub,guess` are Scilab externals i.e. functions (see syntax below) or the name of a Fortran subroutine (character string) with specified calling sequence or a list. An external as a character string refers to the name of a Fortran subroutine. The Fortran coded function interface to bvode are specified in the file `fcol.f`.

fsub
: name of subroutine for evaluating

$$f(x,z(u(x))) = \left(f_1,\cdots,f_{\mathrm{ncomp}}\right)^t$$

: at a point x in `(aleft,aright)`. it should have the heading `[f]=fsub(x,z)` where `f` is the vector containing the value of `fi(x,z(u))` in the i-th component and

$$z(u(x)) = (z(1), \cdots, z(\text{mstar}))^t$$

is defined as above under purpose .

dfsub
: name of subroutine for evaluating the Jacobian of `f(x,z(u))` at a point x. it should have the heading `[df]=dfsub (x , z )` where `z(u(x))` is defined as for `fsub` and the (`ncomp`) by (`mstar`) array df should be filled by the partial derivatives of f, viz, for a particular call one calculates

$$\text{df}(i,j) = \frac{\text{dfi}}{\text{dzj}}, \begin{matrix} i = 1, \cdots, \text{ncmp} \\ j = 1, \cdots, \text{mstar} \end{matrix}$$

gsub
: name of subroutine for evaluating the i-th component of `g(x,z(u(x))) = g (zeta(i),z(u(zeta(i))))` at a point `x = zeta(i)` where `1<=i<=mstar.`

  it should have the heading`[g]=gsub(i, z)` where `z(u)` is as for `fsub`, and `i` and `g=gi` are as above. Note that in contrast to `f` in `fsub` , here only one value per call is returned in `g`.

dgsub
: name of subroutine for evaluating the i-th row of the Jacobian of `g(x,u(x))`. it should have the heading `[dg]=dgsub(i, z)` where `z(u)` is as for fsub, i as for gsub and the mstar-vector `dg` should be filled with the partial derivatives of g, viz, for a particular call one calculates

guess
: name of subroutine to evaluate the initial approximation for `z(u(x))` and for `dmval(u(x))`= vector of the mj-th derivatives of `u(x)`. it should have the heading `[z,dmval]= guess (x )` note that this subroutine is used only if `ipar(9) = 1`, and then all `mstar` components of z and ncomp components of dmval should be specified for any x,

  `aleft <= x <= aright .`

Description

this package solves a multi-point boundary value problem for a mixed order system of ode-s given by

$$u_i^{(m(i))} = f_i(x; z(u(x))), \begin{matrix} i = 1, \cdots, \text{ncomp} \\ \text{aleft} < x < \text{aright} \end{matrix}$$

$$g_j(\text{zeta}(j); z(u(\text{zeta}(j)))) = 1, \begin{matrix} j = 1, \cdots, \text{mstar} \\ \text{mstar} = m(1) + m(2) + \cdots + m(\text{ncomp}) \end{matrix}$$

where

$$u = (u_1, u_2, \cdots, u_{\text{ncomp}})^t$$

is the exact solution vector

$$u_i^{(m)} \text{is the} m = m(i)\text{th derivative of} u_i$$

$$z(u(x)) = \left(u_1(x), u_1^{(1)}(x), \cdots, u_1^{(m1-1)}(x), \cdots, u_{\text{ncomp}}^{(m\text{ncomp}-1)}(x)\right)$$

$$f_i(x,z(u))$$

is a (generally) nonlinear function of `z(u)=z(u(x))`.

$$g_j(\text{zeta}(j);z(u))$$

is a (generally) nonlinear function used to represent a boundary condition.

the boundary points satisfy

`aleft <= zeta(1) <= .. <= zeta(mstar) <= aright`.

the orders `mi` of the differential equations satisfy

`1<=m(i)<=4`.

Examples

```
deff('df=dfsub(x,z)','df=[0,0,-6/x**2,-6/x]')
deff('f=fsub(x,z)','f=(1 -6*x**2*z(4)-6*x*z(3))/x**3')
deff('g=gsub(i,z)','g=[z(1),z(3),z(1),z(3)];g=g(i)')
deff('dg=dgsub(i,z)',['dg=[1,0,0,0;0,0,1,0;1,0,0,0;0,0,1,0]';
                      'dg=dg(i,:)'])
deff('[z,mpar]=guess(x)','z=0;mpar=0')// unused here

 //define trusol for testing purposes
deff('u=trusol(x)',['u=0*ones(4,1)';
                    'u(1) =  0.25*(10*log(2)-3)*(1-x) + 0.5 *( 1/x   + (3+x)*lo
                    'u(2) = -0.25*(10*log(2)-3)       + 0.5 *(-1/x^2 + (3+x)/x
                    'u(3) = 0.5*( 2/x^3 + 1/x   - 3/x^2)'
                    'u(4) = 0.5*(-6/x^4 - 1/x/x + 6/x^3)'])

fixpnt=0;m=4;
ncomp=1;aleft=1;aright=2;
zeta=[1,1,2,2];
ipar=zeros(1,11);
ipar(3)=1;ipar(4)=2;ipar(5)=2000;ipar(6)=200;ipar(7)=1;
ltol=[1,3];tol=[1.e-11,1.e-11];
res=aleft:0.1:aright;

z=bvode(res,ncomp,m,aleft,aright,zeta,ipar,ltol,tol,fixpnt,...
        fsub,dfsub,gsub,dgsub,guess)

z1=[];for x=res,z1=[z1,trusol(x)]; end;
z-z1
```

See Also

fort, link, external, ode, dassl

Authors

u. ascher, department of computer science, university of british; columbia, vancouver, b. c., canada v6t 1w5; g. bader, institut f. angewandte mathematik university of heidelberg; im neuenheimer feld 294d-6900 heidelberg 1 ; ; Fortran subroutine colnew.f

Name

bvodeS — simplified call of bvode

```
z=bvodeS(x,m,n,a,b,fsub,gsub,zeta, [ystart,dfsub,dgsub,fixpnt,ndimf,ndimi,ltol,
```

Parameters

x
: array of points at which approximations of the solution will be computed. The points x(i) must be given in increasing order.

m
: array of orders for the differential equations given in `fsub`.

n
: number of differential equations, length of `m`.

a
: left end of solution interval, where a<=x(1).

b
: right end of solution interval, where x($)<=b.

fsub
: name of a Scilab function for evaluating the rhs of the differential equation system to be solved; fsub also may be a list for parameter transfer.

gsub
: name of a Scilab function for evaluating the boundary conditions for the given differential equation system; gsub may also be a list as for fsub.

zeta
: array of points at which the boundary or side conditions are given in increasing order. Each point must occur as often as boundary or side conditions are given there. Each side condition point other than a or b must be included in the array fixpnt. The length of zeta should be sum(m).

z
: array of all derivatives of the solution functions up to the order m(i)-1 for the i-th function. (See the examples below.) The length of z should be sum(m).

ystart,dfsub,dgsub,fixpnt,ndimf,ndimi,ltol,tol,ntol,nonlin, collpnt,subint,iprint,ireg,ifail
: These optional arguments may be called by name in any order in the form argument=name. The meaning of ystart to ireg is given in the bvode help page. The Scilab functions `dfsub` and `dgsub` for evaluating the Jacobians may also be lists for parameter transfer. The function `ystart` is called `guess` in bvode.

ifail
: if ifail=1, all parameters needed for the call of bvode are displayed.

Description

This interface program simplifies the call of bvode, a program for the numerical solution of multi-point boundary value problems for mixed order systems of ordinary differential equations. The Scilab program bvode is adapted from the fortran program colnew. See the paper by U. Asher, J. Christiansen, and R.D. Russel: Collocation software for boundary-value ODE's, ACM Trans. Math. Soft. 7:209-222, 1981. The following examples should demonstrate not only how such systems can be solved with the help of bvodeS, but also should emphazise some important problems which occur with boundary value problems for ordinary ode's.

Examples

```
// 1. Modified example from help bvode.

// DE:  y1''''(x)=(1-6*x*x*y1'''(x)-6*x*y1''(x))/(y2(x)^3)
//         y2'(x)=1
// z=[y1 ; y1' ; y1'' ; y1''' ; y2]
// BV: y1(1)=0 y1''(1)=0
// BV: y1(2)=1 y1''(2)=0 y2(2)=2

function RhS=fsub(x,z)
  RhS=[(1-6*x*x*z(4)-6*x*z(3))/(z(5)^3);1]
endfunction

function g=gsub(i,z)
  g=[z(1) z(3) z(1)-1 z(3) z(5)-2]
  g=g(i)
endfunction

function [z,lhS]=ystart(x)
  z=zeros(5,1);z(5)=1;
  lhS=[0;1];
endfunction

n=2;
m=[4 1];
N=100;
a=1; b=2;
zeta=[a a b b b];
x=linspace(a,b,N);

ltol=4; // We want to change the default error for y1'''.
tol=1e-12;
tic()
z=bvodeS(x,m,n,a,b,fsub,gsub,zeta,ltol=ltol,tol=tol,ystart=ystart);
// Try tol=1e-14 etc.
toc()

function z=yex(x) // True solution
  z=zeros(5,1);
  z(1)=0.25*(10*log(2)-3)*(1-x)+0.5*(1/x+(3+x)*log(x)-x)+(x-1)
  z(2)=-0.25*(10*log(2)-3)+0.5*(-1/x^2+(3+x)/x+log(x)-1)+1
  z(3)=0.5*(2/x^3+1/x-3/x^2)
  z(4)=0.5*(-6/x^4-1/x/x+6/x^3)
  z(5)=x
endfunction

zex=[];for xx=x, zex=[zex yex(xx)]; end
scf(0); clf();
plot2d(x,abs(z-zex)',style=[1 2 3 5 6])
xtitle('Absolute error','x',' ')
legend(['z1(x)';'z2(x)';'z3(x)';'z4(x)';'z5(x)'])

// example #2. An eigenvalue problem

// y''(x)=-la*y(x)
```

```
// BV: y(0)=y'(0); y(1)=0
// Eigenfunctions and eigenvalues are y(x,n)=sin(s(n)*(1-x)), la(n)=s(n)^2,
// where s(n) are the zeros of f(s,n)=s+atan(s)-(n+1)*pi, n=0,1,2,...
// To get a third boundary condition, we choose y(0)=1
// (With y(x) also c*y(x) is a solution for each constant c.)
// We solve the following ode system:
// y''=-la*y
// la'=0
// BV: y(0)=y'(0), y(0)=1; y(1)=0
// z=[y(x) ; y'(x) ; la]

function rhs=fsub(x,z)
  rhs=[-z(3)*z(1);0]
endfunction

function g=gsub(i,z)
  g=[z(1)-z(2) z(1)-1 z(1)]
  g=g(i)
endfunction

// The following start function is good for the first 8 eigenfunctions.
function [z,lhs]=ystart(x,z,la0)
  z=[1;0;la0]
  lhs=[0;0]
endfunction

a=0;b=1;
m=[2;1];
n=2;
zeta=[a a b];
N=101;
x=linspace(a,b,N)';

// We have s(n)-(n+1/2)*pi -> 0 for n to infinity.
la0=input('n-th eigenvalue: n= ?');la0=(%pi/2+la0*%pi)^2;

z=bvodeS(x,m,n,a,b,fsub,gsub,zeta,ystart=list(ystart,la0));

clf()
plot2d(x,[z(1,:)' z(2,:)'],style=[5 1],axesflag=5)
xtitle(['Startvalue =  '+string(la0);'Eigenvalue = '+string(z(3,1))],'x',' ')
legend(['y(x)';'y''(x)'])


// example #3. A boundary value problem with more than one solution.

// DE: y''(x)=-exp(y(x))
// BV: y(0)=0; y(1)=0
// This boundary value problem has more than one solution.
// It is demonstrated how to find two of them with the help of
// some preinformation of the solutions y(x) to build the function ystart.
// z=[y(x);y'(x)]

a=0;b=1;m=2;n=1;
zeta=[a b];
N=101;
tol=1e-8*[1 1];
x=linspace(a,b,N);
```

```
function rhs=fsub(x,z),rhs=-exp(z(1));endfunction

function g=gsub(i,z)
  g=[z(1) z(1)]
  g=g(i)
endfunction

function [z,lhs]=ystart(x,z,M)
  //z=[4*x*(1-x)*M ; 4*(1-2*x)*M]
  z=[M;0]
  //lhs=[-exp(4*x*(1-x)*M)]
  lhs=0
endfunction

for M=[1 4]
   if M==1
      z=bvodeS(x,m,n,a,b,fsub,gsub,zeta,ystart=list(ystart,M),tol=tol);
   else
      z1=bvodeS(x,m,n,a,b,fsub,gsub,zeta,ystart=list(ystart,M),tol=tol);
   end
end

// Integrating the ode yield e.g. the two solutions yex and yex1.

function y=f(c),y=c.*(1-tanh(sqrt(c)/4).^2)-2;endfunction
c=fsolve(2,f);

function y=yex(x,c)
  y=log(c/2*(1-tanh(sqrt(c)*(1/4-x/2)).^2))
endfunction

function y=f1(c1), y=2*c1^2+tanh(1/4/c1)^2-1;endfunction
c1=fsolve(0.1,f1);

function y=yex1(x,c1)
  y=log((1-tanh((2*x-1)/4/c1).^2)/2/c1/c1)
endfunction

disp(norm(z(1,:)-yex(x)),'norm(yex(x)-z(1,:))= ')
disp(norm(z1(1,:)-yex1(x)),'norm(yex1(x)-z1(1,:))= ')

clf();
subplot(2,1,1)
plot2d(x,z(1,:),style=[5])
xtitle('Two different solutions','x',' ')
subplot(2,1,2)
plot2d(x,z1(1,:),style=[5])
xtitle(' ','x',' ')


// example #4. A multi-point boundary value problem.

// DE y'''(x)=1
// z=[y(x);y'(x);y''(x)]
// BV: y(-1)=2 y(1)=2
// Side condition: y(0)=1
```

```
a=-1;b=1;c=0;
// The side condition point c must be included in the array fixpnt.
n=1;
m=[3];

function rhs=fsub(x,z)
  rhs=1
endfunction

function g=gsub(i,z)
  g=[z(1)-2 z(1)-1 z(1)-2]
  g=g(i)
endfunction

N=10;
zeta=[a c b];
x=linspace(a,b,N);

z=bvodeS(x,m,n,a,b,fsub,gsub,zeta,fixpnt=c);

function y=yex(x)
y=x.^3/6+x.^2-x./6+1
endfunction

disp(norm(yex(x)-z(1,:)),'norm(yex(x)-z(1,:))= ')
```

See Also

bvode, ode, dassl

Authors

Rainer von Seggern

Name

datafit — Parameter identification based on measured data

```
[p,err]=datafit([imp,] G [,DG],Z [,W],[contr],p0,[algo],[df0,[mem]],
[work],[stop],['in'])
```

Parameters

imp
: scalar argument used to set the trace mode. `imp=0` nothing (execpt errors) is reported, `imp=1` initial and final reports, `imp=2` adds a report per iteration, `imp>2` add reports on linear search. Warning, most of these reports are written on the Scilab standard output.

G
: function descriptor (e=G(p,z), e: ne x 1, p: np x 1, z: nz x 1)

DG
: partial of G wrt p function descriptor (optional; S=DG(p,z), S: ne x np)

Z
: matrix [z_1,z_2,...z_n] where z_i (nz x 1) is the ith measurement

W
: weighting matrix of size ne x ne (optional; defaut no ponderation)

contr
: `'b',binf,bsup` with `binf` and `bsup` real vectors with same dimension as `p0`. `binf` and `bsup` are lower and upper bounds on `p`.

p0
: initial guess (size np x 1)

algo
: `'qn'` or `'gc'` or `'nd'` . This string stands for quasi-Newton (default), conjugate gradient or non-differentiable respectively. Note that `'nd'` does not accept bounds on `x`).

df0
: real scalar. Guessed decreasing of `f` at first iteration. (`df0=1` is the default value).

mem :
: integer, number of variables used to approximate the Hessian, (`algo='gc' or 'nd'`). Default value is around 6.

stop
: sequence of optional parameters controlling the convergence of the algorithm. `stop= 'ar',nap, [iter [,epsg [,epsf [,epsx]]]]`

  "ar"
  : reserved keyword for stopping rule selection defined as follows:

  nap
  : maximum number of calls to `fun` allowed.

  iter
  : maximum number of iterations allowed.

  epsg
  : threshold on gradient norm.

epsf
: threshold controlling decreasing of `f`

epsx
: threshold controlling variation of `x`. This vector (possibly matrix) of same size as `x0` can be used to scale `x`.

"in"
: reserved keyword for initialization of parameters used when `fun` in given as a Fortran routine (see below).

p
: Column vector, optimal solution found

err
: scalar, least square error.

Description

`datafit` is used for fitting data to a model. For a given function `G(p,z)`, this function finds the best vector of parameters `p` for approximating `G(p,z_i)=0` for a set of measurement vectors `z_i`. Vector `p` is found by minimizing `G(p,z_1)'WG(p,z_1)+G(p,z_2)'WG(p,z_2)+...+G(p,z_n)'WG(p,z_n)`

`datafit` is an improved version of `fit_dat`.

Examples

```
//generate the data
function y=FF(x,p),y=p(1)*(x-p(2))+p(3)*x.*x,endfunction
X=[];Y=[];
pg=[34;12;14] //parameter used to generate data
for x=0:.1:3, Y=[Y,FF(x,pg)+100*(rand()-.5)];X=[X,x];end
Z=[Y;X];


//The criterion function
function e=G(p,z),
  y=z(1),x=z(2);
  e=y-FF(x,p),
endfunction

//Solve the problem
p0=[3;5;10]
[p,err]=datafit(G,Z,p0);

scf(0);clf()
plot2d(X,FF(X,pg),5) //the curve without noise
plot2d(X,Y,-1)  // the noisy data
plot2d(X,FF(X,p),12) //the solution

//the gradient of the criterion function
function s=DG(p,z),
  a=p(1),b=p(2),c=p(3),y=z(1),x=z(2),
  s=-[x-b,-a,x*x]
endfunction
```

```
[p,err]=datafit(G,DG,Z,p0);
scf(1);clf()
plot2d(X,FF(X,pg),5) //the curve without noise
plot2d(X,Y,-1)  // the noisy data
plot2d(X,FF(X,p),12) //the solution

// Add some bounds on the estimate of the parameters
// We want positive estimation (the result will not change)
[p,err]=datafit(G,DG,Z,'b',[0;0;0],[%inf;%inf;%inf],p0,algo='gc');
scf(1);clf()
plot2d(X,FF(X,pg),5) //the curve without noise
plot2d(X,Y,-1)  // the noisy data
plot2d(X,FF(X,p),12) //the solution
```

See Also

lsqrsolve, optim, leastsq

Name

derivative — approximate derivatives of a function

```
derivative(F,x)
[J [,H]] = derivative(F,x [,h ,order ,H_form ,Q])
```

Parameters

F

a Scilab function F: `R^n --> R^m` or a `list(F,p1,...,pk)`, where F is a scilab function in the form `y=F(x,p1,...,pk)`, p1, ..., pk being any scilab objects (matrices, lists,...).

x

real column vector of dimension n.

h

(optional) real, the stepsize used in the finite difference approximations.

order

(optional) integer, the order of the finite difference formula used to approximate the derivatives (order = 1,2 or 4, default is order=2).

H_form

(optional) string, the form in which the Hessean will be returned. Possible forms are:

H_form='default'

H is a m x (`n^2`) matrix ; in this form, the k-th row of H corresponds to the Hessean of the k-th component of F, given as the following row vector :

$$\left(\frac{d(grad(F_k))}{dx_1}, \cdots, \frac{d(grad(F_k))}{dx_n} \right)$$

((grad(F_k) being a row vector).

H_form='blockmat' :

H is a (mxn) x n block matrix : the classic Hessean matrices (of each component of F) are stacked by row (H = [H1 ; H2 ; ... ; Hm] in scilab syntax).

H_form='hypermat' :

H is a n x n matrix for m=1, and a n x n x m hypermatrix otherwise. H(:,:,k) is the classic Hessean matrix of the k-th component of F.

Q

(optional) real matrix, orthogonal (default is eye(n,n)).

Description

Numerical approximation of the first and second derivatives of a function F: `R^n --> R^m` at the point x. The Jacobian is computed by approximating the directional derivatives of the components of F in the direction of the columns of Q. (For m=1, v=Q(:,k) : grad(F(x))*v = Dv(F(x)).) The second derivatives are computed by composition of first order derivatives. If H is given in its default form the Taylor series of F(x) up to terms of second order is given by :

$$F(x+dx) = F(x) + J(x) \cdot dx + \frac{1}{2} \cdot dx^t \cdot H(x) \cdot dx + \cdots$$

(([J,H]=derivative(F,x,H_form='default'), J=J(x), H=H(x).)

Remarks

Numerical approximation of derivatives is generally an unstable process. The step size h must be small to get a low error but if it is too small floating point errors will dominate by cancellation. As a rule of thumb don't change the default step size. To work around numerical difficulties one may also change the order and/or choose different orthogonal matrices Q (the default is eye(n,n)), especially if the approximate derivatives are used in optimization routines. All the optional arguments may also be passed as named arguments, so that one can use calls in the form :

```
derivative(F, x, H_form = "hypermat")
derivative(F, x, order = 4) etc.
```

Examples

```
function y=F(x)
  y=[sin(x(1)*x(2))+exp(x(2)*x(3)+x(1)) ; sum(x.^3)];
endfunction

function y=G(x,p)
  y=[sin(x(1)*x(2)*p)+exp(x(2)*x(3)+x(1)) ; sum(x.^3)];
endfunction

x=[1;2;3];[J,H]=derivative(F,x,H_form='blockmat')

n=3;
// form an orthogonal matrix :
nu=0; while nu<n, [Q,nu]=colcomp(rand(n,n)); end
for i=[1,2,4]
  [J,H]=derivative(F,x,order=i,H_form='blockmat',Q=Q);
  mprintf("order= %d \n",i);
  H,
end

p=1;h=1e-3;
[J,H]=derivative(list(G,p),x,h,2,H_form='hypermat');H
[J,H]=derivative(list(G,p),x,h,4,Q=Q);H

// Taylor series example:
dx=1e-3*[1;1;-1];
[J,H]=derivative(F,x);
F(x+dx)
F(x+dx)-F(x)
F(x+dx)-F(x)-J*dx
F(x+dx)-F(x)-J*dx-1/2*H*(dx .*. dx)

// A trivial example
function y=f(x,A,p,w), y=x'*A*x+p'*x+w; endfunction
// with Jacobian and Hessean given by J(x)=x'*(A+A')+p', and H(x)=A+A'.
A = rand(3,3); p = rand(3,1); w = 1;
x = rand(3,1);
[J,H]=derivative(list(f,A,p,w),x,h=1,H_form='blockmat')

// Since f(x) is quadratic in x, approximate derivatives of order=2 or 4 by fin
// differences should be exact for all h~=0. The apparent errors are caused by
```

```
// cancellation in the floating point operations, so a "big" h is choosen.
// Comparison with the exact matrices:
Je = x'*(A+A')+p'
He = A+A'
clean(Je - J)
clean(He - H)
```

See Also

numdiff, derivat

Authors

Rainer von Seggern, Bruno Pincon

Name

fit_dat — Parameter identification based on measured data

```
[p,err]=fit_dat(G,p0,Z [,W] [,pmin,pmax] [,DG])
```

Parameters

G
: Scilab function (e=G(p,z), e: nex1, p: npx1, z: nzx1)

p0
: initial guess (size npx1)

Z
: matrix [z_1,z_2,...z_n] where z_i (nzx1) is the ith measurement

W
: weighting matrix of size nexne (optional; default 1)

pmin
: lower bound on p (optional; size npx1)

pmax
: upper bound on p (optional; size npx1)

DG
: partial of G wrt p (optional; S=DG(p,z), S: nexnp)

Description

`fit_dat` is used for fitting data to a model. For a given function G(p,z), this function finds the best vector of parameters p for approximating G(p,z_i)=0 for a set of measurement vectors z_i. Vector p is found by minimizing `G(p,z_1)'WG(p,z_1)+G(p,z_2)'WG(p,z_2)+...+G(p,z_n)'WG(p,z_n)`

Examples

```
deff('y=FF(x)','y=a*(x-b)+c*x.*x')
X=[];Y=[];
a=34;b=12;c=14;for x=0:.1:3, Y=[Y,FF(x)+100*(rand()-.5)];X=[X,x];end
Z=[Y;X];
deff('e=G(p,z)','a=p(1),b=p(2),c=p(3),y=z(1),x=z(2),e=y-FF(x)')

[p,err]=fit_dat(G,[3;5;10],Z)

xset('window',0)
xbasc();
plot2d(X',Y',-1)
plot2d(X',FF(X)',5,'002')
a=p(1),b=p(2),c=p(3);plot2d(X',FF(X)',12,'002')

a=34;b=12;c=14;
deff('s=DG(p,z)','y=z(1),x=z(2),s=-[x-p(2),-p(1),x*x]')

[p,err]=fit_dat(G,[3;5;10],Z,DG)
```

```
xset('window',1)
xbasc();
plot2d(X',Y',-1)
plot2d(X',FF(X)',5,'002')
a=p(1),b=p(2),c=p(3);plot2d(X',FF(X)',12,'002')
```

See Also

optim, datafit

Name

fsolve — find a zero of a system of n nonlinear functions

```
[x [,v [,info]]]=fsolve(x0,fct [,fjac] [,tol])
```

Parameters

x0
: real vector (initial value of function argument).

fct
: external (i.e function or list or string).

fjac
: external (i.e function or list or string).

tol
: real scalar. precision tolerance: termination occurs when the algorithm estimates that the relative error between x and the solution is at most tol. (`tol=1.d-10` is the default value).

x :
: real vector (final value of function argument, estimated zero).

v :
: real vector (value of function at x).

info
: termination indicator

  0
  : improper input parameters.

  1
  : algorithm estimates that the relative error between x and the solution is at most tol.

  2
  : number of calls to fcn reached

  3
  : tol is too small. No further improvement in the approximate solution x is possible.

  4
  : iteration is not making good progress.

Description

find a zero of a system of n nonlinear functions in n variables by a modification of the powell hybrid method. Jacobian may be provided.

```
0 = fct(x) w.r.t x.
```

`fct` is an "external". This external returns `v=fct(x)` given `x`.

The simplest calling sequence for `fct` is:

```
[v]=fct(x).
```

If `fct` is a character string, it refers to a C or Fortran routine which must be linked to Scilab. Fortran calling sequence must be

```
fct(n,x,v,iflag)
integer n,iflag
double precision x(n),v(n)
```

and C Calling sequence must be

```
fct(int *n, double x[],double v[],int *iflag)
```

Incremental link is possible (help `link`).

`jac` is an "external". This external returns `v=d(fct)/dx (x)` given `x`.

The simplest calling sequence for `jac` is:

```
[v]=jac(x).
```

If `jac` is a character string, it refers to a to a C or Fortran routine which must be linked to Scilab calling sequences are the same as those for fct. Note however that v must be a nxn array.

Examples

```
// A simple example with fsolve
a=[1,7;2,8];b=[10;11];

deff('[y]=fsol1(x)','y=a*x+b');
deff('[y]=fsolj1(x)','y=a');

[xres]=fsolve([100;100],fsol1);
a*xres+b

[xres]=fsolve([100;100],fsol1,fsolj1);
a*xres+b

// See routines/default/Ex-fsolve.f
[xres]=fsolve([100;100],'fsol1','fsolj1',1.e-7);
a*xres+b
```

See Also

external, qpsolve, linpro, optim

Name

karmarkar — karmarkar algorithm

```
[x1]=karmarkar(a,b,c,x0)
```

Parameters

a
: matrix (n,p)

b
: n - vector

c
: p - vector

x0
: initial vector

eps
: threshold (default value : 1.d-5)

gamma
: descent step `0<gamma<1` , default value : 1/4

x1
: solution

crit
: value of c'*x1

Description

Computes `x` which minimizes

$$\min c^t \cdot x$$
$$\text{with} a \cdot x = b$$
$$\text{and} x \geq 0$$

Examples

```
// n=10;p=20;
// a=rand(n,p);c=rand(p,1);x0=abs(rand(p,1));b=a*x0;x1=karmarkar(a,b,c,x0);
```

Name

leastsq — Solves non-linear least squares problems

```
[fopt,[xopt,[grdopt]]]=leastsq(fun, x0)
[fopt,[xopt,[grdopt]]]=leastsq(fun, dfun, x0)
[fopt,[xopt,[grdopt]]]=leastsq(fun, cstr, x0)
[fopt,[xopt,[grdopt]]]=leastsq(fun, dfun, cstr, x0)
[fopt,[xopt,[grdopt]]]=leastsq(fun, dfun, cstr, x0, algo)
[fopt,[xopt,[grdopt]]]=leastsq([imp], fun [,dfun] [,cstr],x0 [,algo],[df0,[mem]
```

Parameters

fopt
: value of the function *f(x)=||fun(x)||^2* at `xopt`

xopt
: best value of `x` found to minimize *||fun(x)||^2*

grdopt
: gradient of *f* at `xopt`

fun
: a scilab function or a list defining a function from *R^n* to *R^m* (see more details in DESCRIPTION).

x0
: real vector (initial guess of the variable to be minimized).

dfun
: a scilab function or a string defining the Jacobian matrix of `fun` (see more details in DESCRIPTION).

cstr
: bound constraints on `x`. They must be introduced by the string keyword `'b'` followed by the lower bound `binf` then by the upper bound `bsup` (so `cstr` appears as `'b',binf,bsup` in the calling sequence). Those bounds are real vectors with same dimension than `x0` (-%inf and +%inf may be used for dimension which are unrestricted).

algo
: a string with possible values: `'qn'` or `'gc'` or `'nd'`. These strings stand for quasi-Newton (default), conjugate gradient or non-differentiable respectively. Note that `'nd'` does not accept bounds on `x`.

imp
: scalar argument used to set the trace mode. `imp=0` nothing (except errors) is reported, `imp=1` initial and final reports, `imp=2` adds a report per iteration, `imp>2` add reports on linear search. Warning, most of these reports are written on the Scilab standard output.

df0
: real scalar. Guessed decreasing of *||fun||^2* at first iteration. (`df0=1` is the default value).

mem
: integer, number of variables used to approximate the Hessean (second derivatives) of *f* when `algo='qn'`. Default value is around 6.

stop
: sequence of optional parameters controlling the convergence of the algorithm. They are introduced by the keyword `'ar'`, the sequence being of the form `'ar',nap, [iter [,epsg [,epsf [,epsx]]]]`

nap
: maximum number of calls to `fun` allowed.

iter
: maximum number of iterations allowed.

epsg
: threshold on gradient norm.

epsf
: threshold controlling decreasing of `f`

epsx
: threshold controlling variation of `x`. This vector (possibly matrix) of same size as `x0` can be used to scale `x`.

Description

fun being a function from *R^n* to *R^m* this routine tries to minimize w.r.t. *x*, the function:

$$f(x) = \| \mathrm{fun}(x) \|^2 = \sum_{i=1}^{m} \mathrm{fun}_i^2(x)$$

which is the sum of the squares of the components of *fun*. Bound constraints may be imposed on `x`.

How to provide fun and dfun

`fun` can be either a usual scilab function (case 1) or a fortran or a C routine linked to scilab (case 2). For most problems the definition of *fun* will need supplementary parameters and this can be done in both cases.

case 1:
: when `fun` is a Scilab function, its calling sequence must be: `y=fun(x [,opt_par1,opt_par2,...])`. When `fun` needs optional parameters it must appear as `list(fun,opt_par1,opt_par2,...)` in the calling sequence of `leastsq`.

case 2:
: when `fun` is defined by a Fortran or C routine it must appear as `list(fun_name,m [,opt_par1,opt_par2,...])` in the calling sequence of `leastsq`, `fun_name` (a string) being the name of the routine which must be linked to Scilab (see link). The generic calling sequences for this routine are:

```
In Fortran:    subroutine fun(m, n, x, params, y)
               integer m,n
               double precision x(n), params(*), y(m)

In C:          void fun(int *m, int *n, double *x, double *params, double *y
```

: where `n` is the dimension of vector `x`, `m` the dimension of vector `y` (which must store the evaluation of *fun* at *x*) and `params` is a vector which contains the optional parameters *opt_par1*, *opt_par2*, ... (each parameter may be a vector, for instance if *opt_par1* has 3 components, the description of *opt_par2* begin from `params(4)` (fortran case), and from `params[3]` (C case), etc... Note that even if `fun` doesn't need supplementary parameters you must anyway write the fortran code with a `params` argument (which is then unused in the subroutine core).

In many cases it is adviced to provide the Jacobian matrix `dfun` (*dfun(i,j)=dfi/dxj*) to the optimizer (which uses a finite difference approximation otherwise) and as for `fun` it may be given as a usual scilab function or as a fortran or a C routine linked to scilab.

case 1:
when `dfun` is a scilab function, its calling sequence must be: `y=dfun(x [, optional parameters])` (notes that even if `dfun` needs optional parameters it must appear simply as `dfun` in the calling sequence of `leastsq`).

case 2:
when `dfun` is defined by a Fortran or C routine it must appear as `dfun_name` (a string) in the calling sequence of `leastsq` (`dfun_name` being the name of the routine which must be linked to Scilab). The calling sequences for this routine are nearly the same than for `fun`:

```
In Fortran:    subroutine dfun(m, n, x, params, y)
               integer m,n
               double precision x(n), params(*), y(m,n)

In C:          void fun(int *m, int *n, double *x, double *params, double *y
```

in the C case *dfun(i,j)=dfi/dxj* must be stored in `y[m*(j-1)+i-1]`.

Remarks

Like datafit, `leastsq` is a front end onto the optim function. If you want to try the Levenberg-Marquard method instead, use lsqrsolve.

A least squares problem may be solved directly with the optim function ; in this case the function NDcost may be useful to compute the derivatives (see the NDcost help page which provides a simple example for parameters identification of a differential equation).

Examples

```
// We will show different calling possibilities of leastsq on one (trivial) exa
// which is non linear but doesn't really need to be solved with leastsq (apply
// log linearizes the model and the problem may be solved with linear algebra).
// In this example we look for the 2 parameters x(1) and x(2) of a simple
// exponential decay model (x(1) being the unknow initial value and x(2) the
// decay constant):

function y = yth(t, x)
   y  = x(1)*exp(-x(2)*t)
endfunction

// we have the m measures (ti, yi):
m = 10;
tm = [0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25, 2.5]';
ym = [0.79, 0.59, 0.47, 0.36, 0.29, 0.23, 0.17, 0.15, 0.12, 0.08]';
wm = ones(m,1); // measure weights (here all equal to 1...)

// and we want to find the parameters x such that the model fits the given
// datas in the least square sense:
//
//  minimize  f(x) = sum_i  wm(i)^2 ( yth(tm(i),x) - ym(i) )^2

// initial parameters guess
x0 = [1.5 ; 0.8];

// in the first examples, we define the function fun and dfun
// in scilab language
```

```
function e = myfun(x, tm, ym, wm)
   e = wm.*( yth(tm, x) - ym )
endfunction

function g = mydfun(x, tm, ym, wm)
   v = wm.*exp(-x(2)*tm)
   g = [v , -x(1)*tm.*v]
endfunction

// now we could call leastsq:

// 1- the simplest call
[f,xopt, gropt] = leastsq(list(myfun,tm,ym,wm),x0)

// 2- we provide the Jacobian
[f,xopt, gropt] = leastsq(list(myfun,tm,ym,wm),mydfun,x0)

// a small graphic (before showing other calling features)
tt = linspace(0,1.1*max(tm),100)';
yy = yth(tt, xopt);
xbasc()
plot2d(tm, ym, style=-2)
plot2d(tt, yy, style = 2)
legend(["measure points", "fitted curve"]);
xtitle("a simple fit with leastsq")

// 3- how to get some information (we use imp=1)
[f,xopt, gropt] = leastsq(1,list(myfun,tm,ym,wm),mydfun,x0)

// 4- using the conjugate gradient (instead of quasi Newton)
[f,xopt, gropt] = leastsq(1,list(myfun,tm,ym,wm),mydfun,x0,"gc")

// 5- how to provide bound constraints (not useful here !)
xinf = [-%inf,-%inf]; xsup = [%inf, %inf];
[f,xopt, gropt] = leastsq(list(myfun,tm,ym,wm),"b",xinf,xsup,x0) // without Jac
[f,xopt, gropt] = leastsq(list(myfun,tm,ym,wm),mydfun,"b",xinf,xsup,x0) // with

// 6- playing with some stopping parameters of the algorithm
//    (allows only 40 function calls, 8 iterations and set epsg=0.01, epsf=0.1)
[f,xopt, gropt] = leastsq(1,list(myfun,tm,ym,wm),mydfun,x0,"ar",40,8,0.01,0.1)


// 7 and 8: now we want to define fun and dfun in fortran then in C
//          Note that the "compile and link to scilab" method used here
//          is believed to be OS independant (but there are some requirements,
//          in particular you need a C and a fortran compiler, and they must
//          be compatible with the ones used to build your scilab binary).

// 7- fun and dfun in fortran

// 7-1/ Let 's Scilab write the fortran code (in the TMPDIR directory):
f_code = ["      subroutine myfun(m,n,x,param,f)"
          "*     param(i) = tm(i), param(m+i) = ym(i), param(2m+i) = wm(i)"
          "      implicit none"
          "      integer n,m"
          "      double precision x(n), param(*), f(m)"
          "      integer i"
          "      do i = 1,m"
```

```
          "         f(i) = param(2*m+i)*( x(1)*exp(-x(2)*param(i)) - param(m+i)
          "      enddo"
          "      end ! subroutine fun"
          ""
          "      subroutine mydfun(m,n,x,param,df)"
          "*     param(i) = tm(i), param(m+i) = ym(i), param(2m+i) = wm(i)"
          "      implicit none"
          "      integer n,m"
          "      double precision x(n), param(*), df(m,n)"
          "      integer i"
          "      do i = 1,m"
          "         df(i,1) =  param(2*m+i)*exp(-x(2)*param(i))"
          "         df(i,2) = -x(1)*param(i)*df(i,1)"
          "      enddo"
          "      end ! subroutine dfun"];

mputl(f_code,TMPDIR+'/myfun.f')

// 7-2/ compiles it. You need a fortran compiler !
names = ["myfun" "mydfun"]
flibname = ilib_for_link(names,"myfun.o",[],"f",TMPDIR+"/Makefile");

// 7-3/ link it to scilab (see link help page)
link(flibname,names,"f")

// 7-4/ ready for the leastsq call: be carreful don't forget to
//      give the dimension m after the routine name !
[f,xopt, gropt] = leastsq(list("myfun",m,tm,ym,wm),x0)  // without Jacobian
[f,xopt, gropt] = leastsq(list("myfun",m,tm,ym,wm),"mydfun",x0) // with Jacobia

// 8- last example: fun and dfun in C

// 8-1/ Let 's Scilab write the C code (in the TMPDIR directory):
c_code = ["#include <math.h>"
          "void myfunc(int *m,int *n, double *x, double *param, double *f)"
          "{"
          "  /*  param[i] = tm[i], param[m+i] = ym[i], param[2m+i] = wm[i] */"
          "  int i;"
          "  for ( i = 0 ; i < *m ; i++ )"
          "    f[i] = param[2*(*m)+i]*( x[0]*exp(-x[1]*param[i]) - param[(*m)+i
          "  return;"
          "}"
          ""
          "void mydfunc(int *m,int *n, double *x, double *param, double *df)"
          "{"
          "  /*  param[i] = tm[i], param[m+i] = ym[i], param[2m+i] = wm[i] */"
          "  int i;"
          "  for ( i = 0 ; i < *m ; i++ )"
          "    {"
          "      df[i] = param[2*(*m)+i]*exp(-x[1]*param[i]);"
          "      df[i+(*m)] = -x[0]*param[i]*df[i];"
          "    }"
          "  return;"
          "}"];

mputl(c_code,TMPDIR+'/myfunc.c')

// 8-2/ compiles it. You need a C compiler !
```

```
names = ["myfunc" "mydfunc"]
clibname = ilib_for_link(names,"myfunc.o",[],"c",TMPDIR+"/Makefile");

// 8-3/ link it to scilab (see link help page)
link(clibname,names,"c")

// 8-4/ ready for the leastsq call
[f,xopt, gropt] = leastsq(list("myfunc",m,tm,ym,wm),"mydfunc",x0)
```

See Also

lsqrsolve, optim, NDcost, datafit, external, qpsolve, linpro

Name

linpro — linear programming solver (obsolete)

Descriptio

linpro has been moved to a external contribution.

```
Please download quapro toolboxe at http://www.scilab.org/contrib/
index_contrib.php?page=download.php.
```

See Also

qpsolve

Name

lmisolver — linear matrix inequation solver

```
[XLISTF[,OPT]] = lmisolver(XLIST0,evalfunc [,options])
```

Parameters

XLIST0
: a list of containing initial guess (e.g. `XLIST0=list(X1,X2,..,Xn)`)

evalfunc
: a Scilab function ("external" function with specific syntax)

  The syntax the function `evalfunc` must be as follows:

  `[LME,LMI,OBJ]=evalfunct(X)` where `X` is a list of matrices, `LME, LMI` are lists and `OBJ` a real scalar.

XLISTF
: a list of matrices (e.g. `XLIST0=list(X1,X2,..,Xn)`)

options
: optional parameter. If given, `options` is a real row vector with 5 components `[Mbound,abstol,nu,maxiters,reltol]`

Description

`lmisolver` solves the following problem:

minimize `f(X1,X2,...,Xn)` a linear function of Xi's

under the linear constraints: `Gi(X1,X2,...,Xn)=0` for i=1,...,p and LMI (linear matrix inequalities) constraints:

`Hj(X1,X2,...,Xn) > 0` for j=1,...,q

The functions f, G, H are coded in the Scilab function `evalfunc` and the set of matrices Xi's in the list X (i.e. `X=list(X1,...,Xn)`).

The function `evalfun` must return in the list `LME` the matrices `G1(X),...,Gp(X)` (i.e. `LME(i)=Gi(X1,...,Xn),` i=1,...,p). `evalfun` must return in the list `LMI` the matrices `H1(X0),...,Hq(X)` (i.e. `LMI(j)=Hj(X1,...,Xn)`, j=1,...,q). `evalfun` must return in `OBJ` the value of `f(X)` (i.e. `OBJ=f(X1,...,Xn)`).

`lmisolver` returns in `XLISTF`, a list of real matrices, i. e. `XLIST=list(X1,X2,..,Xn)` where the Xi's solve the LMI problem:

Defining `Y,Z` and `cost` by:

`[Y,Z,cost]=evalfunc(XLIST)`, `Y` is a list of zero matrices, `Y=list(Y1,...,Yp)`, `Y1=0, Y2=0, ..., Yp=0`.

`Z` is a list of square symmetric matrices, `Z=list(Z1,...,Zq)` , which are semi positive definite `Z1>0, Z2>0, ..., Zq>0` (i.e. `spec(Z(j))` > 0),

`cost` is minimized.

`lmisolver` can also solve LMI problems in which the `Xi's` are not matrices but lists of matrices. More details are given in the documentation of LMITOOL.

Examples

```
//Find diagonal matrix X (i.e. X=diag(diag(X), p=1) such that
//A1'*X+X*A1+Q1 < 0, A2'*X+X*A2+Q2 < 0 (q=2) and trace(X) is maximized
n  = 2;
A1 = rand(n,n);
A2 = rand(n,n);
Xs = diag(1:n);
Q1 = -(A1'*Xs+Xs*A1+0.1*eye());
Q2 = -(A2'*Xs+Xs*A2+0.2*eye());

deff('[LME,LMI,OBJ]=evalf(Xlist)','X   = Xlist(1); ...
                                   LME = X-diag(diag(X));...
                                   LMI = list(-(A1''*X+X*A1+Q1),-(A2''*X+X*A2+Q
                                   OBJ = -sum(diag(X))  ');

X=lmisolver(list(zeros(A1)),evalf);

X=X(1)
[Y,Z,c]=evalf(X)
```

See Also

lmitool

Name

lmitool — tool for solving linear matrix inequations

```
lmitool()

lmitool(filename)

txt=lmitool(probname,varlist,datalist)
```

Parameters

filename
: a string referring to a `.sci` function

probname
: a string containing the name of the problem

varlist
: a string containing the names of the unknown matrices (separated by commas if there are more than one)

datalist
: a string containing the names of data matrices (separated by commas if there are more than one)

txt
: a string providing information on what the user should do next

Description

`lmitool()` or `lmitool(filename)` is used to define interactively a LMI problem. In the non interactive mode, `txt=lmitool(probname,varlist,datalist)` generates a file in the current directory. The name of this file is obtained by adding `.sci` to the end of `probname`. This file is the skeleton of a solver function and the corresponding evaluation function needed by `lmisolver`.

See Also

lmisolver

Name

lsqrsolve — minimize the sum of the squares of nonlinear functions, levenberg-marquardt algorithm

```
[x [,v [,info]]]=lsqrsolve(x0,fct,m [,stop [,diag]])
[x [,v [,info]]]=lsqrsolve(x0,fct,m ,fjac [,stop [,diag]])
```

Parameters

x0
: real vector of size `n` (initial estimate of the solution vector).

fct
: external (i.e function or list or string).

m
: integer, the number of functions. `m` must be greater than or equal to `n`.

fjac
: external (i.e function or list or string).

stop
: optional vector `[ftol,xtol,gtol,maxfev,epsfcn,factor]` the default value is `[1.d-8,1.d-8,1.d-5,1000,0,100]`

  ftol
  : A positive real number,termination occurs when both the actual and predicted relative reductions in the sum of squares are at most `ftol`. therefore, `ftol` measures the relative error desired in the sum of squares.

  xtol
  : A positive real number, termination occurs when the relative error between two consecutive iterates is at most `xtol`. therefore, `xtol` measures the relative error desired in the approximate solution.

  gtol
  : A nonnegative input variable. termination occurs when the cosine of the angle between `fct`(x) and any column of the jacobian is at most `gtol` in absolute value. therefore, `gtol` measures the orthogonality desired between the function vector and the columns of the jacobian.

  maxfev
  : A positive integer, termination occurs when the number of calls to `fct` is at least maxfev by the end of an iteration.

  epsfcn
  : A positive real number, used in determining a suitable step length for the forward-difference approximation. this approximation assumes that the relative errors in the functions are of the order of `epsfcn`. if `epsfcn` is less than the machine precision, it is assumed that the relative errors in the functions are of the order of the machine precision.

  factor
  : A positive real number, used in determining the initial step bound. this bound is set to the product of factor and the euclidean norm of `diag*x` if nonzero, or else to `factor` itself. in most cases `factor` should lie in the interval `(0.1,100)`. `100` is a generally recommended value.

diag
: is an array of length `n`. `diag` must contain positive entries that serve as multiplicative scale factors for the variables.

x :
real vector (final estimate of the solution vector).

v :
real vector (value of `fct(x)`).

info
termination indicator

0
improper input parameters.

1
algorithm estimates that the relative error between `x` and the solution is at most `tol`.

2
number of calls to `fcn` reached

3
`tol` is too small. No further improvement in the approximate solution `x` is possible.

4
iteration is not making good progress.

5
number of calls to `fcn` has reached or exceeded `maxfev`

6
`ftol` is too small. no further reduction in the sum of squares is possible.

7
`xtol` is too small. no further improvement in the approximate solution`x` is possible.

8
`gtol` is too small. `fvec` is orthogonal to the columns of the jacobian to machine precision.

Description

minimize the sum of the squares of m nonlinear functions in n variables by a modification of the levenberg-marquardt algorithm. the user must provide a subroutine which calculates the functions. the jacobian is then calculated by a forward-difference approximation.

minimize `sum(fct(x,m).^2)` where `fct` is function from `R^n` to `R^m`

`fct` should be :

- a Scilab function whose calling sequence is `v=fct(x,m)` given `x` and `m`.
- a character string which refers to a C or Fortran routine which must be linked to Scilab.

  Fortran calling sequence should be `fct(m,n,x,v,iflag)` where `m`, `n`, `iflag` are integers, `x` a double precision vector of size `n` and `v` a double precision vector of size `m`.

  C calling sequence should be `fct(int *m, int *n, double x[],double v[],int *iflag)`

`fjac` is an external which returns `v=d(fct)/dx (x)`. it should be :

a Scilab function
whose calling sequence is `J=fjac(x,m)` given `x` and `m`.

a character string
it refers to a C or Fortran routine which must be linked to Scilab.

Fortran calling sequence should be `fjac(m,n,x,jac,iflag)` where `m`, `n`, `iflag` are integers, `x` a double precision vector of size `n` and `jac` a double precision vector of size `m*n`.

C calling sequence should be `fjac(int *m, int *n, double x[],double v[],int *iflag)`

return -1 in iflag to stop the algoritm if the function or jacobian could not be evaluated.

Examples

```
// A simple example with lsqrsolve
a=[1,7;
   2,8
   4 3];
b=[10;11;-1];

function y=f1(x,m)
  y=a*x+b;
endfunction

[xsol,v]=lsqrsolve([100;100],f1,3)
xsol+a\b


function y=fj1(x,m)
  y=a;
endfunction

[xsol,v]=lsqrsolve([100;100],f1,3,fj1)
xsol+a\b

// Data fitting problem
// 1 build the data
a=34;b=12;c=14;

deff('y=FF(x)','y=a*(x-b)+c*x.*x');
X=(0:.1:3)';Y=FF(X)+100*(rand()-.5);

//solve
function e=f1(abc,m)
  a=abc(1);b=abc(2),c=abc(3),
  e=Y-(a*(X-b)+c*X.*X);
endfunction

[abc,v]=lsqrsolve([10;10;10],f1,size(X,1));
abc
norm(v)
```

See Also

external, qpsolve, linpro, optim, fsolve

Used Functions

lmdif, lmder from minpack, Argonne National Laboratory.

Name

mps2linpro — convert lp problem given in MPS format to linpro format (obsolete)

Description

mps2linpro has been moved to a external contribution.

Please download quapro toolboxe at http://www.scilab.org/contrib/index_contrib.php?page=download.php.

See Also

qpsolve

Name

numdiff — numerical gradient estimation

```
g=numdiff(fun,x [,dx])
```

Parameters

fun
: an external, Scilab function or list. See below for calling sequence, see also external for details about external functions.

x
: vector, the argument of the function `fun`

dx
: vector, the finite difference step. Default value is `dx=sqrt(%eps)*(1+1d-3*abs(x))`

g
: vector, the estimated gradient

Description

given a function `fun(x)` from `R^n` to `R^p` computes the matrix `g` such as

```
[ d f ] [ i ] g = [ ---- ] ij [ d x ] [ j ]
```

using finite difference methods.

Without parameters, the function fun calling sequence is `y=fun(x)`, and numdiff can be called as `g=numdiff(fun,x)`. Else the function fun calling sequence must be `y=fun(x,param_1,pararm_2,..,param_q)`. If parameters `param_1,param_2,..param_q` exist then `numdiff` can be called as follow `g=numdiff(list(fun,param_1,param_2,..param_q),x)`.

Examples

```
// example 1 (without parameters)
// myfun is a function from R^2 to R :   (x(1),x(2)) |--> myfun(x)
function f=myfun(x)
  f=x(1)*x(1)+x(1)*x(2)
endfunction

x=[5 8]
g=numdiff(myfun,x)

// The exact gradient (i.e derivate belong x(1) :first component and derivate b
exact=[2*x(1)+x(2)  x(1)]


//example 2 (with parameters)
// myfun is a function from R to R:  x(1) |--> myfun(x)
// myfun contains 3 parameters, a, b, c
function  f=myfun(x,a,b,c)
  f=(x+a)^c+b
```

```
endfunction

a=3; b=4; c=2;
x=1
g2=numdiff(list(myfun,a,b,c),x)

// The exact gradient, i.e derivate belong x(1), is :
exact2=c*(x+a)^(c-1)
```

See Also

optim, external

Name

optim — non-linear optimization routine

```
[f,xopt]=optim(costf,x0)
[f [,xopt [,gradopt [,work]]]]=optim(costf [,<contr>],x0 [,algo] [,df0 [,mem]]
```

Parameters

costf
: external, i.e Scilab function list or string (`costf` is the cost function, that is, a Scilab script, a Fortran 77 routine or a C function.

x0
: real vector (initial value of variable to be minimized).

f
: value of optimal cost (`f=costf(xopt)`)

xopt
: best value of `x` found.

<contr>
: keyword representing the following sequence of arguments: `'b',binf,bsup` with `binf` and `bsup` are real vectors with same dimension as `x0`. `binf` and `bsup` are lower and upper bounds on `x`.

algo

- `'qn'` : quasi-Newton (this is the default solver)
- `'gc'` : conjugate gradient
- `'nd'` : non-differentiable.

  Note that the conjugate gradient solver does not accept bounds on `x`.

df0
: real scalar. Guessed decreasing of `f` at first iteration. (`df0=1` is the default value).

mem :
: integer, number of variables used to approximate the Hessian. Default value is 10. This feature is available for the Gradient-Conjugate algorithm "gc" without constraints and the non-smooth algorithm "nd" without constraints.

<stop>
: keyword representing the sequence of optional parameters controlling the convergence of the algorithm. `'ar',nap [,iter [,epsg [,epsf [,epsx]]]]`

  "ar"
  : reserved keyword for stopping rule selection defined as follows:

  nap
  : maximum number of calls to `costf` allowed (default is 100).

  iter
  : maximum number of iterations allowed (default is 100).

  epsg
  : threshold on gradient norm.

epsf
: threshold controlling decreasing of `f`

epsx
: threshold controlling variation of `x`. This vector (possibly matrix) of same size as `x0` can be used to scale `x`.

<params>
: keyword representing the method to initialize the arguments `ti, td` passed to the objective function, provided as a C or Fortran routine. This option has no meaning when the cost function is a Scilab script. <params> can be set to only one of the following values.

- "in"

  That mode allows to allocate memory in the internal Scilab workspace so that the objective function can get arrays with the required size, but without directly allocating the memory. "in" stands for "initialization". In that mode, before the value and derivative of the objective function is to be computed, there is a dialog between the optim Scilab primitive and the objective function. In this dialog, the objective function is called two times, with particular values of the "ind" parameter. The first time, ind is set to 10 and the objective function is expected to set the nizs, nrzs and ndzs integer parameters of the "nird" common.

  ```
  common /nird/ nizs,nrzs,ndzs
  ```

  This allows Scilab to allocate memory inside its internal workspace. The second time the objective function is called, ind is set to 11 and the objective function is expected to set the ti, tr and tz arrays. After this initialization phase, each time it is called, the objective function is ensured that the ti, tr and tz arrays which are passed to it have the values that have been previously initialized.

- "ti",valti

  In this mode, valti is expected to be a Scilab vector variable containing integers. Whenever the objective function is called, the ti array it receives contains the values of the Scilab variable.

- "td", valtd

  In this mode, valtd is expected to be a Scilab vector variable containing double values. Whenever the objective function is called, the td array it receives contains the values of the Scilab variable.

- "ti",valti,"td",valtd

  This mode combines the two previous.

The `ti, td` arrays may be used so that the objective function can be computed. For example, if the objective function is a polynomial, the ti array may may be used to store the coefficients of that polynomial.

Users should choose carefully between the "in" mode and the "ti" and "td" mode, depending on the fact that the arrays are Scilab variables or not. If the data is available as Scilab variables, then the "ti", valti, "td", valtd mode should be chosen. If the data is available directly from the objective function, the "in" mode should be chosen. Notice that there is no "tr" mode, since, in Scilab, all real values are of "double" type.

If neither the "in" mode, nor the "ti", "td" mode is chosen, that is, if <params> is not present as an option of the optim primitive, the user may should not assume that the ti,tr and td arrays can be used : reading or writing the arrays may generate unpredictable results.

"imp=iflag"
named argument used to set the trace mode. The possible values for iflag are 0,1,2 and >2. Use this option with caution : most of these reports are written on the Scilab standard output.

- iflag=0: nothing (except errors) is reported (this is the default),
- iflag=1: initial and final reports,
- iflag=2: adds a report per iteration,
- iflag>2: add reports on linear search.

gradopt
gradient of `costf` at `xopt`

work
working array for hot restart for quasi-Newton method. This array is automatically initialized by `optim` when `optim` is invoked. It can be used as input parameter to speed-up the calculations.

Description

Non-linear optimization routine for programs without constraints or with bound constraints:

```
min costf(x) w.r.t x.
```

`costf` is an "external" i.e a Scilab function, a list or a string giving the name of a C or Fortran routine (see "external"). This external must return the value `f` of the cost function at the point `x` and the gradient `g` of the cost function at the point `x`.

- Scilab function case
If `costf` is a Scilab function, the calling sequence for `costf` must be:

```
[f,g,ind]=costf(x,ind)
```

Here, `costf` is a function which returns `f`, value (real number) of cost function at `x`, and `g`, gradient vector of cost function at `x`. The variable `ind` is described below.

- List case
If `costf` is a list, it should be of the form: `list(real_costf, arg1,...,argn)` with `real_costf` a Scilab function with calling sequence : `[f,g,ind]=costf(x,ind,arg1,... argn)`. The `x`, `f`, `g`, `ind` arguments have the same meaning that above. `argi` arguments can be used to pass function parameters.

- String case
If `costf` is a character string, it refers to the name of a C or Fortran routine which must be linked to Scilab

* Fortran case
The interface of the Fortran subroutine computing the objective must be :

```
subroutine costf(ind,n,x,f,g,ti,tr,td)
```

with the following declarations:

```
integer ind,n ti(*)
```

```
double precision x(n),f,g(n),td(*)
real tr(*)
```

The argument `ind` is described below.

If ind = 2, 3 or 4, the inputs of the routine are : `x, ind, n, ti, tr,td`.

If ind = 2, 3 or 4, the outputs of the routine are : `f` and `g`.

* C case

The interface of the C function computing the objective must be :

```
void costf(int *ind, int *n, double *x, double *f, double *g, int *ti, fl
```

The argument `ind` is described below.

The inputs and outputs of the function are the same as in the fortran case.

If `ind=2` (resp. `3, 4`), `costf` must provide `f` (resp. `g, f` and `g`).

If `ind=1` nothing is computed (used for display purposes only).

On output, `ind<0` means that `f` cannot be evaluated at `x` and `ind=0` interrupts the optimization.

Example #1 : Scilab function

The following is an example with a Scilab function. Notice, for simplifications reasons, the Scilab function "cost" of the following example computes the objective function f and its derivative no matter of the value of ind. This allows to keep the example simple. In practical situations though, the computation of "f" and "g" may raise performances issues so that a direct optimization may be to use the value of "ind" to compute "f" and "g" only when needed.

```
// External function written in Scilab
xref=[1;2;3];x0=[1;-1;1]
deff('[f,g,ind]=cost(x,ind)','f=0.5*norm(x-xref)^2,g=x-xref');

// Simplest call
[f,xopt]=optim(cost,x0)

// By conjugate gradient - you can use 'qn', 'gc' or 'nd'
[f,xopt,gopt]=optim(cost,x0,'gc')

//Seen as non differentiable
[f,xopt,gopt]=optim(cost,x0,'nd')

// Upper and lower bounds on x
[f,xopt,gopt]=optim(cost,'b',[-1;0;2],[0.5;1;4],x0)

// Upper and lower bounds on x and setting up the algorithm to 'gc'
[f,xopt,gopt]=optim(cost,'b',[-1;0;2],[0.5;1;4],x0,'gc')

// Bound on the number of call to the objective function
[f,xopt,gopt]=optim(cost,'b',[-1;0;2],[0.5;1;4],x0,'gc','ar',3)

// Set max number of call to the objective function (3)
// Set max number of iterations (100)
```

```
// Set stopping threshold on the value of f (1e-6),
// on the value of the norm of the gradient of the objective function (1e-6)
// on the improvement on the parameters x_opt (1e-6;1e-6;1e-6)
[f,xopt,gopt]=optim(cost,'b',[-1;0;2],[0.5;1;4],x0,'gc','ar',3,100,1e-6,1e-6,[1

// Print information messages while optimizing
// Be careful, some messages are printed in a terminal. You must
// Scilab from the command line to see these messages.
[f,xopt]=optim(cost,x0,imp=3)

// Use the 'derivative' function to compute the partial
// derivatives of the previous problem
deff('y=my_f(x)','y=0.5*norm(x-xref)^2');
deff('y=my_df(x)','y=derivative(my_f,x)');
deff('[f,g,ind]=cost(x,ind)','f=my_f(x); ...
                              g=my_df(x)');

// Simplest call
xref=[1;2;3];x0=[1;-1;1]
[f,xopt]=optim(cost,x0)
```

Example #2 : C function

The following is an example with a C function, where a C source code is written into a file, dynamically compiled and loaded into Scilab, and then used by the "optim" solver. The interface of the "rosenc" function is fixed, even if the arguments are not really used in the cost function. This is because the underlying optimization solvers must assume that the objective function has a known, constant interface. In the following example, the arrays ti and tr are not used, only the array "td" is used, as a parameter of the Rosenbrock function. Notice that the content of the arrays ti and td are the same that the content of the Scilab variable, as expected.

```
// External function written in C (C compiler required)
// write down the C code (Rosenbrock problem)
C=['#include <math.h>'
   'double sq(double x)'
   '{ return x*x;}'
   'void rosenc(int *ind, int *n, double *x, double *f, double *g, '
   '                                int *ti, float *tr, double *td)'
   '{'
   '  double p;'
   '  int i;'
   '  p=td[0];'
   '  if (*ind==2||*ind==4) {'
   '    *f=1.0;'
   '    for (i=1;i<*n;i++)'
   '      *f+=p*sq(x[i]-sq(x[i-1]))+sq(1.0-x[i]);'
   '  }'
   '  if (*ind==3||*ind==4) {'
   '    g[0]=-4.0*p*(x[1]-sq(x[0]))*x[0];'
   '    for (i=1;i<*n-1;i++)'
   '      g[i]=2.0*p*(x[i]-sq(x[i-1]))-4.0*p*(x[i+1]-sq(x[i]))*x[i]-2.0*(1.0-x[
   '    g[*n-1]=2.0*p*(x[*n-1]-sq(x[*n-2]))-2.0*(1.0-x[*n-1]);'
   '  }'
   '}'];
mputl(C,TMPDIR+'/rosenc.c')
```

```
// compile the C code
l=ilib_for_link('rosenc','rosenc.o',[],'c',TMPDIR+'/Makefile');

// incremental linking
link(l,'rosenc','c')

//solve the problem
x0=[40;10;50];
p=100;
[f,xo,go]=optim('rosenc',x0,'td',p)
```

Example #3 : Fortran function

The following is an example with a Fortran function.

```
// External function written in Fortran (Fortran compiler required)
// write down the Fortran  code (Rosenbrock problem)
F=[ '      subroutine rosenf(ind, n, x, f, g, ti, tr, td)'
    '      integer ind,n,ti(*)'
    '      double precision x(n),f,g(n),td(*)'
    '      real tr(*)'
    'c'
    '      double precision y,p'
    '      p=td(1)'
    '      if (ind.eq.2.or.ind.eq.4) then'
    '        f=1.0d0'
    '        do i=2,n'
    '          f=f+p*(x(i)-x(i-1)**2)**2+(1.0d0-x(i))**2'
    '        enddo'
    '      endif'
    '      if (ind.eq.3.or.ind.eq.4) then'
    '        g(1)=-4.0d0*p*(x(2)-x(1)**2)*x(1)'
    '        if(n.gt.2) then'
    '          do i=2,n-1'
    '            g(i)=2.0d0*p*(x(i)-x(i-1)**2)-4.0d0*p*(x(i+1)-x(i)**2)*x(i)'
    '     &           -2.0d0*(1.0d0-x(i))'
    '          enddo'
    '        endif'
    '        g(n)=2.0d0*p*(x(n)-x(n-1)**2)-2.0d0*(1.0d0-x(n))'
    '      endif'
    '      return'
    '      end'];

mputl(F,TMPDIR+'/rosenf.f')

// compile the Fortran code
l=ilib_for_link('rosenf','rosenf.o',[],'f',TMPDIR+'/Makefile');

// incremental linking
link(l,'rosenf','f')

//solve the problem
x0=[40;10;50];
p=100;
[f,xo,go]=optim('rosenf',x0,'td',p)
```

Example #4 : Fortran function with initialization

The following is an example with a Fortran function in which the "in" option is used to allocate memory inside the Scilab environment. In this mode, there is a dialog between Scilab and the objective function. The goal of this dialog is to initialize the parameters of the objective function. Each part of this dialog is based on a specific value of the "ind" parameter.

At the beginning, Scilab calls the objective function, with the ind parameter equals to 10. This tells the objective function to initialize the sizes of the arrays it needs by setting the nizs, nrzs and ndzs integer parameters of the "nird" common. Then the objective function returns. At this point, Scilab creates internal variables and allocate memory for the variable izs, rzs and dzs. Scilab calls the objective function back again, this time with ind equals to 11. This tells the objective function to initialize the arrays izs, rzs and dzs. When the objective function has done so, it returns. Then Scilab enters in the real optimization mode and calls the optimization solver the user requested. Whenever the objective function is called, the izs, rzs and dzs arrays have the values that have been previously initialized.

```
//
// Define a fortran source code and compile it (fortran compiler required)
//
fortransource=['      subroutine rosenf(ind,n,x,f,g,izs,rzs,dzs)'
               'C     ----------------------------------------'
               'c     Example of cost function given by a subroutine'
               'c     if n<=2 returns ind=0'
               'c     f.bonnans, oct 86'
               '      implicit double precision (a-h,o-z)'
               '      real rzs(1)'
               '      double precision dzs(*)'
               '      dimension x(n),g(n),izs(*)'
               '      common/nird/nizs,nrzs,ndzs'
               '      if (n.lt.3) then'
               '        ind=0'
               '        return'
               '      endif'
               '      if(ind.eq.10) then'
               '         nizs=2'
               '         nrzs=1'
               '         ndzs=2'
               '         return'
               '      endif'
               '      if(ind.eq.11) then'
               '         izs(1)=5'
               '         izs(2)=10'
               '         dzs(2)=100.0d+0'
               '         return'
               '      endif'
               '      if(ind.eq.2)go to 5'
               '      if(ind.eq.3)go to 20'
               '      if(ind.eq.4)go to 5'
               '      ind=-1'
               '      return'
               '5     f=1.0d+0'
               '      do 10 i=2,n'
               '        im1=i-1'
               '10      f=f + dzs(2)*(x(i)-x(im1)**2)**2 + (1.0d+0-x(i))**2'
               '      if(ind.eq.2)return'
               '20    g(1)=-4.0d+0*dzs(2)*(x(2)-x(1)**2)*x(1)'
               '      nm1=n-1'
```

```
                '      do 30 i=2,nm1'
                '        im1=i-1'
                '        ip1=i+1'
                '        g(i)=2.0d+0*dzs(2)*(x(i)-x(im1)**2)'
                '30      g(i)=g(i) -4.0d+0*dzs(2)*(x(ip1)-x(i)**2)*x(i) - '
                '     &        2.0d+0*(1.0d+0-x(i))'
                '      g(n)=2.0d+0*dzs(2)*(x(n)-x(nm1)**2) - 2.0d+0*(1.0d+0-x(n)
                '      return'
                '      end'];
mputl(fortransource,TMPDIR+'/rosenf.f')

// compile the C code
libpath=ilib_for_link('rosenf','rosenf.o',[],'f',TMPDIR+'/Makefile');

// incremental linking
linkid=link(libpath,'rosenf','f');

x0=1.2*ones(1,5);
//
// Solve the problem
//
[f,x,g]=optim('rosenf',x0,'in');
```

Example #5 : Fortran function with initialization on Windows with Intel Fortran Compiler

Under the Windows operating system with Intel Fortran Compiler, one must carefully design the fortran source code so that the dynamic link works properly. On Scilab's side, the optimization component is dynamically linked and the symbol "nird" is exported out of the optimization dll. On the cost function's side, which is also dynamically linked, the "nird" common must be imported in the cost function dll.

The following example is a re-writing of the previous example, with special attention for the Windows operating system with Intel Fortran compiler as example. In that case, we introduce additionnal compiling instructions, which allows the compiler to import the "nird" symbol.

```
fortransource=['subroutine rosenf(ind,n,x,f,g,izs,rzs,dzs)'
               'cDEC$ IF DEFINED (FORDLL)'
               'cDEC$ ATTRIBUTES DLLIMPORT:: /nird/'
               'cDEC$ ENDIF'
               'C     ------------------------------------------'
               'c     Example of cost function given by a subroutine'
               'c     if n<=2 returns ind=0'
               'c     f.bonnans, oct 86'
               '      implicit double precision (a-h,o-z)'
[etc...]
```

See Also

external, qpsolve, linpro, datafit, leastsq, numdiff, derivative, NDcost

References

The following is a map from the various options to the underlying solvers, with some comments about the algorithm, when available.

"qn" without constraints
n1qn1 : a quasi-Newton method with a Wolfe-type line search

"qn" with bounds constraints
qnbd : a quasi-Newton method with projection

RR-0242 - A variant of a projected variable metric method for bound constrained optimization problems, Bonnans Frederic, Rapport de recherche de l'INRIA - Rocquencourt, Octobre 1983

"gc" without constraints
n1qn3 : a conjugate gradient method with BFGS.

"gc" with bounds constraints
gcbd : a BFGS-type method with limited memory and projection

"nd" without constraints
n1fc1 : a bundle method

"nd" with bounds constraints
not available

Name

qld — linear quadratic programming solver

```
[x,lagr]=qld(Q,p,C,b,ci,cs,me [,tol])
[x,lagr,info]=qld(Q,p,C,b,ci,cs,me [,tol])
```

Parameters

Q
: real positive definite symmetric matrix (dimension `n x n`).

p
: real (column) vector (dimension `n`)

C
: real matrix (dimension `(me + md) x n`)

b
: RHS column vector (dimension `(me + md)`)

ci
: column vector of lower-bounds (dimension `n`). If there are no lower bound constraints, put `ci = []`. If some components of `x` are bounded from below, set the other (unconstrained) values of `ci` to a very large negative number (e.g. `ci(j) = -number_properties('huge')`.

cs
: column vector of upper-bounds. (Same remarks as above).

me
: number of equality constraints (i.e. `C(1:me,:)*x = b(1:me)`)

tol
: :Floatting point number, required prÃ©cision.

x
: optimal solution found.

lagr
: vector of Lagrange multipliers. If lower and upper-bounds `ci,cs` are provided, `lagr` has `n + me + md` components and `lagr(1:n)` is the Lagrange vector associated with the bound constraints and `lagr (n+1 : n + me + md)` is the Lagrange vector associated with the linear constraints. (If an upper-bound (resp. lower-bound) constraint `i` is active `lagr(i)` is > 0 (resp. <0). If no bounds are provided, `lagr` has only `me + md` components.

info
: integer, return the execution status instead of sending errors.

  info==1 : Too many iterations needed

  info==2 : Accuracy insufficient to statisfy convergence criterion

  info==5 : Length of working array is too short

  info==10: The constraints are inconsistent

Description

$$\min \frac{1}{2} \cdot x^t \cdot Q \cdot x + p^t \cdot x$$
$$\text{with} C(j,:) \cdot x = b(j), j = 1, \cdots, me$$
$$C(j,:) \cdot x \leq b(j), j = me+1, \cdots, me+md$$
$$ci \leq x \leq cs$$

This function requires `Q` to be positive definite, if it is not the case, one may use the The contributed toolbox "**quapro**".

Examples

```
//Find x in R^6 such that:
//C1*x = b1 (3 equality constraints i.e me=3)
C1= [1,-1,1,0,3,1;
    -1,0,-3,-4,5,6;
     2,5,3,0,1,0];
b1=[1;2;3];

//C2*x <= b2 (2 inequality constraints)
C2=[0,1,0,1,2,-1;
    -1,0,2,1,1,0];
b2=[-1;2.5];

//with  x between ci and cs:
ci=[-1000;-10000;0;-1000;-1000;-1000];cs=[10000;100;1.5;100;100;1000];

//and minimize 0.5*x'*Q*x + p'*x with
p=[1;2;3;4;5;6]; Q=eye(6,6);

//No initial point is given;
C=[C1;C2];
b=[b1;b2];
me=3;
[x,lagr]=qld(Q,p,C,b,ci,cs,me)
//Only linear constraints (1 to 4) are active (lagr(1:6)=0):
```

See Also

qpsolve, optim

The contributed toolbox "quapro" may also be of interest, in particular for singular `Q`.

Authors

K.Schittkowski
, University of Bayreuth, Germany

A.L. Tits and J.L. Zhou
, University of Maryland

Used Functions

`ql0001.f` in `modules/optimization/src/fortran/ql0001.f`

Name

qp_solve — linear quadratic programming solver builtin

```
[x [,iact [,iter [,f]]]]=qp_solve(Q,p1,C1,b,me)
```

Parameters

Q
: real positive definite symmetric matrix (dimension `n x n`).

p
: real (column) vector (dimension `n`)

C
: real matrix (dimension `(me + md) x n`). This matrix may be dense or sparse.

b
: RHS column vector (dimension `m=(me + md)`)

me
: number of equality constraints (i.e. `x'*C(:,1:me) = b(1:me)'`)

x
: optimal solution found.

iact
: vector, indicator of active constraints. The first non zero entries give the index of the active constraints

iter
: 2x1 vector, first component gives the number of "main" iterations, the second one says how many constraints were deleted after they became active.

Description

$$\min \frac{1}{2} \cdot x^t \cdot Q \cdot x + p^t \cdot x$$
$$\text{with} x^t \cdot C(:,j) = b(j), j = 1, \cdots, \text{me}$$
$$x^t \cdot C(:,j) \geq b(j), j = \text{me}+1, \cdots, \text{me}+\text{md}$$

This function requires `Q` to be symmetric positive definite. If this hypothesis is not satisfied, one may use the contributed **quapro toolbox**.

Examples

```
// Find x in R^6 such that:
// x'*C1 = b1 (3 equality constraints i.e me=3)
C1= [ 1,-1, 2;
     -1, 0, 5;
      1,-3, 3;
      0,-4, 0;
      3, 5, 1;
      1, 6, 0];
b1=[1;2;3];
```

```
// x'*C2 >= b2 (2 inequality constraints)
C2= [ 0 ,1;
     -1, 0;
      0,-2;
     -1,-1;
     -2,-1;
      1, 0];
b2=[ 1;-2.5];

// and minimize 0.5*x'*Q*x - p'*x with
p=[-1;-2;-3;-4;-5;-6]; Q=eye(6,6);

me=3;
[x,iact,iter,f]=qp_solve(Q,p,[C1 C2],[b1;b2],me)
// Only linear constraints (1 to 4) are active
```

See Also

optim, qld, qpsolve

The contributed toolbox "quapro" may also be of interest, in particular for singular `Q`.

Memory requirements

Let r be

```
r=min(m,n)
```

Then the memory required by qp_solve during the computations is

```
2*n+r*(r+5)/2 + 2*m +1
```

Authors

S. Steer
INRIA (Scilab interface)

Berwin A. Turlach
School of Mathematics and Statistics (M019), The University of Western Australia, Crawley, AUSTRALIA (solver code)

References

- Goldfarb, D. and Idnani, A. (1982). "Dual and Primal-Dual Methods for Solving Strictly Convex Quadratic Programs", in J.P. Hennart (ed.), Numerical Analysis, Proceedings, Cocoyoc, Mexico 1981, Vol. 909 of Lecture Notes in Mathematics, Springer-Verlag, Berlin, pp. 226-239.
- Goldfarb, D. and Idnani, A. (1983). "A numerically stable dual method for solving strictly convex quadratic programs", Mathematical Programming 27: 1-33.
- QuadProg (Quadratic Programming Routines), Berwin A Turlach,http://www.maths.uwa.edu.au/~berwin/software/quadprog.html

Used Functions

qpgen2.f and >qpgen1.f (also named QP.solve.f) developped by Berwin A. Turlach according to the Goldfarb/Idnani algorithm

Name

qpsolve — linear quadratic programming solver

```
[x [,iact [,iter [,f]]]]=qpsolve(Q,p,C,b,ci,cs,me)
```

Parameters

Q
: real positive definite symmetric matrix (dimension `n x n`).

p
: real (column) vector (dimension `n`)

C
: real matrix (dimension `(me + md) x n`). This matrix may be dense or sparse.

b
: RHS column vector (dimension `m=(me + md)`)

ci
: column vector of lower-bounds (dimension `n`). If there are no lower bound constraints, put `ci = []`. If some components of `x` are bounded from below, set the other (unconstrained) values of `ci` to a very large negative number (e.g. `ci(j) = -number_properties('huge')`.

cs
: column vector of upper-bounds. (Same remarks as above).

me
: number of equality constraints (i.e. `C(1:me,:)*x = b(1:me)`)

x
: optimal solution found.

iact
: vector, indicator of active constraints. The first non zero entries give the index of the active constraints

iter
: . 2x1 vector, first component gives the number of "main" iterations, the second one says how many constraints were deleted after they became active.

Description

$$\min \frac{1}{2} \cdot x^t \cdot Q \cdot x + p^t \cdot x$$
$$\text{with} C(j,:) \cdot x = b(j), j = 1, \cdots, me$$
$$C(j,:) \cdot x \leq b(j), j = me+1, \cdots, me+md$$
$$ci \leq x \leq cs$$

This function requires `Q` to be symmetric positive definite. If that hypothesis is not satisfied, one may use the quapro function, which is provided in the Scilab quapro toolbox.

The qpsolve solver is implemented as a Scilab script, which calls the compiled qp_solve primitive. It is provided as a facility, in order to be a direct replacement for the former quapro solver : indeed, the qpsolve solver has been designed so that it provides the same interface, that is, the same input/output arguments. But the x0 and imp input arguments are available in quapro, but not in qpsolve.

Examples

```
//Find x in R^6 such that:
//C1*x = b1 (3 equality constraints i.e me=3)
C1= [1,-1,1,0,3,1;
    -1,0,-3,-4,5,6;
     2,5,3,0,1,0];
b1=[1;2;3];

//C2*x <= b2 (2 inequality constraints)
C2=[0,1,0,1,2,-1;
    -1,0,2,1,1,0];
b2=[-1;2.5];

//with  x between ci and cs:
ci=[-1000;-10000;0;-1000;-1000;-1000];
cs=[10000;100;1.5;100;100;1000];

//and minimize 0.5*x'*Q*x + p'*x with
p=[1;2;3;4;5;6]; Q=eye(6,6);

//No initial point is given;
C=[C1;C2];
b=[b1;b2];
me=3;
[x,iact,iter,f]=qpsolve(Q,p,C,b,ci,cs,me)
//Only linear constraints (1 to 4) are active
```

See Also

optim, qp_solve, qld

The contributed toolbox "quapro" may also be of interest, in particular for singular `Q`.

Memory requirements

Let r be

```
r=min(m,n)
```

Then the memory required by qpsolve during the computations is

```
2*n+r*(r+5)/2 + 2*m +1
```

Authors

S. Steer
INRIA (Scilab interface)

Berwin A. Turlach
School of Mathematics and Statistics (M019), The University of Western Australia, Crawley, AUSTRALIA (solver code)

References

- Goldfarb, D. and Idnani, A. (1982). "Dual and Primal-Dual Methods for Solving Strictly Convex Quadratic Programs", in J.P. Hennart (ed.), Numerical Analysis, Proceedings, Cocoyoc, Mexico 1981, Vol. 909 of Lecture Notes in Mathematics, Springer-Verlag, Berlin, pp. 226-239.
- Goldfarb, D. and Idnani, A. (1983). "A numerically stable dual method for solving strictly convex quadratic programs", Mathematical Programming 27: 1-33.
- QuadProg (Quadratic Programming Routines), Berwin A Turlach,http://www.maths.uwa.edu.au/~berwin/software/quadprog.html

Used Functions

qpgen1.f (also named QP.solve.f) developped by Berwin A. Turlach according to the Goldfarb/Idnani algorithm

Name

quapro — linear quadratic programming solver (obsolete)

Description

This function is superseded by qpsolve.

Users who are still interested by quapro may consider the Scilab quapro toolbox which provide the same features as in older Scilab releases.

See at http://www.scilab.org/contrib/index_contrib.php?page=download.php.

See Also

qpsolve

Name

semidef — semidefinite programming

```
[x,Z,ul,info]=semidef(x0,Z0,F,blck_szs,c,options)
```

Parameters

x0
: m x 1 real column vector (must be strictly primal feasible, see below)

Z0
: L x 1 real vector (compressed form of a strictly feasible dual matrix, see below)

F
: L x (m+1) real matrix

blck_szs
: p x 2 integer matrix (sizes of the blocks) defining the dimensions of the (square) diagonal blocks `size(Fi(j)=blck_szs(j) j=1,...,m+1`.

c
: m x 1 real vector

options
: row vector with five entries `[nu,abstol,reltol,0,maxiters]`

ul
: row vector with two entries

Description

`[x,Z,ul,info]=semidef(x0,Z0,F,blck_szs,c,options)` solves semidefinite program:

$$\min c^t \cdot x$$
$$\text{with} F_0 + x_1 \cdot F_1 + \cdots + x_m \cdot F_m \geq 0$$

and its dual:

$$\max -\text{trace}(F_0 \cdot Z)$$
$$\text{with} \text{trace}(F_i \cdot Z) = c_i, i = 1, \cdots, m$$
$$Z \geq 0$$

exploiting block structure in the matrices `F_i`.

It interfaces L. Vandenberghe and S. Boyd sp.c program.

The `Fi's` matrices are stored columnwise in `F` in compressed format: if `F_i^j`, i=0,..,m, j=1,...,L denote the jth (symmetric) diagonal block of `F_i`, then

$$F = \begin{pmatrix} \text{pack}(F_0^1) & \text{pack}(F_1^1) & \cdots & \text{pack}(F_m^1) \\ \text{pack}(F_0^2) & \text{pack}(F_1^2) & \cdots & \text{pack}(F_m^2) \\ \cdots & \cdots & \cdots & \cdots \\ \text{pack}(F_0^L) & \text{pack}(F_1^L) & \cdots & \text{pack}(F_m^L) \end{pmatrix}$$

where `pack(M)`, for symmetric `M`, is the vector `[M(1,1);M(1,2);...;M(1,n);M(2,2);M(2,3);...;M(2,n);...;M(n,n)]` (obtained by scanning columnwise the lower triangular part of `M`).

`blck_szs` gives the size of block `j`, ie, `size(F_i^j)=blck_szs(j)`.

Z is a block diagonal matrix with L blocks `Z^0, ..., Z^{L-1}`. `Z^j` has size `blck_szs[j] times blck_szs[j]`. Every block is stored using packed storage of the lower triangular part.

The 2 vector `ul` contains the primal objective value `c'*x` and the dual objective value `-trace(F_0*Z)`.

The entries of `options` are respectively: `nu` = a real parameter which ntrols the rate of convergence. `abstol` = absolute tolerance. `reltol` = relative tolerance (has a special meaning when negative). `tv` target value, only referenced if `reltol < 0`. `iters` = on entry: maximum number of iterations >= 0, on exit: the number of iterations taken. Notice that the absolute tolerance cannot be lower than 1.0e-8, that is, the absolute tolerance used in the algorithm is the maximum of the user-defined tolerance and the constant tolerance 1.0e-8.

`info` returns 1 if maxiters exceeded, 2 if absolute accuracy is reached, 3 if relative accuracy is reached, 4 if target value is reached, 5 if target value is not achievable; negative values indicate errors.

Convergence criterion:

```
(1) maxiters is exceeded
(2) duality gap is less than abstol
(3) primal and dual objective are both positive and
    duality gap is less than (reltol * dual objective)
    or primal and dual objective are both negative and
    duality gap is less than (reltol * minus the primal objective)
(4) reltol is negative and
    primal objective is less than tv or dual objective is greater
    than tv
```

Examples

```
F0=[2,1,0,0;
    1,2,0,0;
    0,0,3,1
    0,0,1,3];

F1=[1,2,0,0;
    2,1,0,0;
    0,0,1,3;
    0,0,3,1]

F2=[2,2,0,0;
    2,2,0,0;
    0,0,3,4;
    0,0,4,4];

blck_szs=[2,2];

F01=F0(1:2,1:2);F02=F0(3:4,3:4);
F11=F1(1:2,1:2);F12=F1(3:4,3:4);
F21=F2(1:2,1:2);F22=F2(3:4,3:4);
```

```
x0=[0;0]
Z0=2*F0;
Z01=Z0(1:2,1:2);Z02=Z0(3:4,3:4);
FF=[[F01(:);F02(:)],[F11(:);F12(:)],[F21(:);F22(:)]]
ZZ0=[[Z01(:);Z02(:)]];

c=[trace(F1*Z0);trace(F2*Z0)];
options=[10,1.d-10,1.d-10,0,50];

[x,Z,ul,info]=semidef(x0,pack(ZZ0),pack(FF),blck_szs,c,options)

w=vec2list(unpack(Z,blck_szs),[blck_szs;blck_szs]);Z=sysdiag(w(1),w(2))

c'*x+trace(F0*Z)
spec(F0+F1*x(1)+F2*x(2))
trace(F1*Z)-c(1)
trace(F2*Z)-c(2)
```

References

L. Vandenberghe and S. Boyd, " Semidefinite Programming," Informations Systems Laboratory, Stanford University, 1994.

Ju. E. Nesterov and M. J. Todd, "Self-Scaled Cones and Interior-Point Methods in Nonlinear Programming," Working Paper, CORE, Catholic University of Louvain, Louvain-la-Neuve, Belgium, April 1994.

SP: Software for Semidefinite Programming, http://www.ee.ucla.edu/~vandenbe/sp.html

Overloading

Name

overloading — display, functions and operators overloading capabilities

Description

In scilab, variable display, functions and operators may be defined for new objects using functions (scilab coded or primitives).

Display

The display of new objects defined by `tlist` structure may be overloaded (the default display is similar to `list`'s one). The overloading function must have no output argument a single input argument. It's name is formed as follow `%<tlist_type>_p` where `%<tlist_type>` stands for the first entry of the `tlist` type component truncated to the first 9 characters.

Operators

Each operator which is not defined for given operands type may be defined. The overloading function must have a single output argument and one or two inputs according to the number of operands. The function name is formed as follow:

for binary operators: `%<first_operand_type>_<op_code>_<second_operand_type>`

for unary operators: `%<operand_type>_<op_code>`

extraction and insertion operators which are n-nary operators are described below.

`<operand_type>`, `<first_operand_type>`, `<second_operand_type>` are sequence of characters associated with each data type as described in the following table:

data type	char code	data type	char code
constant	`s`	boolean	`b`
string	`c`	library	`f`
function pointer	`fptr`	handle	`h`
integer	`i`	list	`l`
function	`m`	compiled function	`mc`
polynomial	`p`	sparse	`sp`
boolean sparse	`spb`	tlist	`tlist_type`
size implicit polynomial	`ip`	Matlab sparse matrix	`msp`
mlist	`mlist_type`	pointer	`ptr`

`<op_code>` is a single character associated with each operator as described in the following table:

op	char code	op	char code
`'`	t	+	a
-	s	*	m
`/`	r	\	l
`^`	p	.*	x
./	d	.\	q
.*.	k	./.	y
`.\.`	z	:	b
*.	u	/.	v
`\.`	w	[a,b]	c

[a;b]	f	() extraction	e
() insertion	i	==	o
<>	n	|	g
&	h	.^	j
~	5	.'	0
<	1	>	2
<=	3	>=	4
		iext	6

The overloading function for extraction syntax `b=a(i1,...,in)` has the following calling sequence: `b=%<type_of_a>_e_(i1,...,in,a)`

and the syntax `[x1,..,xm]=a(i1,...,in)` has the following calling sequence: `[x1,..,xm]=%<type_of_a>_e_(i1,...,in,a)`

The overloading function associated to the insertion syntax `a(i1,...,in)=b` has the following calling sequence: `a=%<type_of_b>_i_<type_of_a>(i1,...,in,b,a)`.

The `6` char code may be used for some complex insertion algorithm like `x.b(2)=33` where `b` field is not defined in the structure `x`. The insertion is automatically decomposed into `temp=x.b; temp(2)=33; x.b=temp`. The `6` char code is used for the first step of this algorithm. The `6` overloading function is very similar to the `e`'s one.

Functions :

Some basic primitive function

may also be overloaded for new data type. When such a function is undefined for a particular data types the function `%<type_of_an_argument>_<function_name>` is called. User may add in this called function the definition associated with the input data types.

Examples

```
//DISPLAY
deff('[]=%tab_p(l)','disp([['' '';l(3)] [l(2);string(l(4))]])')
tlist('tab',['a','b'],['x';'y'],rand(2,2))

//OPERATOR
deff('x=%c_a_s(a,b)','x=a+string(b)')
's'+1

//FUNCTION
deff('x=%c_sin(a)','x=''sin(''+a+'')''')
sin('2*x')
```

See Also

tlist, disp, symbols

Parameters

Name

add_param — Add a parameter to a list of parameters

```
[ga_list,err] = add_param(list_name,param_name,param_value)
```

Parameters

list_name
: the list of parameters. This list must have been initialize by a call to init_param.

param_name
: a string. The name of the parameter to be added in the list of parameters.

param_value
: the value associated to the parameter param_name. This parameter is optional. You can set the value of this parameter via a call to set_param.

ga_list
: the updated list of parameters.

err
: an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

Description

- This function creates a new parameter in a list of parameters. You can set the value of the parameter using this function or you can set it via a call to set_param.

Examples

```
mylist = init_param();
mylist = add_param(mylist,'minbound',[0 0 0]);
```

See Also

init_param , set_param , get_param , remove_param , is_param

Authors

collette
: ycollet@freesurf.fr

Name

get_param — Get the value of a parameter in a parameter list

```
[res,err] = get_param(list_name,param_name)
```

Parameters

list_name
: the list of parameters. This list must have been initialize by a call to init_param.

param_name
: a string. The name of the parameter to be add in the list of parameters.

res
: the value of the parameter. If the parameter doesn't exist, res = [].

err
: an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

Description

- This function returns the value of the parameter param_name in a parameter list.

Examples

```
mylist = init_param();
mylist = add_param(mylist,'minbound',[0 0 0]);
disp(get_param(mylist,'minbound'));
```

See Also

init_param , set_param , add_param , remove_param , is_param

Authors

collette
: ycollet@freesurf.fr

Name

init_param — Initialize the structure which will handles the parameters list

```
ga_list = init_param()
```

Parameters

ga_list
: an initialized list of parameters (this list is empty and is of type plist).

Description

- This function initialize an empty list of parameters. You must initialize the list of parameters before using it.

Examples

```
mylist = init_param();
mylist = add_param(mylist,'minbound',[0 0 0]);
```

See Also

add_param , set_param , get_param , remove_param , is_param

Authors

collette
: ycollet@freesurf.fr

Name

is_param — Check if a parameter is present in a parameter list

```
[res,err] = is_param(list_name,param_name)
```

Parameters

list_name
: the list of parameters. This list must have been initialize by a call to init_param.

param_name
: a string. The name of the parameter to be add in the list of parameters.

res
: the result: %T is the parameter is present, %F otherwise.

err
: an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

Description

- This function checks if a parameter is present in a parameter list.

Examples

```
mylist = init_param();
mylist = add_param(mylist,'minbound',[0 0 0]);
disp(is_param(mylist,'minbound'));
disp(is_param(mylist,'maxbound'));
```

See Also

init_param , set_param , get_param , remove_param , add_param

Authors

collette
: ycollet@freesurf.fr

Name

list_param — List all the parameters name in a list of parameters

```
[string_list,err] = list_param(list_name)
```

Parameters

list_name
: the list of parameters. This list must have been initialize by a call to init_param.

string_list
: the list of parameters name.

err
: an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

Description

- List all the parameters name in a list of parameters.

Examples

```
mylist = init_param();
mylist = add_param(mylist,'minbound',[0 0 0]);
mylist = add_param(mylist,'maxbound',[1 1 1]);
disp(list_param(mylist));
```

See Also

init_param , set_param , get_param , remove_param , is_param

Authors

collette
: ycollet@freesurf.fr

Name

remove_param — Remove a parameter and its associated value from a list of parameters

```
[ga_list,err] = remove_param(list_name,param_name)
```

Parameters

list_name
: the list of parameters. This list must have been initialize by a call to init_param.

param_name
: a string. The name of the parameter to be removed from the list of parameters. If the parameter doesn't exist, nothing happens.

ga_list
: the updated list of parameters.

err
: an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

Description

- This function allows to remove a parameter and its associated value from a list of parameters.

Examples

```
mylist = init_param();
mylist = add_param(mylist,'minbound',[0 0 0]);
mylist = add_param(mylist,'maxbound',[0 0 0]);
mylist = remove_param(mylist,'minbound');
```

See Also

init_param , set_param , get_param , add_param , is_param

Authors

collette
: ycollet@freesurf.fr

Name

set_param — Set the value of a parameter in a parameter list

```
[ga_list,err] = set_param(list_name,param_name,param_value)
```

Parameters

list_name
: the list of parameters. This list must have been initialize by a call to init_param.

param_name
: a string. The name of the parameter to be added in the list of parameters.

param_value
: the value to be associated to the parameter param_name.

ga_list
: the updated list of parameters.

err
: an error flag which is set to %T if list_name is not of type plist (this list hasn't been initialized by a call to init_param).

Description

- This function sets the value of an already existing parameter. If the parameter doesn't exist, err is set to %T.

Examples

```
mylist = init_param();
mylist = add_param(mylist,'minbound',[0 0 0]);
[mylist,err] = set_param(mylist,'minbound',[1 1 1]); disp(err);
[mylist,err] = set_param(mylist,'maxbound',[1 1 1]); disp(err);
```

See Also

init_param , add_param , get_param , remove_param , is_param

Authors

collette
: ycollet@freesurf.fr

Polynomials

Name

bezout — Bezout equation for polynomials or integers

```
[thegcd,U]=bezout(p1,p2)
```

Parameters

p1, p2
: two real polynomials or two integer scalars (type equal to 8)

Description

`[thegcd,U]=bezout(p1,p2)` computes GCD `thegcd` of `p1` and `p2` and in addition a (2x2) unimodular matrix `U` such that:

```
[p1,p2]*U = [thegcd,0]
```

The lcm of `p1` and `p2` is given by:

`p1*U(1,2)` (or `-p2*U(2,2)`)

Examples

```
// polynomial case
x=poly(0,'x');
p1=(x+1)*(x-3)^5;p2=(x-2)*(x-3)^3;
[thegcd,U]=bezout(p1,p2)
det(U)
clean([p1,p2]*U)
thelcm=p1*U(1,2)
lcm([p1,p2])
// integer case
i1=int32(2*3^5); i2=int32(2^3*3^2);
[thegcd,U]=bezout(i1,i2)
V=int32([2^2*3^5, 2^3*3^2,2^2*3^4*5]);
[thegcd,U]=gcd(V)
V*U
lcm(V)
```

See Also

poly , roots , simp , clean , lcm

Authors

S. Steer INRIA

Name

clean — cleans matrices (round to zero small entries)

```
B=clean(A [,epsa [,epsr]])
```

Parameters

A
: a numerical matrix (scalar, polynomial, sparse...)

epsa,epsr
: real numbers. Cleaning tolerances (default values resp. 1.d-10 and 1.d-10)

Description

This function eliminates (i.e. set to zero) all the coefficients with absolute value < `epsa` or relative value < `epsr` (relative means relative w.r.t. 1-norm of coefficients) in a polynomial (possibly matrix polynomial or rational matrix).

Default values are `epsa=1.d-10` and `epsr=1.d-10;`

For a constant (non polynomial) matrix `clean(A,epsa)` sets to zero all entries of `A` smaller than `epsa`.

Examples

```
x=poly(0,'x');
w=[x,1,2+x;3+x,2-x,x^2;1,2,3+x]/3;
w*inv(w)
clean(w*inv(w))
```

Name

cmndred — common denominator form

```
[n,d]=cmndred(num,den)
```

Parameters

num, den
: two polynomial matrices of same dimensions

Description

`[n,d]=cmndred(num,den)` computes a polynomial matrix `n` and a common denominator polynomial `d` such that:

```
n/d=num./den
```

The rational matrix defined by `num./den` is `n/d`

See Also

simp , clean

Name

coeff — coefficients of matrix polynomial

```
[C]=coeff(Mp [,v])
```

Parameters

Mp
: polynomial matrix

v
: integer (row or column) vector of selected degrees

C
: big matrix of the coefficients

Description

`C=coeff(Mp)` returns in a big matrix `C` the coefficients of the polynomial matrix `Mp` . `C` is partitioned as `C=[C0,C1,...,Ck]` where the `Ci` are arranged in increasing order `k = maxi(degree(Mp))`

`C=coeff(Mp,v)` returns the matrix of coefficients with degree in `v` . (`v` is a row or column vector).

See Also

poly , degree , inv_coeff

Authors

S. Steer INRIA

Name

coffg — inverse of polynomial matrix

```
[Ns,d]=coffg(Fs)
```

Parameters

Fs
: square polynomial matrix

Description

`coffg` computes `Fs^-1` where `Fs` is a polynomial matrix by co-factors method.

`Fs` inverse = `Ns/d`

`d` = common denominator; `Ns` = numerator (a polynomial matrix)

(For large matrices,be patient...results are generally reliable)

Examples

```
s=poly(0,'s')
a=[ s, s^2+1; s  s^2-1];
[a1,d]=coffg(a);
(a1/d)-inv(a)
```

See Also

determ , detr , invr , penlaur , glever

Authors

F. D.; ;

Name

colcompr — column compression of polynomial matrix

```
[Y,rk,ac]=colcompr(A);
```

Parameters

A
: polynomial matrix

Y
: square polynomial matrix (right unimodular basis)

rk
: normal rank of A

Ac
: Ac=A*Y, polynomial matrix

Description

column compression of polynomial matrix A (compression to the left)

Examples

```
s=poly(0,'s');
p=[s;s*(s+1)^2;2*s^2+s^3];
[Y,rk,ac]=colcompr(p*p');
p*p'*Y
```

See Also

rowcompr

Name

degree — degree of polynomial matrix

```
[D]=degree(M)
```

Parameters

M
: polynomial matrix

D
: integer matrix

Description

returns the matrix of highest degrees of `M`.

See Also

poly , coeff , clean

Name

denom — denominator

```
den=denom(r)
```

Parameters

r
: rational or polynomial or constant matrix.

den
: polynomial matrix

Description

`den=denom(r)` returns the denominator of a rational matrix.

Since rationals are internally represented as `r=list(['r','num','den','dt'],num,den,[])`, `denom(r)` is the same as `r(3)`, `r('den')` or `r.den`

See Also

numer

Name

derivat — rational matrix derivative

```
pd=derivat(p)
```

Parameters

p
: polynomial or rational matrix

Description

computes the derivative of the polynomial or rational function matrix w.r.t the dummy variable.

Examples

```
s=poly(0,'s');
derivat(1/s)  // -1/s^2;
```

Name

determ — determinant of polynomial matrix

```
res=determ(W [,k])
```

Parameters

W
: real square polynomial matrix

k
: integer (upper bound for the degree of the determinant of W)

Description

returns the determinant of a real polynomial matrix (computation made by FFT if W size is greater than 2*2).

`res=determ(W [,k])k` is an integer larger than the actual degree of the determinant of `W`.

The default value of `k` is the smallest power of 2 which is larger than `n*maxi(degree(W))`.

Method (Only if W size is greater than 2*2) : evaluate the determinant of `W` for the Fourier frequencies and apply inverse FFT to the coefficients of the determinant.

Examples

```
s=poly(0,'s');
w=s*rand(10,10);
determ(w)
det(coeff(w,1))*s^10
```

See Also

det , detr , coffg

Authors

F.D.

Name

detr — polynomial determinant

```
d=detr(h)
```

Parameters

h
: polynomial or rational square matrix

Description

`d=detr(h)` returns the determinant `d` of the polynomial or rational function matrix `h`. Based on Leverrier's algorithm.

See Also

det , determ

Name

diophant — diophantine (Bezout) equation

```
[x,err]=diophant(p1p2,b)
```

Parameters

p1p2
: polynomial vector `p1p2 = [p1 p2]`

b
: polynomial

x
: polynomial vector [x1;x2]

Description

`diophant` solves the bezout equation:

`p1*x1+p2*x2=b` with `p1p2` a polynomial vector. If the equation is not solvable

else `err=0`

Examples

```
s=poly(0,'s');p1=(s+3)^2;p2=(1+s);
x1=s;x2=(2+s);
[x,err]=diophant([p1,p2],p1*x1+p2*x2);
p1*x1+p2*x2-p1*x(1)-p2*x(2)
```

Name

factors — numeric real factorization

```
[lnum,g]=factors(pol [,'flag'])
[lnum,lden,g]=factors(rat [,'flag'])
rat=factors(rat,'flag')
```

Parameters

pol
: real polynomial

rat
: real rational polynomial (`rat=pol1/pol2`)

lnum
: list of polynomials (of degrees 1 or 2)

lden
: list of polynomials (of degrees 1 or 2)

g
: real number

flag
: character string `'c'` or `'d'`

Description

returns the factors of polynomial `pol` in the list `lnum` and the "gain" g.

One has pol= g times product of entries of the list `lnum` (if `flag` is not given). If `flag='c'` is given, then one has `|pol(i omega)|` = `|g*prod(lnum_j(i omega)|`. If `flag='d'` is given, then one has `|pol(exp(i omega))|` = `|g*prod(lnum_i(exp(i omega))|`. If argument of `factors` is a 1x1 rational `rat=pol1/pol2`, the factors of the numerator `pol1` and the denominator `pol2` are returned in the lists `lnum` and `lden` respectively.

The "gain" is returned as `g`,i.e. one has: rat= g times (product entries in lnum) / (product entries in lden).

If `flag` is `'c'` (resp. `'d'`), the roots of `pol` are refected wrt the imaginary axis (resp. the unit circle), i.e. the factors in `lnum` are stable polynomials.

Same thing if `factors` is invoked with a rational arguments: the entries in `lnum` and `lden` are stable polynomials if `flag` is given. `R2=factors(R1,'c')` or `R2=factors(R1,'d')` with `R1` a rational function or SISO `syslin` list then the output `R2` is a transfer with stable numerator and denominator and with same magnitude as `R1` along the imaginary axis (`'c'`) or unit circle (`'d'`).

Examples

```
n=poly([0.2,2,5],'z');
d=poly([0.1,0.3,7],'z');
R=syslin('d',n,d);
R1=factors(R,'d')
roots(R1('num'))
roots(R1('den'))
```

```
w=exp(2*%i*%pi*[0:0.1:1]);
norm(abs(horner(R1,w))-abs(horner(R,w)))
```

See Also

simp

Name

gcd — gcd calculation

```
[pgcd,U]=gcd(p)
```

Parameters

p
: polynomial row vector `p=[p1,..,pn]` or integer row vector (type equal to 8)

Description

computes the gcd of components of `p` and a unimodular matrix (with polynomial inverse) `U`, with minimal degree such that

```
p*U=[0 ... 0 pgcd]
```

Examples

```
//polynomial case
s=poly(0,'s');
p=[s,s*(s+1)^2,2*s^2+s^3];
[pgcd,u]=gcd(p);
p*u

//integer case
V=int32([2^2*3^5, 2^3*3^2,2^2*3^4*5]);
[thegcd,U]=gcd(V)
V*U
```

See Also

bezout , lcm , hermit

Name

hermit — Hermite form

```
[Ar,U]=hermit(A)
```

Parameters

A
: polynomial matrix

Ar
: triangular polynomial matrix

U
: unimodolar polynomial matrix

Description

Hermite form: `U` is an unimodular matrix such that `A*U` is in Hermite triangular form:

The output variable is `Ar=A*U`.

Warning: Experimental version

Examples

```
s=poly(0,'s');
p=[s, s*(s+1)^2, 2*s^2+s^3];
[Ar,U]=hermit(p'*p);
clean(p'*p*U), det(U)
```

See Also

hrmt , htrianr

Name

horner — polynomial/rational evaluation

```
horner(P,x)
```

Parameters

P
: polynomial or rational matrix

x
: array of numbers or polynomials or rationals

Description

evaluates the polynomial or rational matrix `P = P(s)` when the variable `s` of the polynomial is replaced by `x`:

`horner(P,x)=P(x)`

Example (Bilinear transform): Assume `P = P(s)` is a rational matrix then the rational matrix `P((1+s)/(1-s))` is obtained by `horner(P,(1+s)/(1-s))`.

To evaluate a rational matrix at given frequencies use preferably the `freq` primitive.

Examples

```
//evaluation of a polynomial for a vector of numbers
P=poly(1:3,'x')
horner(P,[1 2 5])
horner(P,[1 2 5]+%i)

//evaluation of a rational
s=poly(0,'s');M=[s,1/s];
horner(M,1)
horner(M,%i)
horner(M,1/s)

//evaluation of a polynomial for a matrix of numbers
X= [1 2;3 4]
p=poly(1:3,'x','c')
m=horner(p, X)
1*X.^0+2*X.^1+3*X.^2
```

See Also

freq , repfreq , evstr

Name

hrmt — gcd of polynomials

```
[pg,U]=hrmt(v)
```

Parameters

v
: row of polynomials i.e. `1xk` polynomial matrix

pg
: polynomial

U
: unimodular matrix polynomial

Description

`[pg,U]=hrmt(v)` returns a unimodular matrix `U` and `pg` = gcd of row of polynomials `v` such that `v*U = [pg,0]`.

Examples

```
x=poly(0,'x');
v=[x*(x+1),x^2*(x+1),(x-2)*(x+1),(3*x^2+2)*(x+1)];
[pg,U]=hrmt(v);U=clean(U)
det(U)
```

See Also

gcd , htrianr

Authors

S. Steer INRIA

Name

htrianr — triangularization of polynomial matrix

```
[Ar,U,rk]=htrianr(A)
```

Parameters

A
: polynomial matrix

Ar
: polynomial matrix

U
: unimodular polynomial matrix

rk
: integer, normal rank of A

Description

triangularization of polynomial matrix A.

A is [m,n] , m <= n.

Ar=A*U

Warning: there is an elimination of "small" terms (see function code).

Examples

```
x=poly(0,'x');
M=[x;x^2;2+x^3]*[1,x-2,x^4];
[Mu,U,rk]=htrianr(M)
det(U)
M*U(:,1:2)
```

See Also

hrmt , colcompr

Name

invr — inversion of (rational) matrix

```
F = invr(H)
```

Parameters

H
: polynomial or rational matrix

F
: polynomial or rational matrix

Description

If `H` is a polynomial or rational function matrix, `invr` computes `H^(-1)` using Leverrier's algorithm (see function code)

Examples

```
s=poly(0,'s')
H=[s,s*s+2;1-s,1+s]; invr(H)
[Num,den]=coffg(H);Num/den
H=[1/s,(s+1);1/(s+2),(s+3)/s];invr(H)
```

See Also

glever , coffg , inv

Name

lcm — least common multiple

```
[pp,fact]=lcm(p)
```

Parameters

p
:

fact
: polynomial vector or integer vector (type equal to 8)

pp
: polynomial or integer

Description

`pp=lcm(p)` computes the lcm `pp` of polynomial vector `p`.

`[pp,fact]=lcm(p)` computes in addition the vector `fact` such that:

`p.*fact=pp*ones(p)`

Examples

```
//polynomial case
s=poly(0,'s');
p=[s,s*(s+1)^2,s^2*(s+2)];
[pp,fact]=lcm(p);
p.*fact, pp
//integer case
V=int32([2^2*3^5, 2^3*3^2,2^2*3^4*5]);
lcm(V)
```

See Also

gcd , bezout

Name

lcmdiag — least common multiple diagonal factorization

```
[N,D]=lcmdiag(H)
[N,D]=lcmdiag(H,flag)
```

Parameters

H
: rational matrix

N
: polynomial matrix

D
: diagonal polynomial matrix

flag
: character string: `'row'` or `'col'` (default)

Description

`[N,D]=lcmdiag(H,'row')` computes a factorization `D*H=N`, i.e. `H=D^(-1)*N` where D is a diagonal matrix with D(k,k)=lcm of kth row of H('den').

`[N,D]=lcmdiag(H)` or `[N,D]=lcmdiag(H,'col)` returns `H=N*D^(-1)` with diagonal D and D(k,k)=lcm of kth col of H('den')

Examples

```
s=poly(0,'s');
H=[1/s,(s+2)/s/(s+1)^2;1/(s^2*(s+2)),2/(s+2)];
[N,D]=lcmdiag(H);
N/D-H
```

See Also

lcm , gcd , bezout

Name

ldiv — polynomial matrix long division

```
[x]=ldiv(n,d,k)
```

Parameters

n,d
: two real polynomial matrices

k
: integer

Description

`x=ldiv(n,d,k)` gives the `k` first coefficients of the long division of `n` by `d` i.e. the Taylor expansion of the rational matrix `[nij(z)/dij(z)]` near infinity.

Coefficients of expansion of `nij/dij` are stored in `x((i-1)*n+k,j) k=1:n`

Examples

```
wss=ssrand(1,1,3);[a,b,c,d]=abcd(wss);
wtf=ss2tf(wss);
x1=ldiv(numer(wtf),denom(wtf),5)
x2=[c*b;c*a*b;c*a^2*b;c*a^3*b;c*a^4*b]
wssbis=markp2ss(x1',5,1,1);
wtfbis=clean(ss2tf(wssbis))
x3=ldiv(numer(wtfbis),denom(wtfbis),5)
```

See Also

arl2 , markp2ss , pdiv

Name

numer — numerator

```
num=numer(R)
```

Parameters

R
: rational or polynomial or constant matrix.

num
: polynomial matrix

Description

Utility fonction. `num=numer(R)` returns the numerator `num` of a rational function matrix `R` (`R` may be also a constant or polynomial matrix). `numer(R)` is equivalent to `R(2)`, `R('num')` or `R.num`

See Also

denom

Name

pdiv — polynomial division

```
[R,Q]=pdiv(P1,P2)
[Q]=pdiv(P1,P2)
```

Parameters

P1
: polynomial matrix

P2
: polynomial or polynomial matrix

R,Q
: two polynomial matrices

Description

Element-wise euclidan division of the polynomial matrix `P1` by the polynomial `P2` or by the polynomial matrix `P2`. `Rij` is the matrix of remainders, `Qij` is the matrix of quotients and `P1ij = Qij*P2 + Qij` or `P1ij = Qij*P2ij + Qij`.

Examples

```
x=poly(0,'x');
p1=(1+x^2)*(1-x);p2=1-x;
[r,q]=pdiv(p1,p2)
p2*q-p1
p2=1+x;
[r,q]=pdiv(p1,p2)
p2*q+r-p1
```

See Also

ldiv , gcd

Name

pol2des — polynomial matrix to descriptor form

```
[N,B,C]=pol2des(Ds)
```

Parameters

Ds
: polynomial matrix

N, B, C
: three real matrices

Description

Given the polynomial matrix `Ds=D_0 +D_1 s +D_2 s^2 +... +D_k s^k`, `pol2des` returns three matrices `N, B, C`, with `N` nilpotent such that:

```
Ds = C (s*N-eye())^-1 B
```

Examples

```
s=poly(0,'s');
G=[1,s;1+s^2,3*s^3];[N,B,C]=pol2des(G);
G1=clean(C*inv(s*N-eye())*B),G2=numer(G1)
```

See Also

ss2des , tf2des

Authors

F.D.;

Name

pol2str — polynomial to string conversion

```
[str]=pol2str(p)
```

Parameters

p
: real polynomial

str
: character string

Description

converts polynomial to character string (utility function).

See Also

string , pol2tex

Name

polfact — minimal factors

```
[f]=polfact(p)
```

Parameters

p
: polynomial

f
: vector `[f0 f1 ... fn]` such that `p=prod(f)`

  f0
  : constant

  fi
  : polynomial

Description

`f=polfact(p)` returns the minimal factors of p i.e. `f=[f0 f1 ... fn]` such that `p=prod(f)`

See Also

lcm , cmndred , factors

Authors

S. Steer INRIA

Name

residu — residue

```
[V]=residu(P,Q1,Q2)
```

Parameters

P, Q1, Q2
: polynomials or matrix polynomials with real or complex coefficients.

Description

`V=residu(P,Q1,Q2)` returns the matrix `V` such that `V(i,j)` is the sum of the residues of the rational fraction `P(i,j)/(Q1(i,j)*Q2(i,j))` calculated at the zeros of `Q1(i,j)`.

`Q1(i,j)` and `Q2(i,j)` must not have any common root.

Examples

```
s=poly(0,'s');
H=[s/(s+1)^2,1/(s+2)];N=numer(H);D=denom(H);
w=residu(N.*horner(N,-s),D,horner(D,-s));  //N(s) N(-s) / D(s) D(-s)
sqrt(sum(w))  //This is H2 norm
h2norm(tf2ss(H))
//
p=(s-1)*(s+1)*(s+2)*(s+10);a=(s-5)*(s-1)*(s*s)*((s+1/2)**2);
b=(s-3)*(s+2/5)*(s+3);
residu(p,a,b)+531863/4410    //Exact
z=poly(0,'z');a=z^3+0.7*z^2+0.5*z-0.3;b=z^3+0.3*z^2+0.2*z+0.1;
atild=gtild(a,'d');btild=gtild(b,'d');
residu(b*btild,z*a,atild)-2.9488038   //Exact
a=a+0*%i;b=b+0*%i;
real(residu(b*btild,z*a,atild)-2.9488038) //Complex case
```

See Also

pfss , bdiag , roots , poly , gtild

Authors

F.Delebecque INRIA

Name

roots — roots of polynomials

```
[x]=roots(p)
[x]=roots(p,'e')
```

Parameters

p
: polynomial with real or complex coefficients or vector of the polynomial coefficients in decreasing degree order (Matlab compatibility).

Description

`x=roots(p)` returns in the complex vector `x` the roots of the polynomial `p`. For real polynomials of degree <=100 the fast RPOLY algorithm (based on Jenkins-Traub method) is used. In the other cases the roots are computed as the eigenvalues of the associated companion matrix. Use `x=roots(p,'e')` to force this algorithm in any cases.

Examples

```
p=poly([0,10,1+%i,1-%i],'x');
roots(p)
A=rand(3,3);roots(poly(A,'x'))    // Evals by characteristic polynomial
spec(A)
```

See Also

poly, spec, companion

Authors

Serge Steer (INRIA)

References

The RPOLY algorithm is described in "Algorithm 493: Zeros of a Real Polynomial", ACM TOMS Volume 1, Issue 2 (June 1975), pp. 178-189

Jenkins, M. A. and Traub, J. F. (1970), A Three-Stage Algorithm for Real Polynomials Using Quadratic Iteration, SIAM J. Numer. Anal., 7(1970), 545-566.

Jenkins, M. A. and Traub, J. F. (1970), Principles for Testing Polynomial Zerofinding Programs. ACM TOMS 1, 1 (March 1975), pp. 26-34

Used Functions

The rpoly.f source codes can be found in the directory routines/control of a Scilab source distribution. In the case where the companion matrix is used, the eigenvalue computation is perfomed using DGEEV and ZGEEV LAPACK codes.

Name

rowcompr — row compression of polynomial matrix

```
[X,rk,Ac]=rowcompr(A)
```

Parameters

A
: polynomial matrix

Y
: square polynomial matrix (left unimodular basis)

rk
: normal rank of A

Ac
: Ac=X*A, polynomial matrix

Description

row compression of polynomial matrix A.

X is a left polynomial unimodular basis which row compressed thee rows of A. rk is the normal rank of A.

Warning: elimination of "small" terms (use with care!).

See Also

colcompr

Name

sfact — discrete time spectral factorization

```
F=sfact(P)
```

Parameters

P
: real polynomial matrix

Description

Finds F, a spectral factor of P. P is a polynomial matrix such that each root of P has a mirror image w.r.t the unit circle. Problem is singular if a root is on the unit circle.

`sfact(P)` returns a polynomial matrix `F(z)` which is antistable and such that

```
P = F(z)* F(1/z) *z^n
```

For scalar polynomials a specific algorithm is implemented. Algorithms are adapted from Kucera's book.

Examples

```
//Simple polynomial example
z=poly(0,'z');
p=(z-1/2)*(2-z)
w=sfact(p);
w*numer(horner(w,1/z))
//matrix example
F1=[z-1/2,z+1/2,z^2+2;1,z,-z;z^3+2*z,z,1/2-z];
P=F1*gtild(F1,'d');  //P is symmetric
F=sfact(P)
roots(det(P))
roots(det(gtild(F,'d')))  //The stable roots
roots(det(F))             //The antistable roots
clean(P-F*gtild(F,'d'))
//Example of continuous time use
s=poly(0,'s');
p=-3*(s+(1+%i))*(s+(1-%i))*(s+0.5)*(s-0.5)*(s-(1+%i))*(s-(1-%i));p=real(p);
//p(s) = polynomial in s^2 , looks for stable f such that p=f(s)*f(-s)
w=horner(p,(1-s)/(1+s));  // bilinear transform w=p((1-s)/(1+s))
wn=numer(w);              //take the numerator
fn=sfact(wn);f=numer(horner(fn,(1-s)/(s+1))); //Factor and back transform
f=f/sqrt(horner(f*gtild(f,'c'),0));f=f*sqrt(horner(p,0));      //normalization
roots(f)    //f is stable
clean(f*gtild(f,'c')-p)    //f(s)*f(-s) is p(s)
```

See Also

gtild , fspecg

Name

simp — rational simplification

```
[N1,D1]=simp(N,D)
H1=simp(H)
```

Parameters

N,D
: real polynomials or real matrix polynomials

H
: rational matrix (i.e matrix with entries `n/d` ,`n` and `d` real polynomials)

Description

`[n1,d1]=simp(n,d)` calculates two polynomials `n1` and `d1` such that `n1/d1 = n/d`.

If `N` and `D` are polynomial matrices the calculation is performed element-wise.

`H1=simp(H)` is also valid (each entry of `H` is simplified in `H1`).

Caution:

-no threshold is given i.e. `simp` cannot forces a simplification.

-For linear dynamic systems which include integrator(s) simplification changes the static gain. (`H(0)` for continuous systems or `H(1)` for discrete systems)

-for complex data, `simp` returns its input(s).

-rational simplification is called after nearly each operations on rationals. It is possible to toggle simplification on or off using `simp_mode` function.

Examples

```
s=poly(0,'s');
[n,d]=simp((s+1)*(s+2),(s+1)*(s-2))

simp_mode(%F);hns=s/s
simp_mode(%T);hns=s/s
```

See Also

roots , trfmod , poly , clean , simp_mode

Name

simp_mode — toggle rational simplification

```
mod=simp_mode()
simp_mode(mod)
```

Parameters

mod
: a boolean

Description

rational simplification is called after nearly each operations on rationals. It is possible to toggle simplification on or off using `simp_mode` function.

`simp_mod(%t)` set rational simplification mode on

`simp_mod(%f)` set rational simplification mode off

`mod=simp_mod()` returns in mod the current rational simplification mode

Examples

```
s=poly(0,'s');
mod=simp_mode()
simp_mode(%f);hns=s/s
simp_mode(%t);hns=s/s
simp_mode(mod);
```

See Also

simp

Name

sylm — Sylvester matrix

```
[S]=sylm(a,b)
```

Parameters

a,b
: two polynomials

S
: matrix

Description

`sylm(a,b)` gives the Sylvester matrix associated to polynomials `a` and `b`, i.e. the matrix `S` such that:

`coeff( a*x + b*y )' = S * [coeff(x)';coeff(y)']`.

Dimension of `S` is equal to `degree(a)+degree(b)`.

If `a` and `b` are coprime polynomials then

`rank(sylm(a,b))=degree(a)+degree(b))` and the instructions

```
u = sylm(a,b) \ eye(na+nb,1)
x = poly(u(1:nb),'z','coeff')
y = poly(u(nb+1:na+nb),'z','coeff')
```

compute Bezout factors `x` and `y` of minimal degree such that `a*x+b*y = 1`

Name

systmat — system matrix

```
[Sm]=systmat(Sl);
```

Parameters

Sl
: linear system (`syslin` list) or descriptor system

Sm
: matrix pencil

Description

System matrix of the linear system `Sl` (`syslin` list) in state-space form (utility function).

```
Sm = [-sI + A   B;
     [    C     D]
```

For a descriptor system (`Sl=list('des',A,B,C,D,E)`), `systmat` returns:

```
Sm = [-sE + A   B;
     [    C     D]
```

See Also

ss2des , sm2des , sm2ss

Randlib

Name

grand — Random number generator(s)

```
Y=grand(m, n, dist_type [,p1,...,pk])
Y=grand(X, dist_type [,p1,...,pk])
Y=grand(n, dist_type [,p1,...,pk])
S=grand(action [,q1,....,ql])
```

Parameters

m, n
: integers, size of the wanted matrix `Y`

X
: a matrix whom only the dimensions (say `m x n`) are used

dist_type
: a string given the distribution which (independants) variates are to be generated ('bin', 'nor', 'poi', etc ...)

p1, ..., pk
: the parameters (reals or integers) required to define completly the distribution `dist_type`

Y
: the resulting `m x n` random matrix

action
: a string given the action onto the base generator(s) ('setgen' to change the current base generator, 'getgen' to retrieve the current base generator name, 'getsd' to retrieve the state (seeds) of the current base generator, etc ...)

q1, ..., ql
: the parameters (generally one string) needed to define the action

S
: output of the action (generaly a string or a real column vector)

Description

This function may be used to generate random numbers from various distributions. In this case you must apply one of the `three first forms` of the possible calling sequences to get an `m x n` matrix. The two firsts are equivalent if `X` is a `m x n` matrix, and the third form corresponds to 'multivalued' distributions (e.g. multinomial, multivariate gaussian, etc...) where a sample is a column vector (says of dim `m`) and you get then `n` such random vectors (as an `m x n` matrix). `The last form` is used to undertake various manipulations onto the base generators like changing the base generator (since v 2.7 you may choose between several base generators), changing or retrieving its internal state (seeds), etc ... These base generators give random integers following a uniform distribution on a large integer interval (lgi), all the others distributions being gotten from it (in general via a scheme lgi -> U([0,1)) -> wanted distribution).

Getting random numbers from a given distribution

beta
: `Y=grand(m,n,'bet',A,B)` generates random variates from the beta distribution with parameters `A` and `B`. The density of the beta is (`0 < x < 1`) :

```
          A-1    B-1
         x   (1-x)   / beta(A,B)
```

A and B must be reals >10^(-37). Related function(s) : cdfbet.

binomial
: Y=grand(m,n,'bin',N,p) generates random variates from the binomial distribution with parameters N (positive integer) and p (real in [0,1]) : number of successes in N independant Bernouilli trials with probability p of success. Related function(s) : binomial, cdfbin.

negative binomial
: Y=grand(m,n,'nbn',N,p) generates random variates from the negative binomial distribution with parameters N (positive integer) and p (real in (0,1)) : number of failures occurring before N successes in independant Bernouilli trials with probability p of success. Related function(s) : cdfnbn.

chisquare
: Y=grand(m,n,'chi', Df) generates random variates from the chisquare distribution with Df (real > 0.0) degrees of freedom. Related function(s) : cdfchi.

non central chisquare
: Y=grand(m,n,'nch',Df,Xnon) generates random variates from the non central chisquare distribution with Df degrees of freedom (real >= 1.0) and noncentrality parameter Xnonc (real >= 0.0). Related function(s) : cdfchn.

exponential
: Y=grand(m,n,'exp',Av) generates random variates from the exponential distribution with mean Av (real >= 0.0).

F variance ratio
: Y=grand(m,n,'f',Dfn,Dfd) generates random variates from the F (variance ratio) distribution with Dfn (real > 0.0) degrees of freedom in the numerator and Dfd (real > 0.0) degrees of freedom in the denominator. Related function(s) : cdff.

non central F variance ratio
: Y=grand(m,n,'nf',Dfn,Dfd,Xnon) generates random variates from the noncentral F (variance ratio) distribution with Dfn (real >= 1) degrees of freedom in the numerator, and Dfd (real > 0) degrees of freedom in the denominator, and noncentrality parameter Xnonc (real >= 0). Related function(s) : cdffnc.

gamma
: Y=grand(m,n,'gam',shape,scale) generates random variates from the gamma distribution with parameters shape (real > 0) and scale (real > 0). The density of the gamma is :

```
              shape  (shape-1)   -scale x
       scale       x          e          /  gamma(shape)
```

Related function(s) : gamma, cdfgam.

Gauss Laplace (normal)
: Y=grand(m,n,'nor',Av,Sd) generates random variates from the normal distribution with mean Av (real) and standard deviation Sd (real >= 0). Related function(s) : cdfnor.

multivariate gaussian (multivariate normal)
: Y=grand(n,'mn',Mean,Cov) generates n multivariate normal random variates ; Mean must be a m x 1 matrix and Cov a m x m symetric positive definite matrix (Y is then a m x n matrix).

geometric
: `Y=grand(m,n,'geom', p)` generates random variates from the geometric distribution with parameter `p` : number of Bernouilli trials (with probability succes of `p`) until a succes is met. `p` must be in `[pmin,1]` (with `pmin = 1.3 10^(-307)`).

`Y` contains positive real numbers with integer values, with are the "number of trials to get a success".

markov
: `Y=grand(n,'markov',P,x0)` generate `n` successive states of a Markov chain described by the transition matrix `P`. Initial state is given by `x0`. If `x0` is a matrix of size `m=size(x0,'*')` then `Y` is a matrix of size `m x n`. `Y(i,:)` is the sample path obtained from initial state `x0(i)`.

multinomial
: `Y=grand(n,'mul',nb,P)` generates `n` observations from the Multinomial distribution : class `nb` events in `m` categories (put `nb` "balls" in `m` "boxes"). `P(i)` is the probability that an event will be classified into category i. `P` the vector of probabilities is of size `m-1` (the probability of category `m` being `1-sum(P)`). `Y` is of size `m x n`, each column `Y(:,j)` being an observation from multinomial distribution and `Y(i,j)` the number of events falling in category `i` (for the `j` th observation) (`sum(Y(:,j)) = nb`).

Poisson
: `Y=grand(m,n,'poi',mu)` generates random variates from the Poisson distribution with mean `mu (real >= 0.0)`. Related function(s) : cdfpoi.

random permutations
: `Y=grand(n,'prm',vect)` generate `n` random permutations of the column vector (`m x 1`) `vect`.

uniform (def)
: `Y=grand(m,n,'def')` generates random variates from the uniform distribution over `[0,1)` (1 is never return).

uniform (unf)
: `Y=grand(m,n,'unf',Low,High)` generates random reals uniformly distributed in `[Low, High)`.

uniform (uin)
: `Y=grand(m,n,'uin',Low,High)` generates random integers uniformly distributed between `Low` and `High` (included). `High` and `Low` must be integers such that `(High-Low+1) < 2,147,483,561`.

uniform (lgi)
: `Y=grand(m,n,'lgi')` returns the basic output of the current generator : random integers following a uniform distribution over :

- `[0, 2^32 - 1]` for mt, kiss and fsultra
- `[0, 2147483561]` for clcg2
- `[0, 2^31 - 2]` for clcg4
- `[0, 2^31 - 1]` for urand.

Set/get the current generator and its state

Since Scilab-2.7 you have the possibility to choose between different base generators (which give random integers following the 'lgi' distribution, the others being gotten from it) :

mt
: the Mersenne-Twister of M. Matsumoto and T. Nishimura, period about `2^19937`, state given by an array of `624` integers (plus an index onto this array); this is the default generator.

kiss
: The Keep It Simple Stupid of G. Marsaglia, period about `2^123`, state given by `4` integers.

clcg2
: a Combined 2 Linear Congruential Generator of P. L'Ecuyer, period about `2^61`, state given by `2` integers ; this was the only generator previously used by grand (but slightly modified)

clcg4
: a Combined 4 Linear Congruential Generator of P. L'Ecuyer, period about `2^121`, state given by 4 integers ; this one is splitted in `101` different virtual (non over-lapping) generators which may be useful for different tasks (see 'Actions specific to clcg4' and 'Test example for clcg4').

urand
: the generator used by the scilab function rand, state given by `1` integer, period of `2^31` (based on theory and suggestions given in d.e. knuth (1969), vol 2. State). This is the faster of this list but a little outdated (don't use it for serious simulations).

fsultra
: a Subtract-with-Borrow generator mixing with a congruential generator of Arif Zaman and George Marsaglia, period more than `10^356`, state given by an array of 37 integers (plus an index onto this array, a flag (0 or 1) and another integer).

The differents actions common to all the generators, are:

action= 'getgen'
: `S=grand('getgen')` returns the current base generator (`S` is a string among 'mt', 'kiss', 'clcg2', 'clcg4', 'urand', 'fsultra'.

action= 'setgen'
: `grand('setgen',gen)` sets the current base generator to be `gen` a string among 'mt', 'kiss', 'clcg2', 'clcg4', 'urand', 'fsultra' (notes that this call returns the new current generator, ie gen).

action= 'getsd'
: `S=grand('getsd')` gets the current state (the current seeds) of the current base generator ; `S` is given as a column vector (of integers) of dimension `625` for mt (the first being an index in `[1,624]`), `4` for kiss, `2` for clcg2, `40` for fsultra, `4` for clcg4 (for this last one you get the current state of the current virtual generator) and `1` for urand.

action= 'setsd'
: `grand('setsd',S), grand('setsd',s1[,s2,s3,s4])` sets the state of the current base generator (the new seeds) :

  for mt
  : `S` is a vector of integers of dim `625` (the first component is an index and must be in `[1,624]`, the `624` last ones must be in `[0,2^32[`) (but must not be all zeros) ; a simpler initialisation may be done with only one integer `s1` (`s1` must be in `[0,2^32[`) ;

  for kiss
  : `4` integers `s1,s2, s3,s4` in `[0,2^32[` must be provided ;

  for clcg2
  : `2` integers `s1` in `[1,2147483562]` and `s2` in `[1,2147483398]` must be given ;

  for clcg4
  : `4` integers `s1` in `[1,2147483646]`, `s2` in `[1,2147483542]`, `s3` in `[1,2147483422]`, `s4` in `[1,2147483322]` are required ; `CAUTION` : with clcg4 you set the seeds of the current virtual generator but you may lost the synchronisation between

this one and the others virtuals generators (ie the sequence generated is not warranty to be non over-lapping with a sequence generated by another virtual generator)=> use instead the 'setall' option.

for urand
: `1` integer `s1` in `[0,2^31[` must be given.

for fsultra
: `S` is a vector of integers of dim `40` (the first component is an index and must be in `[0,37]`, the 2d component is a flag (0 or 1), the 3d an integer in [1,2^32[and the 37 others integers in [0,2^32[) ; a simpler (and recommanded) initialisation may be done with two integers `s1` and `s2` in `[0,2^32[`.

action= 'phr2sd'
: `Sd=grand('phr2sd', phrase)` given a `phrase` (character string) generates a `1 x 2` vector `Sd` which may be used as seeds to change the state of a base generator (initialy suited for clcg2).

Options specific to clcg4

The clcg4 generator may be used as the others generators but it offers the advantage to be splitted in several (`101`) virtual generators with non over-lapping sequences (when you use a classic generator you may change the initial state (seeds) in order to get another sequence but you are not warranty to get a complete different one). Each virtual generator corresponds to a sequence of `2^72` values which is further split into `V=2^31` segments (or blocks) of length `W=2^41`. For a given virtual generator you have the possibility to return at the beginning of the sequence or at the beginning of the current segment or to go directly at the next segment. You may also change the initial state (seed) of the generator `0` with the 'setall' option which then change also the initial state of the other virtual generators so as to get synchronisation (ie in function of the new initial state of gen `0` the initial state of gen `1..100` are recomputed so as to get `101` non over-lapping sequences.

action= 'setcgn'
: `grand('setcgn',G)` sets the current virtual generator for clcg4 (when clcg4 is set, this is the virtual (clcg4) generator number `G` which is used); the virtual clcg4 generators are numbered from `0,1,..,100` (and so `G` must be an integer in `[0,100]`) ; by default the current virtual generator is `0`.

action= 'getcgn'
: `S=grand('getcgn')` returns the number of the current virtual clcg4 generator.

action= 'initgn'
: `grand('initgn',I)` reinitializes the state of the current virtual generator

I = -1
sets the state to its initial seed

I = 0
sets the state to its last (previous) seed (i.e. to the beginning of the current segment)

I = 1
sets the state to a new seed `W` values from its last seed (i.e. to the beginning of the next segment) and resets the current segment parameters.

action= 'setall'
: `grand('setall',s1,s2,s3,s4)` sets the initial state of generator `0` to `s1,s2,s3,s4`. The initial seeds of the other generators are set accordingly to have synchronisation. For constraints on `s1, s2, s3, s4` see the 'setsd' action.

action= 'advnst'
: `grand('advnst',K)` advances the state of the current generator by `2^K` values and resets the initial seed to that value.

Test example for clcg4

An example of the need of the splitting capabilities of clcg4 is as follows. Two statistical techniques are being compared on data of different sizes. The first technique uses bootstrapping and is thought to be as accurate using less data than the second method which employs only brute force. For the first method, a data set of size uniformly distributed between 25 and 50 will be generated. Then the data set of the specified size will be generated and analyzed. The second method will choose a data set size between 100 and 200, generate the data and analyze it. This process will be repeated 1000 times. For variance reduction, we want the random numbers used in the two methods to be the same for each of the 1000 comparisons. But method two will use more random numbers than method one and without this package, synchronization might be difficult. With clcg4, it is a snap. Use generator 0 to obtain the sample size for method one and generator 1 to obtain the data. Then reset the state to the beginning of the current block and do the same for the second method. This assures that the initial data for method two is that used by method one. When both have concluded, advance the block for both generators.

See Also

rand

Authors

randlib
: The codes to generate sequences following other distributions than def, unf, lgi, uin and geom are from "Library of Fortran Routines for Random Number Generation", by Barry W. Brown and James Lovato, Department of Biomathematics, The University of Texas, Houston.

mt
: The code is the mt19937int.c by M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator", ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January, pp.3-30 1998.

kiss
: The code was given by G. Marsaglia at the end of a thread concerning RNG in C in several newsgroups (whom sci.math.num-analysis) "My offer of RNG's for C was an invitation to dance..." only kiss have been included in Scilab (kiss is made of a combinaison of severals others which are not visible at the scilab level).

clcg2
: The method is from P. L'Ecuyer but the C code is provided at the Luc Devroye home page (http://cgm.cs.mcgill.ca/~luc/rng.html).

clcg4
: The code is from P. L'Ecuyer and Terry H.Andres and provided at the P. L'Ecuyer home page (http://www.iro.umontreal.ca/~lecuyer/papers.html) A paper is also provided and this new package is the logical successor of an old 's one from : P. L'Ecuyer and S. Cote. Implementing a Random Number Package with Splitting Facilities. ACM Transactions on Mathematical Software 17:1,pp 98-111.

fsultra
: code from Arif Zaman (arif@stat.fsu.edu) and George Marsaglia (geo@stat.fsu.edu)

scilab packaging
: By Jean-Philippe Chancelier and Bruno Pincon

Scilab to Fortran

Name

sci2for — scilab function to Fortran routine conversion

```
txt=sci2for(fun,nam,vtps [,lvtps])
```

Parameters

fun
: Scilab function

nam
: character string, the name of generated subroutine

vtps
: list

lvtps
: list

txt
: string, text of the subroutine Fortran code

Description

The elements of the list `vtps` give the type and dimensions of the input variables of the calling sequence and `lvtps` optionally gives the type and dimensions of the output variables. This last parameter is usefull if type and/or dimension inference cannot be able to determine the desired values.

These lists are structured as described below:

```
vtps(i)=list(typ,row_dim,col_dim)
```

where :

typ
: is a character string giving the type of the variable :

"0"
: constant,integer vector or matrix

"1"
: constant,double precision vector or matrix

"10"
: character string

row_dim
: character string (row dimension)

col_dim
: character string (column dimension)

txt
: Fortran code

Generated code may use routines of scilab libraries and some others whose source code may be found in <SCIDIR>/util/sci2for.f

Remarks

This function is just a try. Only simple function may be translated. Many function calls have not yet Fortran equivalent, to add the translation of a new function call you may define a scilab function. whose name is f_<name of function>. see <SCIDIR>/macros/sci2for/f_*.sci files for examples.

The following keywords :

```
work,iwork,ierr
iw*  iiw*
ilbN   (N integer)
```

may not appear in the function code.

See Also

function

Scipad

Name

edit_error — opens in SciPad the source of the last recorded error

```
answ = edit_error(clearerror)
```

Parameters

clearerror
: boolean - if true the error condition is cleared, if false it is kept (as in lasterror)

answ
: a string stating which source file is open (or why no file was open)

Description

This function opens in SciPad the source of the function which caused the last recorded error, and highlights the offending line.

This function works only for functions which are defined in libraries, i.e. not for internal functions, nor with functions defined online, nor loaded with individual getf or getd. This is since Scilab presently retains only the path to libraries and not to individual function sources.

Correspondance between the function name foo and function filename foo.sci is tacitly assumed.

Examples

```
acosh abc
edit_error
```

See Also

scipad , lasterror , errclear

Authors

Enrico Segre

Name

scipad — Embedded Scilab text editor

```
scipad()
scipad(f1[,f2,...])
scipad f1 f2 ...
```

Parameters

f1, f2...
(strings or vectors of strings) file or directory pathnames

Description

Scipad is an embedded Scilab text editor written in Tcl/Tk. It can be started with a fresh text buffer pressing the "Editor" button on top of the main Scilab window, or from Scilab command line with the instruction `scipad()`, or it can open specific files if invoked with any of the calling sequences above.

The same invocation adds further files to an already opened Scipad. If any of the arguments is a directory pathname, a file chooser starting in that directory pops up, allowing (multiple) selection of files.

Scipad allows Windows like edition modes. Keyboard shortcuts are defined for most possible editing actions and reported by the menu entries.

Additionally, the following shortcuts are defined:

<F2>	Save file
<F5>	Save file and run it into Scilab
<F6>	Show previous buffer
<F7>	Show next buffer
<Control-F6>	Switch to previous visible buffer
<Control-F7>	Switch to next visible buffer
<double-click mouse-button1>	Select word
<triple-click mouse-button1>	Select line
<Shift-Control-mouse-button1>	Select a block
<mouse-button2>	Paste selection
<mouse-button3>	Popup edit menu, or debug menu if clicked during a debug session
<Shift-mouse-button3>	Popup Execute menu
<Control-mouse-button3>	Popup Options menu
<Shift-Control-mouse-button3>	Open the source of the library function under the pointer
<Control-plus>	Increase the font size
<Control-minus>	Decrease the font size
<double-button1> on a tile title	Maximize this tile
<double-button1> on a sash	Space sashes evenly (for this paned window)
<button2> on a tile title	Switch hidden files

The "Load Into Scilab" (Ctrl-lowercase-l) menu entry can be used to exec the file content into Scilab, while "Execute selection" (Ctrl-lowercase-y) passes the selected lines to the scilab shell using ScilabEval (i.e. execstr).

Debugger

Scipad includes a full featured debugger targeted to Scilab scripts and macros. The user can:

Set/remove breakpoints anywhere in the opened files.
: The breakpointed lines get pink background. No breakpoint can be set on empty, blank or commented lines. Insertion and deletion of breakpoints can be done either before the debug session starts, and during such a session. Breakpoints can have a condition, which is a generic expression written in Scilab language. When the execution process encounters a breakpoint, this breakpoint is said to be reached. When the breakpoint is reached and its associated conditional expression is true (or changed, depending on the user's selection), then the breakpoint is said to be hit. The hit count is the number of times the breakpoint has been hit. Execution stops at a breakpoint if the hit count satisfies a selectable break condition. A user interface dedicated to breakpoints is available to control their conditional expression, the hit count and the break condition.

Remove all breakpoints.
: This allows to quickly remove all the breakpoints from all currently opened files.

Configure execution.
: The user has to provide the function name to execute, its variable names, and variable values. To ease this step, a scan of the currently displayed buffer is implemented to look for functions defined in it, scan their names and variable names. All this is displayed to the user in a dialog for easy selection. First, the user has to select a function in a spinbox, then eventually one of its variables. Once a variable is selected, the user can set/change its value (another dialog pops). Variable values and names are displayed in listboxes. The user can also add a new variable or remove already defined variables. This is in case the user changes the input variables of his function in the file, and he does not want to scan the buffer again (which causes all the variable values to be reset to a null value). Note also that the varargin keyword is fully supported, and that variables that are not given a value by the user are ignored when Scipad launches the function for debug. If the current file contains main level code (i.e. executable code outside of a function definition), Scipad proposes to debug this file as a .sce file (see below).

Go to next breakpoint.
: Scilab executes the code, and stops at the next breakpoint or goes to the end of the file if there is no more breakpoints. The active breakpoint is highlighted in Scipad so that it can easily be identified.

Execute step by step, going into functions (step into).
: Scilab stops before execution of each line. The active stop position is highlighted in Scipad so that it can easily be identified. Every line of code in functions from opened files is taken into account, but Scipad does not search for or open files by itself in order to step into them. Lines with no executable code (blank or commented lines) are skipped.

Execute step by step, without going into ancillary functions (step over).
: Similar to step into, but ancillaries are executed at once without stepping into them. However, if the user has set a breakpoint in an ancillary, Scilab will nevertheless stop at this breakpoint.

Execute step by step, starting back from the return point of the current function (step out).
: Scilab executes instructions until the function returns from the current context, i.e. the next stop occurs just after the current function has returned. However, if the user has set a breakpoint in the

current function or in an ancillary, Scilab will nevertheless stop at this breakpoint. Lines with no executable code (blank or commented lines) are skipped.

Run execution up to the next return point.

Scilab executes instructions until the next return point is reached. It stops just before executing the line that will make the current nest level to return. If the user has set breakpoints in the current function or in an ancillary, Scilab will skip them and stop only when the return point is reached. The list of exit points for the current function includes the line containing the endfunction keyword corresponding to the function declaration line (there can be only one such line in Scilab, no multiple "endfunction" for one "function"), but also possibly multiple "return" and "resume" statements.

Run execution up to the cursor position.

Scilab executes instructions until the cursor position is reached. If the user has set breakpoints in the current function or in an ancillary, Scilab will skip them and stop only when the cursor position is reached. Lines with no executable code (blank or commented lines) are also skipped: if the cursor is in such a line, Scilab will stop just before executing the next line carrying executable code.

Continue ignoring any breakpoint.

Finish execution in Scilab as if there was no breakpoint at all.

Break execution.

Scilab pauses execution. This is useful to check out where a long script is stuck, e.g. in case of an endless loop.

Abort debug.

Abort execution in Scilab and cancel the current debug session in Scipad.

When one of the run commands above is triggered for the first time, Scipad launches execution, i.e. it execs the currently displayed buffer as well as all the opened buffers that contain functions, sends to Scilab the setbpt instructions relative to all the breakpoints that have been set, and executes the selected function with the input variable values provided during the configure execution step. Then the execution is automagically stopped by Scilab according to the debug command that was launched. A new debug command can then be executed.

At any time during the debug, a **watch window** can be displayed on user request. It allows to monitor any variable value, or change a variable value during a breakpoint stop and relaunch execution with the modified value. A watchable variable can be part of a larger structure, for instance if A is a 20x20 matrix, the shorter sub-matrix A(2:4,7:9) can be watched. The user can also watch all local variables without having to input their name manually, or all locals and globals at the same time. It is also possible to "watch" generic expressions, i.e. enter a list of expressions that will be evaluated whenever execution stops, so that for instance an array can be plotted at each step. The watch window also displays the calling stack and contains a toolbar with the most useful commands from the debug menu.

In its current development state the debugger works well with functions, i.e. **pure .sci files**, but support of .sce files or mixed .sce/.sci files is however also fully implemented. Debug of .sce or mixed .sce/sci files makes use of the implementation for the .sci case after having automatically wrapped the code in a function/endfunction clause. The wrapper is automatically removed when the debug ends.

Due to technical limitations, **ancillary files of Scipad** cannot be debugged nor stepped into. During the configure step, Scipad detects if the user intends to configure one of its ancillaries for debugging,

and prevents from doing so. The list of reserved function names is then displayed in a message box. In case it is really needed to debug a Scipad ancillary, it is possible to try to change the name of the reserved function in order to debug a copy of it, but there are some catches to that, if the function calls itself other reserved ancillaries, or if the original file is still currently opened. Not only the name of the function in its definition line should be changed, but also any call to this function, and any call to the original function ancillaries if the original file is still open.

Remarks

Localisation:

Scipad menues and messages can be translated to several languages. The very first time Scipad is launched from a new Scilab installation, the language used by Scipad is the Scilab language. If the Scilab language is not available in Scipad, then the English fallback is used. Later, the localization in effect can be selected with the menu Options/Locale and is remembered across sessions. Currently, the supported languages are: "da_dk" (Danish), "de_de" (German), "en_us" (English), "es_es" (Spanish), "fr_fr" (French), "it_it" (Italian), "no" (Norwegian), "pl" (Polish), "se" (Swedish), "zh_cn" (Chinese-simplified), "zh_tw" (Chinese-Taiwan).

Further languages can be added by creating the proper translation file and putting it in `SCI/modules/scipad/tcl/msg_files/`. If you plan to do such a job, please check the file `SCI/modules/scipad/tcl/msg_files/AddingTranslations.txt` for detailed instructions, and consider to contribute it to the community.

Drag and drop:

DnD has been implemented in Scipad for moving around text, for dropping selected text from and to external applications, and for opening a file or a list of files.

Dragging one or more files from an explorer and dropping to Scipad will open the file(s) in Scipad. Doing the same with a directory will open recursively all the directory contents (beware!)

For text selected within the Scipad window, the possible actions are move (just use mouse button-1) and copy (Control button-1). Text selections can be moved or copied also between different Scipad subpanes, when tiling is active.

Drag and drop capabilities in Scipad rely on the TkDnD package (http://sourceforge.net/projects/tkdnd). Presence of this package should be automatically detected by Scipad, enabling the corresponding features at that time. Windows and linux-i386 binary versions of Scilab are currently shipped with TkDnD. If not, here are some installation instructions:

First of all, please note that tkdnd1.0 shall be used. An alpha version of tkdnd2 exists but shouldn't be used with Scipad. Details about reasons for this can be read at http://bugzilla.scilab.org/show_bug.cgi?id=2998#c4

Windows platforms: Download the full package (tkdnd-1.0a2.tar.gz), and uncompress it somewhere. Copy the content of lib\tkdnd and paste it into `SCI\modules\tclsci\tcl\tk<version>\tkdnd`. That's all!

linux-i386 platforms: Download the rpm package (tkdnd-1.0-b2.i386.rpm). Install it with rpm -U (may have to force --nodeps if it doesn't recognize an existing Tcl installation). If you have a source version of Scilab and an installation of Tcl/Tk, check where they are installed (e.g. `/usr/share/`) and move the newly created directory `/usr/lib/tkdnd1.0/` to there. If you have a binary version of Scilab, move `tkdnd1.0/` to `SCI/modules/tclsci/tcl/`, where the supplied Tcl/Tk binaries are.

Bugs:

There are still a few... Details can be found in file `SCI/modules/scipad/BUGS`. Officially reported bugs are filed in the Bugzilla http://bugzilla.scilab.org and can be easily retrieved by filtering entries wrt the "Scipad Editor" element.

Additional features in Scipad and most recent developments:

Scipad should run on Tcl/Tk 8.4.6 or higher. Scipad however offers a handful of quite handy features that are available as soon as Tcl/Tk 8.5 is running in its background. For instance, peer text widgets are available from Tk 8.5, and this capability is used in Scipad to allow for displaying more than one contiguous area of an opened file at a time in tile mode. Scilab 5 is currently shipped with Tcl/Tk 8.5 (at least on Windows). Should you need to upgrade from Tcl/Tk8.4, instructions about how to do this can be found on the Scilab wiki: http://wiki.scilab.org/Linking_Scilab_with_Tcl/Tk_8.5 .

Examples

```
scipad SCI/etc/scilab.start
```

See Also

edit, manedit, edit_error

Authors

Scipad is derived from tknotepad written by Joseph Acosta;
Mathieu Philippe, INRIA, 2001;
Enrico Segre, Weizmann Institute, 2003-2006;
Francois Vogel, 2004-2009.

Shell

Name

clc — Clear Command Window

```
clc([nblines])
```

Parameters

nblines
: a double value

Description

`clc()` clears all input and output from the Command Window.

After using `clc()`, you cannot use the scroll bar to see the history of functions, but still can use the up arrow to recall statements from the command history.

`clc(nblines)` clears `nblines` above cursor current line and move cursor up to this line.

Note that `clc([nblines])` cannot be used under Unix/Linux platforms when Scilab used in no window mode.

See Also

tohome

Authors

V.C.

Name

lines — rows and columns used for display

```
lines([nl [,nc]])
ncl=lines()
```

Parameters

nl : an integer, the number of lines for vertical paging control. If 0
no vertical paging control is done.

nc
an integer, the number of column of output. Used for formatting output

ncl
a 1x2 vector [nc,nl]

Description

`lines` handles Scilab display paging.

`lines()` returns the vector `[# columns, # rows]` currently used by Scilab for displaying the results.

`lines(nl)` sets the number of displayed lines (before user is asked for more) to `nl`.

`lines(0)` disables vertical paging

`lines(nl,nc)` changes also the size of the output to `nc` columns.

When Scilab is launched without -nw option, the `lines` parameters are automatically set according to the output window size, these parameters are also automaticaly modified when the window is resized.

See Also

disp , print

Name

prompt — Get/Set current prompt

```
currentprompt = prompt()
prompt(userprompt)
```

Parameters

currentprompt
: String: current prompt returned as a character string.

userprompt
: String: prompt to display for next user input. Then current prompt will be used again.

Description

`currentprompt = prompt()` gets the current prompt.

`prompt(userprompt)` sets the prompt.

See Also

pause , input

Authors

A.C.

Name

tohome — Move the cursor to the upper left corner of the Command Window

```
tohome()
```

Description

`tohome()` moves the cursor to the upper-left corner of the Command Window and clears the screen.

You can use the scroll bar to see the history of previous functions.

Note that `tohome()` cannot be used under Windows platforms when Scilab used in no window mode.

See Also

clc

Authors

V.C.

Signal Processing

Name

Signal — Signal manual description

Filters

analpf
: analog low-pass filter

buttmag
: squared magnitude response of a Butterworth filter

casc
: creates cascade realization of filter

cheb1mag
: square magnitude response of a type 1 Chebyshev filter

cheb2mag
: square magnitude response of a type 1 Chebyshev filter

chepol
: recursive implementation of Chebychev polynomial

convol
: convolution of 2 discrete series

ell1 mag
: squared magnitude of an elliptic filter

eqfir
: minimax multi-band, linear phase, FIR filter

eqiir
: design of iir filter

faurre
: optimal lqg filter.

lindquis
: optimal lqg filter lindquist algorithm

ffilt
: FIR low-pass,high-pass, band-pass, or stop-band filter

filter
: compute the filter model

find_freq
: parameter compatibility for elliptic filter design

findm
: for elliptic filter design

frmag
: magnitude of the frequency responses of FIR and IIR filters.

fsfirlin
: design of FIR, linear phase (frequency sampling technique)

fwiir
: optimum design of IIR filters in cascade realization,

iir
: designs an iir digital filter using analog filter designs.

iirgroup
: group delay of iir filter

iirlp
: Lp IIR filters optimization

group
: calculate the group delay of a digital filter

remezb
: minimax approximation of a frequency domain magnitude response.

kalm
: Kalman update and error variance

lev
: resolve the Yule-Walker equations :

levin
: solve recursively Toeplitz system (normal equations)

srfaur
: square-root algorithm for the algebraic Riccati equation.

srkf
: square-root Kalman filter algorithm

sskf
: steady-state Kalman filter

system
: generates the next observation given the old state

trans
: transformation of standardized low-pass filter into low-pass, high-pass, band-pass, stop-band.

wfir
: linear-phase windowed FIR low-pass, band-pass, high-pass, stop-band

wiener
: Wiener estimate (forward-backward Kalman filter formulation)

wigner
: time-frequency wigner spectrum of a signal.

window
: calculate symmetric window

zpbutt
: Butterworth analog filter

zpch1
: poles of a type 1 Chebyshev analog filter

zpch2
: poles and zeros of a type 2 Chebyshev analog filter

zpell
: poles and zeros of prototype lowpass elliptic filter

Spectral estimation

corr
: correlation coefficients

cspect
: spectral estimation using the modified periodogram method.

czt
: chirp z-transform algorithm

intdec
: change the sampling rate of a 1D or 2D signal

mese
: calculate the maximum entropy spectral estimate

pspect
: auto and cross-spectral estimate

wigner
: Wigner-Ville time/frequency spectral estimation

Transforms

dft
: discrete Fourier transform

fft
: fast flourier transform

hilb
: Hilbert transform centred around the origin.

hank
: hankel matrix of the covariance sequence of a vector process

mfft
: fft for a multi-dimensional signal

Identification

lattn,lattp
: recursive solution of normal equations

phc
: State space realisation by the principal hankel component approximation method,

rpem
: identification by the recursive prediction error method

Miscellaneous

lgfft
: computes p = ceil (log_2(x))

sinc
: calculate the function sin(2*pi*fl*t)/(pi*t)

sincd
: calculates the function Sin(N*x)/Sin(x)

%k
: Jacobi's complete elliptic integral

%asn
: .TP the elliptic integral :

%sn
: Jacobi 's elliptic function with parameter m

bilt
: bilinear transform or biquadratic transform.

jmat
: permutes block rows or block columns of a matrix

Name

analpf — create analog low-pass filter

```
[hs,pols,zers,gain]=analpf(n,fdesign,rp,omega)
```

Parameters

n
: positive integer : filter order

fdesign
: string : filter design method : `'butt'` or `'cheb1'` or `'cheb2'` or `'ellip'`

rp
: 2-vector of error values for cheb1, cheb2 and ellip filters where only `rp(1)` is used for cheb1 case, only `rp(2)` is used for cheb2 case, and `rp(1)` and `rp(2)` are both used for ellip case. `0<rp(1),rp(2)<1`

  - for cheb1 filters `1-rp(1)<ripple<1` in passband
  - for cheb2 filters `0<ripple<rp(2)` in stopband
  - for ellip filters `1-rp(1)<ripple<1` in passband `0<ripple<rp(2)` in stopband

omega
: cut-off frequency of low-pass filter in Hertz

hs
: rational polynomial transfer function

pols
: poles of transfer function

zers
: zeros of transfer function

gain
: gain of transfer function

Description

Creates analog low-pass filter with cut-off frequency at omega.

`hs=gain*poly(zers,'s')/poly(pols,'s')`

Examples

```
//Evaluate magnitude response of continuous-time system
hs=analpf(4,'cheb1',[.1 0],5)
fr=0:.1:15;
hf=freq(hs(2),hs(3),%i*fr);
hm=abs(hf);
```

```
plot(fr,hm)
```

Authors

C. B.

Name

bilt — bilinear or biquadratic transform SISO system given by a zero/poles representation

```
[npl,nzr,ngn] = bilt(pl,zr,gn,num,den)
```

Parameters

pl
: a vector, the poles of the given system.

zr
: a vector, the zeros of the given system.

num
: a polynomial with degree equal to the degree of `den`, the numerator of the transform.

den
: a polynomial with degree 1 or 2, the denominator of the transform.

npl
: a vector, the poles of the transformed system.

nzr
: a vector, the zeros of the transformed system.

ngn
: a scalar, the gain of the transformed system.

Description

function for calculating the gain poles and zeros which result from a bilinear transform or from a biquadratic transform. Used by the functions iir and trans.

Examples

```
Hlp=iir(3,'lp','ellip',[0.1 0],[.08 .03]);
pl=roots(Hlp.den);
zr=roots(Hlp.num);
gn=coeff(Hlp.num,degree(Hlp.num))/coeff(Hlp.den,degree(Hlp.den));
z=poly(0,'z');
a=0.3;
num=z-a;
den=1-a*z;
[npl,nzr,ngn] = bilt(pl,zr,gn,num,den)

Hlpt=ngn*poly(nzr,'z','r')/poly(npl,'z','r')
//comparison with horner
horner(Hlp,num/den)
```

Authors

Carey Bunks ;

See Also

iir, trans, horner

Name

buttmag — response of Butterworth filter

```
[h]=buttmag(order,omegac,sample)
```

Parameters

order
: integer : filter order

omegac
: real : cut-off frequency in Hertz

sample
: vector of frequency where `buttmag` is evaluated

h
: Butterworth filter values at sample points

Description

squared magnitude response of a Butterworth filter `omegac` = cutoff frequency ; `sample` = sample of frequencies

Examples

```
//squared magnitude response of Butterworth filter
h=buttmag(13,300,1:1000);
mag=20*log(h)'/log(10);
plot2d((1:1000)',mag,[2],"011"," ",[0,-180,1000,20])
```

Authors

F. D.

Name

casc — cascade realization of filter from coefficients

```
[cels]=casc(x,z)
```

Parameters

x
: (4xN)-matrix where each column is a cascade element, the first two column entries being the numerator coefficients and the second two column entries being the denominator coefficients

z
: string representing the cascade variable

cels
: resulting cascade representation

Description

Creates cascade realization of filter from a matrix of coefficients (utility function).

Examples

```
x=[1,2,3;4,5,6;7,8,9;10,11,12]
cels=casc(x,'z')
```

Name

cepstrum — cepstrum calculation

```
fresp = cepstrum(w,mag)
```

Parameters

w
: positive real vector of frequencies (rad/sec)

mag
: real vector of magnitudes (same size as `w`)

fresp
: complex vector

Description

`fresp = cepstrum(w,mag)` returns a frequency response `fresp(i)` whose magnitude at frequency `w(i)` equals `mag(i)` and such that the phase of `freq` corresponds to a stable and minimum phase system. `w` needs not to be sorted, but minimal entry should not be close to zero and all the entries of `w` should be different.

Examples

```
w=0.1:0.1:5;mag=1+abs(sin(w));
fresp=cepstrum(w,mag);
plot2d([w',w'],[mag(:),abs(fresp)])
```

See Also

frfit

Name

cheb1mag — response of Chebyshev type 1 filter

```
[h2]=cheb1mag(n,omegac,epsilon,sample)
```

Parameters

n
: integer : filter order

omegac
: real : cut-off frequency

epsilon
: real : ripple in pass band

sample
: vector of frequencies where `cheb1mag` is evaluated

h2
: Chebyshev I filter values at sample points

Description

Square magnitude response of a type 1 Chebyshev filter.

`omegac`=passband edge.

`epsilon`such that `1/(1+epsilon^2)`=passband ripple.

`sample`vector of frequencies where the square magnitude is desired.

Examples

```
//Chebyshev; ripple in the passband
n=13;epsilon=0.2;omegac=3;sample=0:0.05:10;
h=cheb1mag(n,omegac,epsilon,sample);
plot2d(sample,h)
xtitle('','frequencies','magnitude')
```

See Also

buttmag

Name

cheb2mag — response of type 2 Chebyshev filter

```
[h2]=cheb2mag(n,omegar,A,sample)
```

Parameters

n
: integer ; filter order

omegar
: real scalar : cut-off frequency

A
: attenuation in stop band

sample
: vector of frequencies where cheb2mag is evaluated

h2
: vector of Chebyshev II filter values at sample points

Description

Square magnitude response of a type 2 Chebyshev filter.

`omegar` = stopband edge, `sample` = vector of frequencies where the square magnitude `h2` is desired.

Examples

```
//Chebyshev; ripple in the stopband
n=10;omegar=6;A=1/0.2;sample=0.0001:0.05:10;
h2=cheb2mag(n,omegar,A,sample);
plot(sample,log(h2)/log(10),'frequencies','magnitude in dB')
//Plotting of frequency edges
minval=(-maxi(-log(h2)))/log(10);
plot2d([omegar;omegar],[minval;0],[2],"000");
//Computation of the attenuation in dB at the stopband edge
attenuation=-log(A*A)/log(10);
plot2d(sample',attenuation*ones(sample)',[5],"000")
```

See Also

cheb1mag

Name

chepol — Chebychev polynomial

```
[Tn]=chepol(n,var)
```

Parameters

n
: integer : polynomial order

var
: string : polynomial variable

Tn
: polynomial in the variable `var`

Description

Recursive implementation of Chebychev polynomial. `Tn=2*poly(0,var)*chepol(n-1,var)-chepol(n-2,var)` with `T0=1` and `T1=poly(0,var)`.

Examples

```
chepol(4,'x')
```

Authors

F. D.

Name

convol — convolution

```
[y]=convol(h,x)
[y,e1]=convol(h,x,e0)
```

Parameters

h
: a vector, first input sequence ("short" one)

x
: a vector, second input sequence ("long" one)

e0
: a vector,old tail to overlap add (not used in first call)

y
: a vector, the convolution.

e1
: new tail to overlap add (not used in last call)

Description

Calculates the convolution `y= h*x` of two discrete sequences by using the fft. The convolution is defined as follow:

$$y_k = \sum_j h_j * x_{k+1-j}$$

Overlap add method can be used.

USE OF OVERLAP ADD METHOD: For `x=[x1,x2,...,xNm1,xN]` First call is `[y1,e1]=convol(h,x1);` Subsequent calls : `[yk,ek]=convol(h,xk,ekm1)`; Final call : `[yN]=convol(h,xN,eNm1);` Finally `y=[y1,y2,...,yNm1,yN]`.

The algorithm based on the convolution definition is implemented for polynomial product: `y=convol(h,x)` is equivalent to `y=coeff(poly(h,'z','c')*poly(x,'z','c'))` but much more efficient if `x` is a "long" array.

Examples

```
x=1:3;
h1=[1,0,0,0,0];h2=[0,1,0,0,0];h3=[0,0,1,0,0];
x1=convol(h1,x),x2=convol(h2,x),x3=convol(h3,x),
convol(h1+h2+h3,x)
p1=poly(x,'x','coeff')
p2=poly(h1+h2+h3,'x','coeff')
p1*p2
```

See Also

corr, fft, pspect

Authors

F. D , C. Bunks Date 3 Oct. 1988; ;

Name

corr — correlation, covariance

```
[cov,Mean]=corr(x,[y],nlags)
[cov,Mean]=corr('fft',xmacro,[ymacro],n,sect)

[w,xu]=corr('updt',x1,[y1],w0)
[w,xu]=corr('updt',x2,[y2],w,xu)
...
[wk]=corr('updt',xk,[yk],w,xu)
```

Parameters

x
: a real vector

y
: a real vector, default value x.

nlags
: integer, number of correlation coefficients desired.

xmacro
: a scilab external (see below).

ymacro
: a scilab external (see below), default value xmacro

n
: an integer, total size of the sequence (see below).

sect
: size of sections of the sequence (see below).

xi
: a real vector

yi
: a real vector,default value xi.

cov
: real vector, the correlation coefficients

Mean
: real number or vector, the mean of x and if given y

Description

Computes

$$cov(m) = \sum_{k=1}^{n-m} (x(k) - xmean)\,(y(m+k) - ymean) * \frac{1}{n}$$

```
                k = 1
```

for m=0,..,`nlag-1` and two vectors `x=[x(1),..,x(n)]` `y=[y(1),..,y(n)]`

Note that if x and y sequences are differents corr(x,y,...) is different with corr(y,x,...)

Short sequences

`[cov,Mean]=corr(x,[y],nlags)` returns the first nlags correlation coefficients and Mean = `mean(x)` (mean of `[x,y]` if `y` is an argument). The sequence `x` (resp. `y`) is assumed real, and `x` and `y` are of same dimension n.

Long sequences

`[cov,Mean]=corr('fft',xmacro,[ymacro],n,sect)` Here `xmacro` is either

- a function of type `[xx]=xmacro(sect,istart)` which returns a vector `xx` of dimension `nsect` containing the part of the sequence with indices from `istart` to `istart+sect-1`.
- a fortran subroutine or C procedure which performs the same calculation. (See the source code of `dgetx` for an example). `n` = total size of the sequence. `sect` = size of sections of the sequence. `sect` must be a power of 2. `cov` has dimension `sect`. Calculation is performed by FFT.

Updating method

```
[w,xu]=corr('updt',x1,[y1],w0)
[w,xu]=corr('updt',x2,[y2],w,xu)
 ...
wk=corr('updt',xk,[yk],w,xu)
```

With this calling sequence the calculation is updated at each call to `corr`.

```
w0 = 0*ones(1,2*nlags);
nlags = power of 2.
```

`x1,x2,...` are parts of `x` such that `x=[x1,x2,...]` and sizes of `xi` a power of 2. To get `nlags` coefficients a final fft must be performed `c=fft(w,1)/n`; `cov=c(1nlags)` (`n` is the size of `x (y)`). Caution: this calling sequence assumes that `xmean = ymean = 0`.

Examples

```
x=%pi/10:%pi/10:102.4*%pi;
rand('seed');rand('normal');
y=[.8*sin(x)+.8*sin(2*x)+rand(x);.8*sin(x)+.8*sin(1.99*x)+rand(x)];
c=[];
for j=1:2,for k=1:2,c=[c;corr(y(k,:),y(j,:),64)];end;end;
c=matrix(c,2,128);cov=[];
for j=1:64,cov=[cov;c(:,(j-1)*2+1:2*j)];end;
rand('unif')
```

```
//
rand('normal');x=rand(1,256);y=-x;
deff('[z]=xx(inc,is)','z=x(is:is+inc-1)');
deff('[z]=yy(inc,is)','z=y(is:is+inc-1)');
[c,mxy]=corr(x,y,32);
x=x-mxy(1)*ones(x);y=y-mxy(2)*ones(y);  //centring
c1=corr(x,y,32);c2=corr(x,32);
norm(c1+c2,1)
[c3,m3]=corr('fft',xx,yy,256,32);
norm(c1-c3,1)
[c4,m4]=corr('fft',xx,256,32);
norm(m3,1),norm(m4,1)
norm(c3-c1,1),norm(c4-c2,1)
x1=x(1:128);x2=x(129:256);
y1=y(1:128);y2=y(129:256);
w0=0*ones(1:64);   //32 coeffs
[w1,xu]=corr('u',x1,y1,w0);w2=corr('u',x2,y2,w1,xu);
zz=real(fft(w2,1))/256;c5=zz(1:32);
norm(c5-c1,1)
[w1,xu]=corr('u',x1,w0);w2=corr('u',x2,w1,xu);
zz=real(fft(w2,1))/256;c6=zz(1:32);
norm(c6-c2,1)
rand('unif')
// test for Fortran or C external
//
deff('[y]=xmacro(sec,ist)','y=sin(ist:(ist+sec-1))');
x=xmacro(100,1);
[cc1,mm1]=corr(x,2^3);
[cc,mm]=corr('fft',xmacro,100,2^3);
[cc2,mm2]=corr('fft','corexx',100,2^3);
[maxi(abs(cc-cc1)),maxi(abs(mm-mm1)),maxi(abs(cc-cc2)),maxi(abs(mm-mm2))]

deff('[y]=ymacro(sec,ist)','y=cos(ist:(ist+sec-1))');
y=ymacro(100,1);
[cc1,mm1]=corr(x,y,2^3);
[cc,mm]=corr('fft',xmacro,ymacro,100,2^3);
[cc2,mm2]=corr('fft','corexx','corexy',100,2^3);
[maxi(abs(cc-cc1)),maxi(abs(mm-mm1)),maxi(abs(cc-cc2)),maxi(abs(mm-mm2))]
```

See Also

fft

Name

cspect — two sided cross-spectral estimate between 2 discrete time signals using the correlation method

```
[sm [,cwp]]=cspect(nlags,npoints,wtype,x [,y] [,wpar])
[sm [,cwp]]=cspect(nlags,npoints,wtype,nx [,ny] [,wpar])
```

Parameters

x
: vector, the data of the first signal.

y
: vector, the data of the second signal. If `y` is omitted it is supposed to be equal to `x` (auto-correlation). If it is present, it must have the same numer of element than `x.`

nx
: a scalar : the number of points in the `x` signal. In this case the segments of the x signal are loaded by a user defined function named `getx` (see below).

ny
: a scalar : the number of points in the `y` signal. In this case the segments of the `y` signal are loaded by a user defined function named `gety` (see below). If present `ny` must be equal to `nx`.

nlags
: number of correlation lags (positive integer)

npoints
: number of transform points (positive integer)

wtype
: The window type

  - `'re'`: rectangular
  - `'tr'`: triangular
  - `'hm'`: Hamming
  - `'hn'` : Hanning
  - `'kr'`: Kaiser,in this case the wpar argument must be given
  - `'ch'`: Chebyshev, in this case the wpar argument must be given

wpar
: optional parameters for `Kaiser and Chebyshev windows:`

  - 'kr': `wpar must be a strictly positive number`
  - 'ch': `wpar` must be a 2 element vector `[main_lobe_width,side_lobe_height]with 0<main_lobe_width<.5`, and `side_lobe_height>0`

sm
: The power spectral estimate in the interval `[0,1]` of the normalized frequencies. It is a row array of size `npoints`. The array is real in case of auto-correlation and complex in case of cross-correlation.

cwp
: the unspecified Chebyshev window parameter in case of Chebyshev windowing, or an empty matrix.

Description

Computes the cross-spectrum estimate of two signals `x` and `y` if both are given and the auto-spectral estimate of `x` otherwise. Spectral estimate obtained using the correlation method.

The cross spectrum of two signal x and y is defined to be

$$S_{xy}(\omega) = \frac{1}{N}\left(\sum_{n=0}^{N-1} x(n)e^{-i\omega n}\right)\left(\sum_{n=0}^{N-1} y(n)e^{i\omega n}\right)$$

The correlation method calculates the spectral estimate as the Fourier transform of a modified estimate of the auto/cross correlation function. This auto/cross correlation modified estimate consist of repeatedly calculating estimates of the autocorrelation function from overlapping sub-segments if the data, and then averaging these estimates to obtain the result.

The number of points of the window is `2*nlags-1.`

For batch processing, the `x` and `y` data may be read segment by segment using the `getx` and `gety` user defined functions. These functions have the following calling sequence:

`xk=getx(ns,offset)` and `yk=gety(ns,offset)` where `ns` is the segment size and `offset` is the index of the first element of the segment in the full signal.

Warning for Scilab version up to 5.0.2 the returned value was the modulus of the current one.

Reference

Oppenheim, A.V., and R.W. Schafer. Discrete-Time Signal Processing, Upper Saddle River, NJ: Prentice-Hall, 1999

Examples

```
rand('normal');rand('seed',0);
x=rand(1:1024-33+1);
//make low-pass filter with eqfir
nf=33;bedge=[0 .1;.125 .5];des=[1 0];wate=[1 1];
h=eqfir(nf,bedge,des,wate);
//filter white data to obtain colored data
h1=[h 0*ones(1:maxi(size(x))-1)];
x1=[x 0*ones(1:maxi(size(h))-1)];
hf=fft(h1,-1);   xf=fft(x1,-1);yf=hf.*xf;y=real(fft(yf,1));
sm=cspect(100,200,'tr',y);
smsize=maxi(size(sm));fr=(1:smsize)/smsize;
plot(fr,log(sm))
```

See Also

pspect, mese, corr

Authors

C. Bunks INRIA

Name

czt — chirp z-transform algorithm

```
[czx]=czt(x,m,w,phi,a,theta)
```

Parameters

x
: input data sequence

m
: czt is evaluated at `m` points in z-plane

w
: magnitude multiplier

phi
: phase increment

a
: initial magnitude

theta
: initial phase

czx
: chirp z-transform output

Description

chirp z-transform algorithm which calcultes the z-transform on a spiral in the z-plane at the points `[a*exp(j*theta)][w^kexp(j*k*phi)]` for `k=0,1,...,m-1`.

Examples

```
a=.7*exp(%i*%pi/6);
[ffr,bds]=xgetech(); //preserve current context
rect=[-1.2,-1.2*sqrt(2),1.2,1.2*sqrt(2)];
t=2*%pi*(0:179)/179;xsetech([0,0,0.5,1]);
plot2d(sin(t)',cos(t)',[2],"012",' ',rect)
plot2d([0 real(a)]',[0 imag(a)]',[3],"000")
xsegs([-1.0,0;1.0,0],[0,-1.0;0,1.0])
w0=.93*exp(-%i*%pi/15);w=exp(-(0:9)*log(w0));z=a*w;
zr=real(z);zi=imag(z);
plot2d(zr',zi',[5],"000")
xsetech([0.5,0,0.5,1]);
plot2d(sin(t)',cos(t)',[2],"012",' ',rect)
plot2d([0 real(a)]',[0 imag(a)]',[-1],"000")
xsegs([-1.0,0;1.0,0],[0,-1.0;0,1.0])
w0=w0/(.93*.93);w=exp(-(0:9)*log(w0));z=a*w;
zr=real(z);zi=imag(z);
plot2d(zr',zi',[5],"000")
xsetech(ffr,bds); //restore context
```

Authors

C. Bunks

Name

detrend — remove constant, linear or piecewise linear trend from a vector

```
y = detrend(x)
y = detrend(x,flag)
y = detrend(x,flag,bp)
```

Parameters

x
: vector or matrix of real or complex numbers (the signal to treat)

flag
: a string equal to "linear" (or "l") for linear or piecewise linear treatment or "constant" (or "c") for constant treatment.

bp
: the breakpoints to provide if you want a piecewise linear treatment.

y
: output, the signal x with the trend removed from it.

Description

This function removes the constant or linear or piecewise linear trend from a vector `x`. In general this can be useful before a fourier analysis. If `x` is matrix this function removes the trend of each column of x.

When `flag` = "constant" or "c" `detrend` removes the constant trend (simply the mean of the signal) and when `flag` = "linear" or "l" the function removes the linear trend. By adding a third argument `bp` it is possible to remove a continuous *piecewise linear* trend. Note that the "instants" of the signal `x` goes from 0 to m-1 (m = length(x) if x is a vector and m = size(x,1) in case x is a matrix). So the breakpoints `bp(i)` must be reals in *[0 m-1]* (breakpoints outside are simply removed from `bp` vector).

The trend is got by a least square fit of `x` on the appropriate function space.

Examples

```
// example #1
t = linspace(0,16*%pi,1000)';
x = -20 + t + 0.3*sin(0.5*t) + sin(t) + 2*sin(2*t) + 0.5*sin(3*t);
y = detrend(x);
xbasc()
plot2d(t,[x y],style=[2 5])
legend(["before detrend","after detrend"]);
xgrid()

// example #2
t = linspace(0,32*%pi,2000)';
x = abs(t-16*%pi) + 0.3*sin(0.5*t) + sin(t) + 2*sin(2*t) + 0.5*sin(3*t);
y = detrend(x,"linear",1000);
xbasc()
plot2d(t,[x y],style=[2 5])
legend(["before detrend","after detrend"]);
xgrid()
```

Authors

Bruno Pincon

Name

dft — discrete Fourier transform

```
[xf]=dft(x,flag);
```

Parameters

x
: input vector

flag
: indicates dft (flag=-1) or idft (flag=1)

xf
: output vector

Description

Function which computes dft of vector x.

Examples

```
n=8;omega = exp(-2*%pi*%i/n);
j=0:n-1;F=omega.^(j'*j);  //Fourier matrix
x=1:8;x=x(:);
F*x
fft(x,-1)
dft(x,-1)
inv(F)*x
fft(x,1)
dft(x,1)
```

See Also

fft

Authors

C. B.

Name

ell1mag — magnitude of elliptic filter

```
[v]=ell1mag(eps,m1,z)
```

Parameters

eps
: passband ripple=`1/(1+eps^2)`

m1
: stopband ripple=`1/(1+(eps^2)/m1)`

z
: sample vector of values in the complex plane

v
: elliptic filter values at sample points

Description

Function used for squared magnitude of an elliptic filter. Usually `m1=eps*eps/(a*a-1)`. Returns `v=real(ones(z)./(ones(z)+eps*eps*s.*s))` for `s=%sn(z,m1)`.

Examples

```
deff('[alpha,BeTa]=alpha_beta(n,m,m1)',...
'if 2*int(n/2)==n then, BeTa=K1; else, BeTa=0;end;...
alpha=%k(1-m1)/%k(1-m);')
epsilon=0.1;A=10;  //ripple parameters
m1=(epsilon*epsilon)/(A*A-1);n=5;omegac=6;
m=find_freq(epsilon,A,n);omegar = omegac/sqrt(m)
%k(1-m1)*%k(m)/(%k(m1)*%k(1-m))-n   //Check...
[alpha,Beta]=alpha_beta(n,m,m1)
alpha*%asn(1,m)-n*%k(m1)      //Check
sample=0:0.01:20;
//Now we map the positive real axis into the contour...
z=alpha*%asn(sample/omegac,m)+Beta*ones(sample);
plot(sample,ell1mag(epsilon,m1,z))
```

See Also

buttmag

Name

eqfir — minimax approximation of FIR filter

```
[hn]=eqfir(nf,bedge,des,wate)
```

Parameters

nf
: number of output filter points desired

bedge
: Mx2 matrix giving a pair of edges for each band

des
: M-vector giving desired magnitude for each band

wate
: M-vector giving relative weight of error in each band

hn
: output of linear-phase FIR filter coefficients

Description

Minimax approximation of multi-band, linear phase, FIR filter

Examples

```
hn=eqfir(33,[0 .2;.25 .35;.4 .5],[0 1 0],[1 1 1]);
[hm,fr]=frmag(hn,256);
plot(fr,hm),
```

Authors

C. B.

Name

eqiir — Design of iir filters

```
[cells,fact,zzeros,zpoles]=eqiir(ftype,approx,om,deltap,deltas)
```

Parameters

ftype
: filter type (`'lp','hp','sb','bp'`)

approx
: design approximation (`'butt','cheb1','cheb2','ellip'`)

om
: 4-vector of cutoff frequencies (in radians) `om=[om1,om2,om3,om4]`, `0 <= om1 <= om2 <= om3 <= om4 <= pi`. When `ftype`='lp' or 'hp', `om3` and `om4` are not used and may be set to 0.

deltap
: ripple in the passband. `0<= deltap <=1`

deltas
: ripple in the stopband. `0<= deltas <=1`

cells
: realization of the filter as second order cells

fact
: normalization constant

zzeros
: zeros in the z-domain

zpoles
: poles in the z-domain

Description

Design of iir filter based on syredi.

The filter obtained is `h(z)=fact`*product of the elements of `cells`.

That is `hz=fact*prod(cells.num)./prod(cells.den).`

Examples

```
[cells,fact,zzeros,zpoles]=eqiir('lp','ellip',[2*%pi/10,4*%pi/10],0.02,0.001)
h=fact*poly(zzeros,'z')/poly(zpoles,'z')
```

See Also

eqfir, iir, syredi

Name

faurre — filter computation by simple Faurre algorithm

```
[P,R,T]=faurre(n,H,F,G,R0)
```

Parameters

n
: number of iterations.

H, F, G
: estimated triple from the covariance sequence of `y`.

R0
: E(yk*yk')

P
: solution of the Riccati equation after n iterations.

R, T
: gain matrix of the filter.

Description

This function computes iteratively the minimal solution of the algebraic Riccati equation and gives the matrices `R` and `T` of the filter model. The algorithm tries to compute the solution P as the growing limit of a sequence of matrices Pn such that

```
                                          -1
Pn+1=F*Pn*F'+(G-F*Pn*h')*(R0-H*Pn*H')  *(G'-H*Pn*F')
       -1
P0=G*R0 *G'
```

Note that this method may not converge,especially when F has poles near the unit circle. Use preferably the srfaur function.

See Also

srfaur , lindquist , phc

Authors

G. Le V.

Name

ffilt — coefficients of FIR low-pass

```
[x]=ffilt(ft,n,fl,fh)
```

Parameters

ft
: filter type where `ft` can take the values

  "lp"
  : for low-pass filter

  "hp"
  : for high-pass filter

  "bp"
  : for band-pass filter

  "sb"
  : for stop-band filter

n
: integer (number of filter samples desired)

fl
: real (low frequency cut-off)

fh
: real (high frequency cut-off)

x
: vector of filter coefficients

Description

Get `n` coefficients of a FIR low-pass, high-pass, band-pass, or stop-band filter. For low and high-pass filters one cut-off frequency must be specified whose value is given in `fl`. For band-pass and stop-band filters two cut-off frequencies must be specified for which the lower value is in `fl` and the higher value is in `fh`

Authors

C. B.

Name

fft — fast Fourier transform.
ifft — fast Fourier transform.

```
x=fft(a ,-1) or x=fft(a)
x=fft(a,1) or x=ifft(a)
x=fft(a,-1,dim,incr)
x=fft(a,1,dim,incr)
```

Parameters

x
: real or complex vector. Real or complex matrix (2-dim fft)

a
: real or complex vector, matrix or multidimensionnal array.

dim
: integer

incr
: integer

Description

Short syntax

direct
: `x=fft(a,-1)` or `x=fft(a)` gives a direct transform.

single variate
: If `a` is a vector a single variate direct FFT is computed that is:

x(k)=sum over m from 1 to n of a(m)*exp(-2i*pi*(m-1)*(k-1)/n)

for k varying from 1 to n (n=size of vector `a`).

(the `-1` argument refers to the sign of the exponent..., NOT to "inverse"),

multivariate
: If `a` is a matrix or or a multidimensionnal array a multivariate direct FFT is performed.

inverse
: `a=fft(x,1)` or `a=ifft(x)`performs the inverse transform normalized by `1/n`.

single variate
: If `a` is a vector a single variate inverse FFT is computed

multivariate
: If `a` is a matrix or or a multidimensionnal array a multivariate inverse FFT is performed.

Long syntax for multidimensional FFT
: `x=fft(a,-1,dim,incr)` allows to perform an multidimensional fft.

If a is a real or complex vector implicitly indexed by `j1,j2,..,jp` i.e. `a(j1,j2,..,jp)` where `j1` lies in `1:dim(1)`, `j2` in `1:dim(2),...` one gets a p-variate FFT by calling p times `fft` as follows

```
incrk=1; x=a; for k=1:p x=fft(x ,-1,dim(k),incrk)
```

```
        incrk=incrk*dim(k) end
```

where `dimk` is the dimension of the current variable w.r.t which one is integrating and `incrk` is the increment which separates two successive `jk` elements in `a`.

In particular,if `a` is an mxn matrix, `x=fft(a,-1)` is equivalent to the two instructions:

`a1=fft(a,-1,m,1)` and `x=fft(a1,-1,n,m)`.

Examples

```
//Comparison with explicit formula
//---------------------------------
a=[1;2;3];n=size(a,'*');
norm(1/n*exp(2*%i*%pi*(0:n-1)'.*.(0:n-1)/n)*a -fft(a,1))
norm(exp(-2*%i*%pi*(0:n-1)'.*.(0:n-1)/n)*a -fft(a,-1))

//Frequency components of a signal
//---------------------------------
// build a noides signal sampled at 1000hz  containing to pure frequencies
// at 50 and 70 Hz
sample_rate=1000;
t = 0:1/sample_rate:0.6;
N=size(t,'*'); //number of samples
s=sin(2*%pi*50*t)+sin(2*%pi*70*t+%pi/4)+grand(1,N,'nor',0,1);

y=fft(s);
//the fft response is symetric we retain only the first N/2 points
f=sample_rate*(0:(N/2))/N; //associated frequency vector
n=size(f,'*')
clf()
plot2d(f,abs(y(1:n)))
```

See Also

corr

Name

fft2 — two-dimension fast Fourier transform

```
y=fft2(x)
y=fft2(x,n,m)
```

Parameters

x
: a vector/matrix/array (Real or Complex)

y
: a vector/matrix/array (Real or Complex)

m
: integer, number of rows.

n
: integer, number of columns.

Description

```
This functions performs the two-dimension discrete Fourier
    transform.
```

`y=fft2(x)` y and x have the same size

`y=fft2(x,m,n):` If `m` (respectively `n`) is less than the rows number (respectively columns) of `x` then the `x` rows number (resp. columns) is truncated, else if m (resp. `n`) is more than the rows number (resp. columns) of `x` then `x` rows are completed by zero (resp. columns) .

if `x` is a matrix then `y` is a matrix, if `x` is a hypermatrix then `y` is a hypermatrix, with the size of the first dimension of `y` is equal to `m`, the size of the second dimension of `y` is equal to `n`, the size of the ith dimension of `y` (for i>2, case hypermatrix) equal to the size of the ith dimension of `x`. (i.e size(y,1)=m, size(y,2)=n and size(y,i)=size(x,i) for i>2)

Examples

```
//Comparison with explicit formula
a=[1 2 3 ;4 5 6 ;7 8 9 ;10 11 12]
m=size(a,1)
n=size(a,2)
// fourier transform along the rows
for i=1:n
a1(:,i)=exp(-2*%i*%pi*(0:m-1)'.*.(0:m-1)/m)*a(:,i)
end
// fourier transform along the columns
for j=1:m
a2temp=exp(-2*%i*%pi*(0:n-1)'.*.(0:n-1)/n)*(a1(j,:)).'
a2(j,:)=a2temp.'
end
norm(a2-fft2(a))
```

See Also

fft

Name

fftshift — rearranges the fft output, moving the zero frequency to the center of the spectrum

```
y=fftshift(x [,job])
```

Parameters

x
: real or complex vector or matrix.

y
: real or complex vector or matrix.

job
: integer, dimension selection, or string 'all'

Description

if `x` results of an fft computation `y= fftshift(x)` or `y= fftshift(x,"all")` moves the zero frequency component to the center of the spectrum, which is sometimes a more convenient form.

If `x` is a vector of size n, y is the vector `x([n/2+1:n,1:n/2])`

If `x` is an `m` by `n` matrix `y` is the matrix `x([m/2+1:n,1:m/2],[n/2+1:n,1:n/2])`.

```
  [x11 x12]                  [x22 x21]
x=[       ]        gives   y=[       ]
  [x21 x22]                  [x12 x11]
```

`y= fftshift(x,n)` make the swap only along the `n`th dimension

Examples

```
//make a signal
t=0:0.1:1000;
x=3*sin(t)+8*sin(3*t)+0.5*sin(5*t)+3*rand(t);
//compute the fft
y=fft(x,-1);
//display
xbasc();
subplot(2,1,1);plot2d(abs(y))
subplot(2,1,2);plot2d(fftshift(abs(y)))

//make a 2D image
t=0:0.1:30;
x=3*sin(t')*cos(2*t)+8*sin(3*t')*sin(5*t)+..
  0.5*sin(5*t')*sin(5*t)+3*rand(t')*rand(t);
//compute the fft
y=fft(x,-1);
//display
xbasc();
```

```
xset('colormap',hotcolormap(256))
subplot(2,1,1);Matplot(abs(y))
subplot(2,1,2);Matplot(fftshift(abs(y)))
```

See Also

fft

Name

filt_sinc — samples of sinc function

```
[x]=filt_sinc(n,fl)
```

Parameters

n
: number of samples

fl
: cut-off frequency of the associated low-pass filter in Hertz.

x
: samples of the sinc function

Description

Calculate n samples of the function `sin(2*pi*fl*t)/(pi*t)` for `t=-(n-1)/2:(n-1)/2` (i.e. centred around the origin).

Examples

```
plot(filt_sinc(100,0.1))
```

See Also

sincd

Authors

C. B.;

Name

filter — filters a data sequence using a digital filter

```
[y,zf] = filter(num,den,x [,zi])
```

Parameters

num
: real vector : the coefficients of the filter numerator in decreasing power order, or a polynomial.

den
: real vector : the coefficients of the filter denominator in decreasing power order, or a polynomial.

x
: real row vector : the input signal

zi
: real row vector of length `max(length(a),length(b))-1`: the initial condition relative to a "direct form II transposed" state space representation. The default value is a vector filled with zeros.

y
: real row vector : the filtered signal.

zf
: real row vector : the final state. It can be used to filter a next batch of the input signal.

Description

This function filters a data sequence using a digital filter using a "direct form II transposed" implementation

Examples

```
```

References

Oppenheim, A. V. and R.W. Schafer. Discrete-Time Signal Processing, Englewood Cliffs, NJ: Prentice-Hall, 1989, pp. 311-312.

See Also

flts , rtitr , ltitr

Authors

Serge Steer, INRIA

Name

find_freq — parameter compatibility for elliptic filter design

```
[m]=find_freq(epsilon,A,n)
```

Parameters

epsilon
: passband ripple

A
: stopband attenuation

n
: filter order

m
: frequency needed for construction of elliptic filter

Description

Search for m such that `n=K(1-m1)K(m)/(K(m1)K(1-m))` with

`m1=(epsilon*epsilon)/(A*A-1)`;

If `m = omegar^2/omegac^2`, the parameters `epsilon,A,omegac,omegar` and `n` are then compatible for defining a prototype elliptic filter. Here, `K=%k(m)` is the complete elliptic integral with parameter `m`.

See Also

%k

Authors

F. D.

Name

findm — for elliptic filter design

```
[m]=findm(chi)
```

Description

Search for `m` such that `chi = %k(1-m)/%k(m)` (For use with `find_freq`).

See Also

%k

Authors

F. D.;

Name

frfit — frequency response fit

```
sys=frfit(w,fresp,order)
[num,den]=frfit(w,fresp,order)
sys=frfit(w,fresp,order,weight)
[num,den]=frfit(w,fresp,order,weight)
```

Parameters

w
: positive real vector of frequencies (Hz)

fresp
: complex vector of frequency responses (same size as `w`)

order
: integer (required order, degree of `den`)

weight
: positive real vector (default value `ones(w)`).

num,den
: stable polynomials

Description

`sys=frfit(w,fresp,order,weight)` returns a bi-stable transfer function `G(s)=sys=num/den`, of of given `order` such that its frequency response `G(w(i))` matches `fresp(i)`, i.e. `freq(num,den,%i*w)` should be close to `fresp`. `weight(i)` is the weight given to `w(i)`.

Examples

```
w=0.01:0.01:2;s=poly(0,'s');
G=syslin('c',2*(s^2+0.1*s+2), (s^2+s+1)*(s^2+0.3*s+1));
fresp=repfreq(G,w);
Gid=frfit(w,fresp,4);
frespfit=repfreq(Gid,w);
bode(w,[fresp;frespfit])
```

See Also

frep2tf , factors , cepstrum , mrfit , freq , calfrq

Name

frmag — magnitude of FIR and IIR filters

```
[xm,fr]=frmag(sys,npts)
[xm,fr]=frmag(num,den,npts)
```

Parameters

sys
: a single input, single output discrete transfer function, or a polynomial or the vector of polynomial coefficients, the filter.

num
: a polynomial or the vector of polynomial coefficients, the numerator of the filter

den
: a polynomial or the vector of polynomial coefficients, the denominator of the filter (the default value is 1).

npts
: integer, the number of points in frequency response.

xm
: vector of magnitude of frequency response at the points `fr`.

fr
: points in the normalized frequency domain where magnitude is evaluated.

Description

calculates the magnitude of the frequency responses of FIR and IIR filters. The filter description can be one or two vectors of coefficients, one or two polynomials, or a single output discrete transfer function.

the frequency discretisation is given by `fr=linspace(0,1/2,npts)`.

Authors

Carey Bunks.

Examples

```
hz=iir(3,'bp','cheb1',[.15 .25],[.08 .03]);
[hzm,fr]=frmag(hz,256);
plot(fr,hzm)
hz=iir(3,'bp','ellip',[.15 .25],[.08 .03]);
[hzm,fr]=frmag(hz,256);
plot(fr,hzm,'r')
```

See Also

iir, eqfir, repfreq, calfrq, phasemag

Name

fsfirlin — design of FIR, linear phase filters, frequency sampling technique

```
[hst]=fsfirlin(hd,flag)
```

Parameters

hd
: vector of desired frequency response samples

flag
: is equal to 1 or 2, according to the choice of type 1 or type 2 design

hst
: vector giving the approximated continuous response on a dense grid of frequencies

Description

function for the design of FIR, linear phase filters using the frequency sampling technique

Examples

```
//
//Example of how to use the fsfirlin macro for the design
//of an FIR filter by a frequency sampling technique.
//
//Two filters are designed : the first (response hst1) with
//abrupt transitions from 0 to 1 between passbands and stop
//bands; the second (response hst2) with one sample in each
//transition band (amplitude 0.5) for smoothing.
//
hd=[zeros(1,15) ones(1,10) zeros(1,39)];//desired samples
hst1=fsfirlin(hd,1);//filter with no sample in the transition
hd(15)=.5;hd(26)=.5;//samples in the transition bands
hst2=fsfirlin(hd,1);//corresponding filter
pas=1/prod(size(hst1))*.5;
fg=0:pas:.5;//normalized frequencies grid
plot2d([1 1].*.fg(1:257)',[hst1' hst2']);
// 2nd example
hd=[0*ones(1,15) ones(1,10) 0*ones(1,39)];//desired samples
hst1=fsfirlin(hd,1);//filter with no sample in the transition
hd(15)=.5;hd(26)=.5;//samples in the transition bands
hst2=fsfirlin(hd,1);//corresponding filter
pas=1/prod(size(hst1))*.5;
fg=0:pas:.5;//normalized frequencies grid
n=prod(size(hst1))
plot(fg(1:n),hst1);
plot2d(fg(1:n)',hst2',[3],"000");
```

See Also

ffilt , wfir

Authors

G. Le Vey

Name

group — group delay for digital filter

```
[tg,fr]=group(npts,a1i,a2i,b1i,b2i)
```

Parameters

npts
: integer : number of points desired in calculation of group delay

a1i
: in coefficient, polynomial, rational polynomial, or cascade polynomial form this variable is the transfer function of the filter. In coefficient polynomial form this is a vector of coefficients (see below).

a2i
: in coeff poly form this is a vector of coeffs

b1i
: in coeff poly form this is a vector of coeffs

b2i
: in coeff poly form this is a vector of coeffs

tg
: values of group delay evaluated on the grid fr

fr
: grid of frequency values where group delay is evaluated

Description

Calculate the group delay of a digital filter with transfer function h(z).

The filter specification can be in coefficient form, polynomial form, rational polynomial form, cascade polynomial form, or in coefficient polynomial form.

In the coefficient polynomial form the transfer function is formulated by the following expression

```
h(z)=prod(a1i+a2i*z+z**2)/prod(b1i+b2i*z+z^2)
```

Examples

```
z=poly(0,'z');
h=z/(z-.5);
[tg,fr]=group(100,h);
plot(fr,tg)
```

Authors

C. B.

Name

hank — covariance to hankel matrix

```
[hk]=hank(m,n,cov)
```

Parameters

m
: number of bloc-rows

n
: number of bloc-columns

cov
: sequence of covariances; it must be given as :[R0 R1 R2...Rk]

hk
: computed hankel matrix

Description

this function builds the hankel matrix of size `(m*d,n*d)` from the covariance sequence of a vector process

Examples

```
//Example of how to use the hank macro for
//building a Hankel matrix from multidimensional
//data (covariance or Markov parameters e.g.)
//
//This is used e.g. in the solution of normal equations
//by classical identification methods (Instrumental Variables e.g.)
//
//1)let's generate the multidimensional data under the form :
//  C=[c_0 c_1 c_2 .... c_n]
//where each bloc c_k is a d-dimensional matrix (e.g. the k-th correlation
//of a d-dimensional stochastic process X(t) [c_k = E(X(t) X'(t+k)], '
//being the transposition in scilab)
//
//we take here d=2 and n=64
//
c=rand(2,2*64)
//
//generate the hankel matrix H (with 4 bloc-rows and 5 bloc-columns)
//from the data in c
//
H=hank(4,5,c);
//
```

See Also

toeplitz

Authors

G. Le Vey

Name

hilb — FIR approximation to a Hilbert transform filter

```
xh=hilb(n [,wtype [,par]])
```

Parameters

n
: odd integer : number of points in filter

wtype
: string : window type `('re','tr','hn','hm','kr','ch')` (default `='re'`)

par
: window parameter for `wtype='kr' or 'ch'` default `par=[0 0]` see the function window for more help

xh
: Hilbert transform

Description

Returns the first n points of an FIR approximation to a Hilbert transform filter centred around the origin.

The FIR filter is designed by appropraitely windowing the ideal impulse response `h(n)=(2/(n*pi))*(sin(n*pi/2))^2` for `n` not equal 0 and `h(0)=0`.

An approximation to an analytic signal generator can be built by designing an FIR (Finite Impulse Response) filter approximation to the Hilbert transform operator. The analytic signal can then be computed by adding the appropriately time-shifted real signal to the imaginary part generated by the Hilbert filter.

References

`http://ieeexplore.ieee.org/iel4/78/7823/00330385.pdf?tp=&arnumber=330385&isnumber=7823`

A. Reilly, G. Frazer, and B. Boashash, "Analytic signal generation Tips and traps", IEEE Trans. Signal Processing, vol. 42, pp.3241-3245, Nov. 1994.

See Also

window , hilbert

Examples

```
plot(hilb(51))
```

Authors

C. B.

Name

hilbert — Discrete-time analytic signal computation of a real signal using Hilbert transform

```
x=hilbert(xr)
```

Parameters

xr
: real vector : the real signal samples

x
: Complex vector: the discrete-time analytic signal.

Description

Returns theanalytic signal, from a real data sequence.

The analytic signal `x = xr + i*xi` has a real part, `xr`, which is the original data, and an imaginary part, `xi`, which contains the Hilbert transform. The imaginary part is a version of the original real sequence with a 90° phase shift.

References

`http://ieeexplore.ieee.org/iel5/78/16975/00782222.pdf?arnumber=782222`

Marple, S.L., "Computing the discrete-time analytic signal via FFT," IEEE Transactions on Signal Processing, Vol. 47, No.9 (September 1999), pp.2600-2603

See Also

window , hil

Examples

```
//compare the discrete-time analytic signal imaginary part of the impulse rea
// with the FIR approximation of the Hilbert transform filter
m=25;
n=2*m+1;
y=hilbert(eye(n,1));
h=hilb(n)';
h=[h((m+1):$);h(1:m)];
plot([imag(y) h])
```

Authors

C. B.

Name

iir — iir digital filter

```
[hz]=iir(n,ftype,fdesign,frq,delta)
```

Parameters

n
: positive number witn inteher value, the filter order.

ftype
: string specifying the filter type, the possible values are: `'lp'` for low-pass,`'hp'` for high pass,`'bp'` for band pass and `'sb'` for stop band.

fdesign
: string specifying the analog filter design, the possible values are: `'butt'`, `'cheb1'`, `'cheb2'` and `'ellip'`

frq
: 2-vector of discrete cut-off frequencies (i.e., `0<frq<.5`). For `'lp'` and `'hp'` filters only `frq(1)` is used. For `'bp'` and `'sb'` filters `frq(1)` is the lower cut-off frequency and `frq(2)` is the upper cut-off frequency

delta
: 2-vector of error values for `cheb1`, `cheb2`, and `ellip` filters where only `delta(1)` is used for `cheb1` case, only `delta(2)` is used for `cheb2` case, and `delta(1)` and `delta(2)` are both used for `ellip` case. `0<delta(1),delta(2)<1`

  - for `cheb1` filters `1-delta(1)<ripple<1` in passband
  - for `cheb2` filters `0<ripple<delta(2)` in stopband
  - for `ellip` filters `1-delta(1)<ripple<1` in passband and `0<ripple<delta(2)` in stopband

Description

function which designs an iir digital filter using analog filter designs and bilinear transformation .

Examples

```
hz=iir(3,'bp','ellip',[.15 .25],[.08 .03]);
[hzm,fr]=frmag(hz,256);
plot2d(fr',hzm')
xtitle('Discrete IIR filter band pass  0.15<fr<0.25 ',' ',' ');
q=poly(0,'q');     //to express the result in terms of the delay operator q=z^-1
hzd=horner(hz,1/q)
```

See Also

eqfir, eqiir, analpf, bilt

Authors

Carey Bunks

Name

iirgroup — group delay Lp IIR filter optimization

```
[lt,grad]=iirgroup(p,r,theta,omega,wt,td)
[cout,grad,ind]=iirlp(x,ind,p,[flag],lambda,omega,ad,wa,td,wt)
```

Parameters

r
: vector of the module of the poles and the zeros of the filters

theta
: vector of the argument of the poles and the zeros of the filters

omega
: frequencies where the filter specifications are given

wt
: weighting function for and the group delay

td
: desired group delay

lt, grad
: criterium and gradient values

Description

optimization of IIR filters for the Lp criterium for the the group delay. (Rabiner & Gold pp270-273).

Name

iirlp — Lp IIR filter optimization

```
[cost,grad,ind]=iirlp(x,ind,p,[flag],lambda,omega,ad,wa,td,wt)
```

Parameters

x
: 1X2 vector of the module and argument of the poles and the zeros of the filters

flag
: string : `'a'` for amplitude, 'gd' for group delay; default case for amplitude and group delay.

omega
: frequencies where the filter specifications are given

wa,wt
: weighting functions for the amplitude and the group delay

lambda
: weighting (with `1-lambda`) of the costs (`'a'` and `'gd'` for getting the global cost.

ad, td
: desired amplitude and group delay

cost, grad
: criterium and gradient values

Description

optimization of IIR filters for the Lp criterium for the amplitude and/or the group delay. (Rabiner & Gold pp270-273).

Name

intdec — Changes sampling rate of a signal

```
[y]=intdec(x,lom)
```

Parameters

x
: input sampled signal

lom
: For a 1D signal this is a scalar which gives the rate change. For a 2D signal this is a 2-Vector of sampling rate changes `lom`=(col rate change,row rate change)

y
: Output sampled signal

Description

Changes the sampling rate of a 1D or 2D signal by the rates in lom

Authors

C. B.

Name

jmat — row or column block permutation

```
[j]=jmat(n,m)
```

Parameters

n
: number of block rows or block columns of the matrix

m
: size of the (square) blocks

Description

This function permutes block rows or block columns of a matrix

Name

kalm — Kalman update

```
[x1,p1,x,p]=kalm(y,x0,p0,f,g,h,q,r)
```

Parameters

f,g,h
: current system matrices

q, r
: covariance matrices of dynamics and observation noise

x0,p0
: state estimate and error variance at t=0 based on data up to t=-1

y
: current observation Output from the function is:

x1,p1
: updated estimate and error covariance at t=1 based on data up to t=0

x
: updated estimate and error covariance at t=0 based on data up to t=0

Description

function which gives the Kalman update and error variance

Authors

C. B.

Name

lattn — recursive solution of normal equations

```
[la,lb]=lattn(n,p,cov)
```

Parameters

n
: maximum order of the filter

p
: fixed dimension of the MA part. If `p= -1`, the algorithm reduces to the classical Levinson recursions.

cov
: matrix containing the `Rk`'s (`d*d` matrices for a d-dimensional process).It must be given the following way

la
: list-type variable, giving the successively calculated polynomials (degree 1 to degree n),with coefficients Ak

Description

solves recursively on `n` (`p` being fixed) the following system (normal equations), i.e. identifies the AR part (poles) of a vector ARMA(n,p) process

where {`Rk;k=1,nlag`} is the sequence of empirical covariances

Authors

G. Le V.

Name

lattp — lattp

```
[la,lb]=lattp(n,p,cov)
```

Description

see lattn

Authors

G.Levey

Name

lev — Yule-Walker equations (Levinson's algorithm)

```
[ar,sigma2,rc]=lev(r)
```

Parameters

r
: correlation coefficients

ar
: auto-Regressive model parameters

sigma2
: scale constant

rc
: reflection coefficients

Description

resolve the Yule-Walker equations

using Levinson's algorithm.

Authors

C. B.

Name

levin — Toeplitz system solver by Levinson algorithm (multidimensional)

```
[la,sig]=levin(n,cov)
```

Parameters

n
: A scalar with integer value: the maximum order of the filter

cov
: A `(nlag*d) x d` matrix. It contains the `Rk` (`d x d` matrices for a `d`-dimensional process) stored in the following way :

$$\begin{pmatrix} R_0 \\ R_1 \\ R_2 \\ \vdots \\ R_{nlags} \end{pmatrix}$$

la
: A list, the successively calculated Levinson polynomials (degree 1 to `n`), with coefficients `Ak`

sig
: A list, the successive mean-square errors.

Description

function which solves recursively on n the following Toeplitz system (normal equations)

$$(I \quad -A_1 \quad \cdots \quad -A_n) * \begin{pmatrix} R_1 & R_2 & \cdots & R_n \\ R_0 & R_1 & \cdots & R_{n-1} \\ R_{-1} & R_0 & \cdots & R_{n-2} \\ \vdots & \vdots & \cdots & \vdots \\ R_{2-n} & R_{3-n} & \cdots & R_1 \\ R_{1-n} & R_{2-n} & \cdots & R_0 \end{pmatrix} = 0$$

where {`Rk;k=1:nlag`} is the sequence of `nlag` empirical covariances

Examples

```
//We use the 'levin' macro for solving the normal equations
//on two examples: a one-dimensional and a two-dimensional process.
//We need the covariance sequence of the stochastic process.
//This example may usefully be compared with the results from
//the 'phc' macro (see the corresponding help and example in it)
//
//
//1) A one-dimensional process
```

```
//   -------------------------
//
//We generate the process defined by two sinusoids (1Hz and 2 Hz)
//in additive Gaussian noise (this is the observed process);
//the simulated process is sampled at 10 Hz (step 0.1 in t, underafter).
//
t1=0:.1:100;rand('normal');
y1=sin(2*%pi*t1)+sin(2*%pi*2*t1);y1=y1+rand(y1);plot(t1,y1);
//
//covariance of y1
//
nlag=128;
c1=corr(y1,nlag);
c1=c1';//c1 needs to be given columnwise (see the section PARAMETERS of this he
//
//compute the filter for a maximum order of n=10
//la is a list-type variable each element of which
//containing the filters of order ranging from 1 to n; (try varying n)
//in the d-dimensional case this is a matrix polynomial (square, d X d)
//sig gives, the same way, the mean-square error
//
n=15;
[la1,sig1]=levin(n,c1);
//
//verify that the roots of 'la' contain the
//frequency spectrum of the observed process y
//(remember that y is sampled -in our example
//at 10Hz (T=0.1s) so that we need to retrieve
//the original frequencies (1Hz and 2 Hz) through
//the log and correct scaling by the frequency sampling)
//we verify this for each filter order
//
for i=1:n, s1=roots(la1(i));s1=log(s1)/2/%pi/.1;
//
//now we get the estimated poles (sorted, positive ones only !)
//
s1=sort(imag(s1));s1=s1(1:i/2);end;
//
//the last two frequencies are the ones really present in the observed
//process ---> the others are "artifacts" coming from the used model size.
//This is related to the rather difficult problem of order estimation.
//
//2) A 2-dimensional process
//   ----------------------
//(4 frequencies 1, 2, 3, and 4 Hz, sampled at 0.1 Hz :
//   |y_1|        y_1=sin(2*Pi*t)+sin(2*Pi*2*t)+Gaussian noise
// y=|   | with :
//   |y_2|        y_2=sin(2*Pi*3*t)+sin(2*Pi*4*t)+Gaussian noise
//
//
d=2;dt=0.1;
nlag=64;
t2=0:2*%pi*dt:100;
y2=[sin(t2)+sin(2*t2)+rand(t2);sin(3*t2)+sin(4*t2)+rand(t2)];
c2=[];
for j=1:2, for k=1:2, c2=[c2;corr(y2(k,:),y2(j,:),nlag)];end;end;
c2=matrix(c2,2,128);cov=[];
for j=1:64,cov=[cov;c2(:,(j-1)*d+1:j*d)];end;//covar. columnwise
```

```
c2=cov;
//
//in the multidimensional case, we have to compute the
//roots of the determinant of the matrix polynomial
//(easy in the 2-dimensional case but tricky if d>=3 !).
//We just do that here for the maximum desired
//filter order (n); mp is the matrix polynomial of degree n
//
[la2,sig2]=levin(n,c2);
mp=la2(n);determinant=mp(1,1)*mp(2,2)-mp(1,2)*mp(2,1);
s2=roots(determinant);s2=log(s2)/2/%pi/0.1;//same trick as above for 1D process
s2=sort(imag(s2));s2=s2(1:d*n/2);//just the positive ones !
//
//There the order estimation problem is seen to be much more difficult !
//many artifacts ! The 4 frequencies are in the estimated spectrum
//but beneath many non relevant others.
//
```

See Also

phc

Authors

G. Le Vey

Name

lgfft — utility for fft

```
[y]=lgfft(x)
```

Parameters

x
: real or complex vector

Description

returns the lowest power of 2 larger than `size(x)` (for FFT use).

Name

lindquist — Lindquist's algorithm

```
[P,R,T]=lindquist(n,H,F,G,R0)
```

Parameters

n
: number of iterations.

H, F, G
: estimated triple from the covariance sequence of `y`.

R0
: E(yk*yk')

P
: solution of the Riccati equation after n iterations.

R, T
: gain matrices of the filter.

Description

computes iteratively the minimal solution of the algebraic Riccati equation and gives the matrices `R` and `T` of the filter model, by the Lindquist's algorithm.

See Also

srfaur , faurre , phc

Authors

G. Le V.

Name

mese — maximum entropy spectral estimation

```
[sm,fr]=mese(x [,npts]);
```

Parameters

x
: Input sampled data sequence

npts
: Optional parameter giving number of points of `fr` and `sm` (default is 256)

sm
: Samples of spectral estimate on the frequency grid `fr`

fr
: npts equally spaced frequency samples in `[0,.5)`

Description

Calculate the maximum entropy spectral estimate of `x`

Authors

C. B.

Name

mfft — multi-dimensional fft

```
[xk]=mfft(x,flag,dim)
```

Parameters

x
: `x(i,j,k,...)` input signal in the form of a row vector whose values are arranged so that the i index runs the quickest, followed by the j index, etc.

flag
(-1) FFT or (1) inverse FFT

dim
dimension vector which gives the number of values of `x` for each of its indices

xk
output of multidimensional fft in same format as for `x`

Description

FFT for a multi-dimensional signal

For example for a three dimensional vector which has three points along its first dimension, two points along its second dimension and three points along its third dimension the row vector is arranged as follows

```
x=[x(1,1,1),x(2,1,1),x(3,1,1),
   x(1,2,1),x(2,2,1),x(3,2,1),
         x(1,1,2),x(2,1,2),x(3,1,2),
         x(1,2,2),x(2,2,2),x(3,2,2),
               x(1,1,3),x(2,1,3),x(3,1,3),
               x(1,2,3),x(2,2,3),x(3,2,3)]
```

and the `dim` vector is: `dim=[3,2,3]`

Authors

C. B.

Name

mrfit — frequency response fit

```
sys=mrfit(w,mag,order)
[num,den]=mrfit(w,mag,order)
sys=mrfit(w,mag,order,weight)
[num,den]=mrfit(w,mag,order,weight)
```

Parameters

w
: positive real vector of frequencies (Hz)

mag
: real vector of frequency responses magnitude (same size as `w`)

order
: integer (required order, degree of `den`)

weight
: positive real vector (default value `ones(w)`).

num,den
: stable polynomials

Description

`sys=mrfit(w,mag,order,weight)` returns a bi-stable transfer function `G(s)=sys=num/den`, of of given `order` such that its frequency response magnitude `abs(G(w(i)))` matches `mag(i)` i.e. `abs(freq(num,den,%i*w))` should be close to `mag`. `weight(i)` is the weigth given to `w(i)`.

Examples

```
w=0.01:0.01:2;s=poly(0,'s');
G=syslin('c',2*(s^2+0.1*s+2),(s^2+s+1)*(s^2+0.3*s+1)); // syslin('c',Num,Den);
fresp=repfreq(G,w);
mag=abs(fresp);
Gid=mrfit(w,mag,4);
frespfit=repfreq(Gid,w);
plot2d([w',w'],[mag(:),abs(frespfit(:))])
```

See Also

cepstrum , frfit , freq , calfrq

Name

%asn — elliptic integral

```
[y]=%asn(x,m)
```

Parameters

x
: upper limit of integral (`x>0`) (can be a vector)

m
: parameter of integral (`0<m<1`)

y
: value of the integral

Description

Calculates the elliptic integral

If `x` is a vector, `y` is a vector of same dimension as `x`.

Examples

```
m=0.8;z=%asn(1/sqrt(m),m);K=real(z);Ktilde=imag(z);
x2max=1/sqrt(m);
x1=0:0.05:1;x2=1:((x2max-1)/20):x2max;x3=x2max:0.05:10;
x=[x1,x2,x3];
y=%asn(x,m);
rect=[0,-Ktilde,1.1*K,2*Ktilde];
plot2d(real(y)',imag(y)',1,'011',' ',rect)
//
deff('y=f(t)','y=1/sqrt((1-t^2)*(1-m*t^2))');
intg(0,0.9,f)-%asn(0.9,m)  //Works for real case only!
```

Authors

F. D.

Name

%k — Jacobi's complete elliptic integral

```
[K]=%k(m)
```

Parameters

m
: parameter of the elliptic integral `0<m<1` (`m` can be a vector)

K
: value of the elliptic integral from 0 to 1 on the real axis

Description

Calculates Jacobi's complete elliptic integral of the first kind :

References

Abramowitz and Stegun page 598

Examples

```
m=0.4;
%asn(1,m)
%k(m)
```

See Also

%asn

Authors

F.D.

Name

%sn — Jacobi 's elliptic function

```
[y]=%sn(x,m)
```

Parameters

x
: a point inside the fundamental rectangle defined by the elliptic integral; `x` is a vector of complex numbers

m
: parameter of the elliptic integral (`0<m<1`)

y
: result

Description

Jacobi 's sn elliptic function with parameter `m`: the inverse of the elliptic integral for the parameter `m`.

The amplitude am is computed in fortran and the addition formulas for elliptic functions are applied

Examples

```
m=0.36;
K=%k(m);
P=4*K; //Real period
real_val=0:(P/50):P;
plot(real_val,real(%sn(real_val,m)))
xbasc();
KK=%k(1-m);
Ip=2*KK;
ima_val1=0:(Ip/50):KK-0.001;
ima_val2=(KK+0.05):(Ip/25):(Ip+KK);
z1=%sn(%i*ima_val1,m);z2=%sn(%i*ima_val2,m);
plot2d([ima_val1',ima_val2'],[imag(z1)',imag(z2)']);
xgrid(3)
```

See Also

%asn , %k

Authors

F. D.

Name

phc — Markovian representation

```
[H,F,G]=phc(hk,d,r)
```

Parameters

hk
: hankel matrix

d
: dimension of the observation

r
: desired dimension of the state vector for the approximated model

H, F, G
: relevant matrices of the Markovian model

Description

Function which computes the matrices `H, F, G` of a Markovian representation by the principal hankel component approximation method, from the hankel matrix built from the covariance sequence of a stochastic process.

Examples

```
//
//This example may usefully be compared with the results from
//the 'levin' macro (see the corresponding help and example)
//
//We consider the process defined by two sinusoids (1Hz and 2 Hz)
//in additive Gaussian noise (this is the observation);
//the simulated process is sampled at 10 Hz.
//
t=0:.1:100;rand('normal');
y=sin(2*%pi*t)+sin(2*%pi*2*t);y=y+rand(y);plot(t,y)
//
//covariance of y
//
nlag=128;
c=corr(y,nlag);
//
//hankel matrix from the covariance sequence
//(we can choose to take more information from covariance
//by taking greater n and m; try it to compare the results !
//
n=20;m=20;
h=hank(n,m,c);
//
//compute the Markov representation (mh,mf,mg)
//We just take here a state dimension equal to 4 :
//this is the rather difficult problem of estimating the order !
//Try varying ns !
```

```
//(the observation dimension is here equal to one)
ns=4;
[mh,mf,mg]=phc(h,1,ns);
//
//verify that the spectrum of mf contains the
//frequency spectrum of the observed process y
//(remember that y is sampled -in our example
//at 10Hz (T=0.1s) so that we need
//to retrieve the original frequencies through the log
//and correct scaling by the frequency sampling)
//
s=spec(mf);s=log(s);
s=s/2/%pi/.1;
//
//now we get the estimated spectrum
imag(s),
//
```

See Also

levin

Name

pspect — two sided cross-spectral estimate between 2 discrete time signals using the Welch's average periodogram method.

```
[sm [,cwp]]=pspect(sec_step,sec_leng,wtype,x [,y] [,wpar])
[sm [,cwp]]=pspect(sec_step,sec_leng,wtype,nx [,ny] [,wpar])
```

Parameters

x
: vector, the time-domain samples of the first signal.

y
: vector, the time-domain samples of the second signal. If `y` is omitted it is supposed to be equal to `x` (auto-correlation). If it is present, it must have the same numer of element than `x`.

nx
: a scalar : the number of samples in the `x` signal. In this case the segments of the `x` signal are loaded by a user defined function named `getx` (see below).

ny
: a scalar : the number of samples in the `y` signal. In this case the segments of the y signal are loaded by a user defined function named `gety` (see below). If present `ny` must be equal to `nx`.

sec_step
: offset of each data window. The overlap `D` is given by sec_leng -sec_step. `if sec_step==sec_leng/2` 50% overlap is made. The overlap

sec_leng
: Number of points of the window.

wtype
: The window type

- `'re'`: rectangular
- `'tr'`: triangular
- `'hm'`: Hamming
- `'hn'` : Hanning
- `'kr'`: Kaiser,in this case the wpar argument must be given
- `'ch'`: Chebyshev, in this case the wpar argument must be given

wpar
: optional parameters for `Kaiser and Chebyshev windows:`

- 'kr': `wpar must be a strictly positive number`
- 'ch': `wpar` must be a 2 element vector `[main_lobe_width,side_lobe_height]with 0<main_lobe_width<.5`, and `side_lobe_height>0`

sm
: Two sided power spectral estimate in the interval `[0,1]` of the normalized frequencies. It is a row array with `sec_len` elements . The array is real in case of auto-correlation and complex in case of cross-correlation.

The associated normalized frequencies array is `linspace(0,1,sec_len)`.

cwp
: unspecified Chebyshev window parameter in case of Chebyshev windowing, or an empty matrix.

Description

Computes the cross-spectrum estimate of two signals `x` and `y` if both are given and the auto-spectral estimate of `x` otherwise. Spectral estimate obtained using the modified periodogram method.

The cross spectrum of two signal `x` and `y` is defined to be

$$S_{xy}(\omega) = \frac{1}{N}\left(\sum_{n=0}^{N-1} x(n)e^{-i\omega n}\right)\left(\sum_{n=0}^{N-1} y(n)e^{i\omega n}\right)$$

The modified periodogram method of spectral estimation repeatedly calculates the periodogram of windowed sub-sections of the data containes in `x` and `y` . These periodograms are then averaged together and normalized by an appropriate constant to obtain the final spectral estimate. It is the averaging process which reduces the variance in the estimate.

For batch processing, the `x` and `y` data may be read segment by segment using the `getx` and `gety` user defined functions. These functions have the following calling sequence:

`xk=getx(ns,offset)` and `yk=gety(ns,offset)` where `ns` is the segment size and `offset` is the index of the first element of the segment in the full signal.

Reference

Oppenheim, A.V., and R.W. Schafer. Discrete-Time Signal Processing, Upper Saddle River, NJ: Prentice-Hall, 1999

Examples

```
rand('normal');rand('seed',0);
x=rand(1:1024-33+1);
//make low-pass filter with eqfir
nf=33;bedge=[0 .1;.125 .5];des=[1 0];wate=[1 1];
h=eqfir(nf,bedge,des,wate);
//filter white data to obtain colored data
h1=[h 0*ones(1:maxi(size(x))-1)];
x1=[x 0*ones(1:maxi(size(h))-1)];
hf=fft(h1,-1); xf=fft(x1,-1);y=real(fft(hf.*xf,1));

//plot magnitude of filter
h2=[h 0*ones(1:968)];hf2=fft(h2,-1);hf2=real(hf2.*conj(hf2));
hsize=maxi(size(hf2));fr=(1:hsize)/hsize;plot(fr,log(hf2));
//pspect example
sm=pspect(100,200,'tr',y);smsize=maxi(size(sm));fr=(1:smsize)/smsize;
plot(fr,log(sm));
rand('unif');
```

See Also

cspect, pspect, mese, window

Authors

C. Bunks INRIA

Name

remez — Remez exchange algorithm for the weighted chebyshev approximation of a continuous function with a sum of cosines.

```
an=remez(guess,mag,fgrid,weight)
```

Parameters

guess
: real array of size `n+2 the` initial guess

fgrid
: real array of size `ng`: the grid of normalized frequency points in [0,.5[

mag
: real array of size `ng`: the desired magnitude on grid `fg`

weight
: real array of size `ng`: weighting function on error on grid `fg`

an
: real array of size `n`: cosine coefficients

Description

Minimax approximation of a frequency domain magnitude response. The approximation takes the form

```
 h = sum[a(i)*cos(weight)], i=1:n
```

An FIR, linear-phase filter can be obtained from the the output of `remez` by using the following commands:

```
  hn(1:nc-1)=an(nc:-1:2)/2;
  hn(nc)=an(1);
  hn(nc+1:2*nc-1)=an(2:nc)/2;
```

This function is mainly intended to be called by the remezb function.

Bibliography

E.W. Cheney, Introduction to Approximation Theory, McGraw-Hill, 1966

http://en.wikipedia.org/wiki/Remez_algorithm

References

This function is based on the fortran code `remez.f` written by:

- James H. Mcclellan, department of electrical engineering and computer science, Massachusetts Institute of Technology, Cambridge, Massachussets. 02139
- Thomas W. Parks, department of electrical engineering, Rice university, Houston, Texas 77001
- Thomas W. Parks, department of electrical engineering, Rice university, Houston, Texas 77001

Examples

```
      nc=21;
```

```
ngrid=nc*250;
fgrid=.5*(0:(ngrid-1))/(ngrid-1);
mag(1:ngrid/2)=ones(1:ngrid/2);
mag(ngrid/2+1:ngrid)=0*ones(1:ngrid/2);
weight=ones(fgrid);
guess=round(1:ngrid/nc:ngrid);
guess(nc+1)=ngrid;
guess(nc+2)=ngrid;
an=remez(guess,mag,fgrid,weight);
```

See Also

remezb, eqfir

Name

remezb — Minimax approximation of magnitude response

```
[an]=remezb(nc,fg,ds,wt)
```

Parameters

nc
: Number of cosine functions

fg
: Grid of frequency points in [0,.5)

ds
: Desired magnitude on grid `fg`

wt
: Weighting function on error on grid `fg`

an
: Cosine filter coefficients

Description

Minimax approximation of a frequency domain magnitude response. The approximation takes the form `h = sum[a(n)*cos(wn)]` for n=0,1,...,nc. An FIR, linear-phase filter can be obtained from the the output of the function by using the following commands

```
hn(1:nc-1)=an(nc:-1:2)/2;
hn(nc)=an(1);
hn(nc+1:2*nc-1)=an(2:nc)/2;
```

Examples

```
// Choose the number of cosine functions and create a dense grid
// in [0,.24) and [.26,.5)
nc=21;ngrid=nc*16;
fg=.24*(0:ngrid/2-1)/(ngrid/2-1);
fg(ngrid/2+1:ngrid)=fg(1:ngrid/2)+.26*ones(1:ngrid/2);
// Specify a low pass filter magnitude for the desired response
ds(1:ngrid/2)=ones(1:ngrid/2);
ds(ngrid/2+1:ngrid)=zeros(1:ngrid/2);
// Specify a uniform weighting function
wt=ones(fg);
// Run remezb
an=remezb(nc,fg,ds,wt)
// Make a linear phase FIR filter
hn(1:nc-1)=an(nc:-1:2)/2;
hn(nc)=an(1);
hn(nc+1:2*nc-1)=an(2:nc)/2;
```

```
// Plot the filter's magnitude response
plot(.5*(0:255)/256,frmag(hn,256));
//////////////
// Choose the number of cosine functions and create a dense grid in [0,.5)
nc=21; ngrid=nc*16;
fg=.5*(0:(ngrid-1))/ngrid;
// Specify a triangular shaped magnitude for the desired response
ds(1:ngrid/2)=(0:ngrid/2-1)/(ngrid/2-1);
ds(ngrid/2+1:ngrid)=ds(ngrid/2:-1:1);
// Specify a uniform weighting function
wt=ones(fg);
// Run remezb
an=remezb(nc,fg,ds,wt)
// Make a linear phase FIR filter
hn(1:nc-1)=an(nc:-1:2)/2;
hn(nc)=an(1);
hn(nc+1:2*nc-1)=an(2:nc)/2;
// Plot the filter's magnitude response
plot(.5*(0:255)/256,frmag(hn,256));
```

See Also

eqfir

Authors

C. B.

Name

rpem — RPEM estimation

```
[w1,[v]]=rpem(w0,u0,y0,[lambda,[k,[c]]])
```

Parameters

a,b,c
: `a=[a(1),...,a(n)], b=[b(1),...,b(n)], c=[c(1),...,c(n)]`

w0
: `list(theta,p,phi,psi,l)` where:

theta
: [a,b,c] is a real vector of order `3*n`

p
: (3*n x 3*n) real matrix.

phi,psi,l
: real vector of dimension `3*n`

During the first call on can take:

```
theta=phi=psi=l=0*ones(1,3*n). p=eye(3*n,3*n)
```

u0
: real vector of inputs (arbitrary size) (if no input take `u0=[ ]`).

y0
: vector of outputs (same dimension as `u0` if `u0` is not empty). (`y0(1)` is not used by rpem).

If the time domain is `(t0,t0+k-1)` the `u0` vector contains the inputs

`u(t0),u(t0+1),..,u(t0+k-1)` and `y0` the outputs

`y(t0),y(t0+1),..,y(t0+k-1)`

Description

Recursive estimation of parameters in an ARMAX model. Uses Ljung-Soderstrom recursive prediction error method. Model considered is the following:

```
y(t)+a(1)*y(t-1)+...+a(n)*y(t-n)=
b(1)*u(t-1)+...+b(n)*u(t-n)+e(t)+c(1)*e(t-1)+...+c(n)*e(t-n)
```

The effect of this command is to update the estimation of unknown parameter `theta=[a,b,c]` with

`a=[a(1),...,a(n)], b=[b(1),...,b(n)], c=[c(1),...,c(n)]`.

Optional parameters

lambda
optional parameter (forgetting constant) choosed close to 1 as convergence occur:

`lambda=[lambda0,alfa,beta]` evolves according to :

```
lambda(t)=alfa*lambda(t-1)+beta
```

with `lambda(0)=lambda0`

`k`contraction factor to be chosen close to 1 as convergence occurs.

`k=[k0,mu,nu]` evolves according to:

```
k(t)=mu*k(t-1)+nu
```

with `k(0)=k0`.

`c`large parameter.(`c=1000` is the default value).

Output parameters:

`w1`update for `w0`.

`v`sum of squared prediction errors on `u0, y0`.(optional).

In particular `w1(1)` is the new estimate of `theta`. If a new sample `u1, y1` is available the update is obtained by:

`[w2,[v]]=rpem(w1,u1,y1,[lambda,[k,[c]]])`. Arbitrary large series can thus be treated.

Name

sincd — digital sinc function or Direchlet kernel

```
[s]=sincd(n,flag)
```

Parameters

n
: integer

flag
: if `flag = 1` the function is centred around the origin; if `flag = 2` the function is delayed by `%pi/(2*n)`

s
: vector of values of the function on a dense grid of frequencies

Description

function which calculates the function `Sin(N*x)/N*Sin(x)`

Examples

```
plot(sincd(10,1))
```

Authors

G. Le V.

Name

srfaur — square-root algorithm

```
[p,s,t,l,rt,tt]=srfaur(h,f,g,r0,n,p,s,t,l)
```

Parameters

h, f, g
: convenient matrices of the state-space model.

r0
: E(yk*yk').

n
: number of iterations.

p
: estimate of the solution after n iterations.

s, t, l
: intermediate matrices for successive iterations;

rt, tt
: gain matrices of the filter model after n iterations.

p, s, t, l
: may be given as input if more than one recursion is desired (evaluation of intermediate values of p).

Description

square-root algorithm for the algebraic Riccati equation.

Examples

```
//GENERATE SIGNAL
x=%pi/10:%pi/10:102.4*%pi;
rand('seed',0);rand('normal');
y=[1;1]*sin(x)+[sin(2*x);sin(1.9*x)]+rand(2,1024);
//COMPUTE CORRELATIONS
c=[];for j=1:2,for k=1:2,c=[c;corr(y(k,:),y(j,:),64)];end;end
c=matrix(c,2,128);
//FINDING H,F,G with 6 states
hk=hank(20,20,c);
[H,F,G]=phc(hk,2,6);
//SOLVING RICCATI EQN
r0=c(1:2,1:2);
[P,s,t,l,Rt,Tt]=srfaur(H,F,G,r0,200);
//Make covariance matrix exactly symetric
Rt=(Rt+Rt')/2
```

See Also

phc , faurre , lindquist

Name

srkf — square root Kalman filter

```
[x1,p1]=srkf(y,x0,p0,f,h,q,r)
```

Parameters

f, h
: current system matrices

q, r
: covariance matrices of dynamics and observation noise

x0, p0
: state estimate and error variance at t=0 based on data up to t=-1

y
: current observation Output from the function is

x1, p1
: updated estimate and error covariance at t=1 based on data up to t=0

Description

square root Kalman filter algorithm

Authors

C. B.

Name

sskf — steady-state Kalman filter

```
[xe,pe]=sskf(y,f,h,q,r,x0)
```

Parameters

y
: data in form `[y0,y1,...,yn]`, `yk` a column vector

f
: system matrix dim(NxN)

h
: observations matrix dim(MxN)

q
: dynamics noise matrix dim(NxN)

r
: observations noise matrix dim(MxM)

x0
: initial state estimate

xe
: estimated state

pe
: steady-state error covariance

Description

steady-state Kalman filter

Authors

C. B.

Name

syredi — Design of iir filters, syredi code interface

```
[fact,b2,b1,b0,c1,c0,zzeros,zpoles]=syredi(ityp,iapro,om,deltap,deltas)
```

Parameters

itype
: integer, the filter type: 1 stands for low-pass, 2 for high-pass, 3 for band-pass, 4 for stop-band.

iappro
: integer, the design approximation type: 1 stands for butterworth, 2 for elliptic, 3 for Chebytchev1, 4 for Chebytchev2.

om
: 4-vector of cutoff frequencies (in radians) `om=[om1,om2,om3,om4]`,

  `0 <= om1 <= om2 <= om3 <= om4 <= pi`.

  When `ftype`='lp' or `'hp'`, `om3` and `om4` are not used and may be set to 0.

deltap
: a real scalar, the ripple in the passband. `0< deltap <1`

deltas
: a real scalar, the ripple in the stopband. `0< deltas <1`

gain
: scalar, the filter gain

b2
: real row vector, degree 2 coefficients of numerators.

b1
: real row vector, degree 1 coefficients of numerators.

b0
: real row vector, degree 0 coefficients of numerators.

c1
: real row vector, degree 1 coefficients of denominators.

c0
: real row vector, degree 0 coefficients of denominators.

zzeros
: complex row vector, filter zeros in the z-domain

zpoles
: complex row vector, filter poles in the z-domain

Description

Computes iir filter approximation. The result is given as a set of second order transfer functions `Hi=(b0(i)+b1(i)*z+b2(i)*z^2)/(c0(i)+c1(i)*z+z^2)` and also as a poles, zeros, gain representation.

The filter obtained is `h=fact*H1*...*Hn`.

Remark

This built-in function is mainly intended to be used by the eqiir function.

References

The syredi code is derived from doredi package written by Guenter F. Dehner, Institut fuer Nachrichtentechnik Universitaet Erlangen-Nuernberg, Germany.

Dehner,G.F. 1979, DOREDI: Program for Design and Optimization of REcursive DIgital filters-Programs for Digital Signal Processing, ed:Digital Signal Processing committee of IEEE Acoustics, Speech and Signal Processing Society.

For DOREDI.f source code see http://michaelgellis.tripod.com/dsp/pgm25.html

Examples

```
[fact,b2,b1,b0,c1,c0,zzeros,zpoles]=syredi(1,4,[2*%pi/10,4*%pi/10,0,0],0.
h=fact*(b0+b1*%z+b2*%z^2)./(c0+c1*%z+%z^2)
```

See Also

eqiir

Name

system — observation update

```
[x1,y]=system(x0,f,g,h,q,r)
```

Parameters

x0
: input state vector

f
: system matrix

g
: input matrix

h
: Output matrix

q
: input noise covariance matrix

r
: output noise covariance matrix

x1
: output state vector

y
: output observation

Description

define system function which generates the next observation given the old state. System recursively calculated

```
x1=f*x0+g*u
y=h*x0+v
```

where `u` is distributed `N(0,q)` and `v` is distribute `N(0,r)`.

Authors

C. B.

Name

trans — low-pass to other filter transform

```
hzt=trans(hz,tr_type,frq)
hzt=trans(pd,zd,gd,tr_type,frq)
```

Parameters

hz
: a single input single output discrete transfer function, the low pass filter

pd
: Vector of given filter poles

zd
: Vector of given filter zeros

gd
: scalar: the given filter gain

tr_type
: string, the type of transformation, see description for possible values

frq
: 2-vector of discrete cut-off frequencies (i.e.,`0<frq<.5`). see description for details.

hzt
: transformed filter transfert function.

Description

function for transforming standardized low-pass filter given its poles-zeros_gain representation into one of the following filters:

tr_type='lp'
: low pass filter, the cutoff frequency is given by the first entry of `frq`, the second one is ignored.

tr_type='hp'
: high pass filter, the cutoff frequency is given by the first entry of `frq`, the second one is ignored.

tr_type='bp'
: band pass filter, the frequency range is given by `frq(1)` and `frq(2)`.

tr_type='sb'
: stop band filter, the frequency range is given by `frq(1)` and `frq(2)`.

Used functions

bilt

Examples

```
clf()
```

```
Hlp=iir(3,'lp','ellip',[0.1 0],[.08 .03]);
Hbp=trans(Hlp,'bp',[0.01 0.1]);
Hsb=trans(Hlp,'sb',[0.01 0.1])

clf();gainplot([Hlp;Hbp;Hsb],1d-3,0.48);
l=legend(['original low pass';'band pass';'stop band']);
l.legend_location="in_lower_left";
```

Authors

Carey Bunks ;

See Also

iir, bilt

Name

wfir — linear-phase FIR filters

```
[wft,wfm,fr]=wfir(ftype,forder,cfreq,wtype,fpar)
```

Parameters

ftype
: string : `'lp','hp','bp','sb'` (filter type)

forder
: Filter order (pos integer)(odd for `ftype='hp'` or `'sb'`)

cfreq
: 2-vector of cutoff frequencies (`0<cfreq(1),cfreq(2)<.5`) only `cfreq(1)` is used when `ftype='lp'` or `'hp'`

wtype
: Window type (`'re','tr','hm','hn','kr','ch'`)

fpar
: 2-vector of window parameters. Kaiser window `fpar(1)>0 fpar(2)=0`. Chebyshev window `fpar(1)>0, fpar(2)<0` or `fpar(1)<0, 0<fpar(2)<.5`

wft
: time domain filter coefficients

wfm
: frequency domain filter response on the grid fr

fr
: Frequency grid

Description

Function which makes linear-phase, FIR low-pass, band-pass, high-pass, and stop-band filters using the windowing technique. Works interactively if called with no arguments.

Authors

C. Bunks

Name

wiener — Wiener estimate

```
[xs,ps,xf,pf]=wiener(y,x0,p0,f,g,h,q,r)
```

Parameters

f, g, h
: system matrices in the interval `[t0,tf]`

  f
  : =`[f0,f1,...,ff]`, and `fk` is a nxn matrix

  g
  : =`[g0,g1,...,gf]`, and `gk` is a nxn matrix

  h
  : =`[h0,h1,...,hf]`, and `hk` is a mxn matrix

q, r
: covariance matrices of dynamics and observation noise

  q
  : =`[q0,q1,...,qf]`, and `qk` is a nxn matrix

  r
  : =`[r0,r1,...,rf]`, and `gk` is a mxm matrix

x0, p0
: initial state estimate and error variance

y
: observations in the interval `[t0,tf]`. `y=[y0,y1,...,yf]`, and `yk` is a column m-vector

xs
: Smoothed state estimate `xs= [xs0,xs1,...,xsf]`, and `xsk` is a column n-vector

ps
: Error covariance of smoothed estimate `ps=[p0,p1,...,pf]`, and `pk` is a nxn matrix

xf
: Filtered state estimate `xf= [xf0,xf1,...,xff]`, and `xfk` is a column n-vector

pf
: Error covariance of filtered estimate `pf=[p0,p1,...,pf]`, and `pk` is a nxn matrix

Description

function which gives the Wiener estimate using the forward-backward Kalman filter formulation

Authors

C. B.

Name

wigner — 'time-frequency' wigner spectrum

```
[tab]=wigner(x,h,deltat,zp)
```

Parameters

tab
: wigner spectrum (lines correspond to the time variable)

x
: analyzed signal

h
: data window

deltat
: analysis time increment (in samples)

zp
: length of FFT's. `%pi/zp` gives the frequency increment.

Description

function which computes the 'time-frequency' wigner spectrum of a signal.

Name

window — compute symmetric window of various type

```
win_l=window('re',n)
win_l=window('tr',n)
win_l=window('hn',n)
win_l=window('hm',n)
win_l=window('kr',n,alpha)
[win_l,cwp]=window('ch',n,par)
```

Parameters

n
: window length

par
: parameter 2-vector `par=[dp,df])`, where `dp` (`0<dp<.5`) rules the main lobe width and `df` rules the side lobe height (`df>0`).

  Only one of these two value should be specified the other one should set equal to `-1`.

alpha
: kaiser window parameter `alpha >0`).

win
: window

cwp
: unspecified Chebyshev window parameter

Description

function which calculates various symmetric window for Disgital signal processing

The Kaiser window is a nearly optimal window function. `alpha` is an arbitrary positive real number that determines the shape of the window, and the integer `n` is the length of the window.

By construction, this function peaks at unity for `k = n/2`, i.e. at the center of the window, and decays exponentially towards the window edges. The larger the value of `alpha`, the narrower the window becomes; `alpha = 0` corresponds to a rectangular window. Conversely, for larger `alpha` the width of the main lobe increases in the Fourier transform, while the side lobes decrease in amplitude. Thus, this parameter controls the tradeoff between main-lobe width and side-lobe area.

alpha	window shape
0	Rectangular shape
5	Similar to the Hamming window
6	Similar to the Hanning window
8.6	Similar to the Blackman window

The Chebyshev window minimizes the mainlobe width, given a particular sidelobe height. It is characterized by an equiripple behavior, that is, its sidelobes all have the same height.

The Hanning and Hamming windows are quite similar, they only differ in the choice of one parameter `alpha`: `w=alpha+(1 - alpha)*cos(2*%pi*x/(n-1))` `alpha` is equal to 1/2 in Hanning window and to 0.54 in Hamming window.

Examples

```
// Hamming window
clf()
N=64;
w=window('hm',N);
subplot(121);plot2d(1:N,w,style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))
subplot(122)
n=256;[W,fr]=frmag(w,n);
plot2d(fr,20*log10(W),style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))

//Kaiser window
clf()
N=64;
w=window('kr',N,6);
subplot(121);plot2d(1:N,w,style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))
subplot(122)
n=256;[W,fr]=frmag(w,n);
plot2d(fr,20*log10(W),style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))

//Chebyshev window
clf()
N=64;
[w,df]=window('ch',N,[0.005,-1]);
subplot(121);plot2d(1:N,w,style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))
subplot(122)
n=256;[W,fr]=frmag(w,n);
plot2d(fr,20*log10(W),style=color('blue'))
set(gca(),'grid',[1 1]*color('gray'))
```

See Also

wfir , frmag , ffilt

Authors

Carey Bunks

Bibliography

IEEE. Programs for Digital Signal Processing. IEEE Press. New York: John Wiley and Sons, 1979. Program 5.2.

Name

yulewalk — least-square filter design

```
Hz = yulewalk(N,frq,mag)
```

Parameters

N
: integer (order of desired filter)

frq
: real row vector (non-decreasing order), frequencies.

mag
: non negative real row vector (same size as frq), desired magnitudes.

Hz
: filter `B(z)/A(z)`

Description

Hz = yulewalk(N,frq,mag) finds the N-th order iir filter

```
                 n-1         n-2
      B(z)   b(1)z     + b(2)z    + .... + b(n)
H(z)= ---- = ---------------------------------
                n-1       n-2
      A(z)    z   + a(2)z    + .... + a(n)
```

which matches the magnitude frequency response given by vectors frq and mag. Vectors frq and mag specify the frequency and magnitude of the desired frequency response. The frequencies in frq must be between 0.0 and 1.0, with 1.0 corresponding to half the sample rate. They must be in increasing order and start with 0.0 and end with 1.0.

Examples

```
f=[0,0.4,0.4,0.6,0.6,1];H=[0,0,1,1,0,0];Hz=yulewalk(8,f,H);
fs=1000;fhz = f*fs/2;
xbasc(0);xset('window',0);plot2d(fhz',H');
xtitle('Desired Frequency Response (Magnitude)')
[frq,repf]=repfreq(Hz,0:0.001:0.5);
xbasc(1);xset('window',1);plot2d(fs*frq',abs(repf'));
xtitle('Obtained Frequency Response (Magnitude)')
```

Name

zpbutt — Butterworth analog filter

```
[pols,gain]=zpbutt(n,omegac)
```

Parameters

n
: integer (filter order)

omegac
: real (cut-off frequency in Hertz)

pols
: resulting poles of filter

gain
: resulting gain of filter

Description

computes the poles of a Butterworth analog filter of order `n` and cutoff frequency omegac transfer function H(s) is calculated by `H(s)=gain/real(poly(pols,'s'))`

Authors

F. Delebecque INRIA

Name

zpch1 — Chebyshev analog filter

```
[poles,gain]=zpch1(n,epsilon,omegac)
```

Parameters

n
: integer (filter order)

epsilon
: real : ripple in the pass band (`0<epsilon<1`)

omegac
: real : cut-off frequency in Hertz

poles
: resulting filter poles

gain
: resulting filter gain

Description

Poles of a Type 1 Chebyshev analog filter. The transfer function is given by :

```
H(s)=gain/poly(poles,'s')
```

Authors

F.D.

Name

zpch2 — Chebyshev analog filter

```
[zeros,poles,gain]=zpch2(n,A,omegar)
```

Parameters

n
: integer : filter order

A
: real : attenuation in stop band (`A>1`)

omegar
: real : cut-off frequency in Hertz

zeros
: resulting filter zeros

poles
: resulting filter poles

gain
: Resulting filter gain

Description

Poles and zeros of a type 2 Chebyshev analog filter gain is the gain of the filter

```
H(s)=gain*poly(zeros,'s')/poly(poles,'s')
```

Authors

F.D.

Name

zpell — lowpass elliptic filter

```
[zeros,poles,gain]=zpell(epsilon,A,omegac,omegar)
```

Parameters

epsilon
: real : ripple of filter in pass band (`0<epsilon<1`)

A
: real : attenuation of filter in stop band (`A>1`)

omegac
: real : pass band cut-off frequency in Hertz

omegar
: real : stop band cut-off frequency in Hertz

zeros
: resulting zeros of filter

poles
: resulting poles of filter

gain
: resulting gain of filter

Description

Poles and zeros of prototype lowpass elliptic filter. `gain` is the gain of the filter

See Also

ell1mag , eqiir

Authors

F.D.

Simulated Annealing

Name

compute_initial_temp — A SA function which allows to compute the initial temperature of the simulated annealing

```
T_init = compute_initial_temp(x0,f,proba_init,ItMX,neigh_func,param_neigh_func)
```

Parameters

x0
: the starting point

f
: the objective function which will be send to the simulated annealing for optimization

proba_init
: the initial probability of accepting a bad solution (usually around 0.7)

ItMX
: the number of iterations of random walk (usually around 100)

neigh_func
: a function which returns a neighbor of a given point (see the help page of neigh_func to see the prototype of this function)

param_neigh_func
: some parameters (can be a list) which will be sent as parameters to neigh_func

T_init
: The initial temperature corresponding to the given probability of accepting a bad solution

Description

- This function computes an initial temperature given an initial probability of accepting a bad solution. This computation is based on some iterations of random walk.

Examples

```
deff('y=f(x)','y=sum(x.^2)');

x0 = [2 2];
Proba_start = 0.7;
It_Pre = 100;
x_test = neigh_func_default(x0);

T0 = compute_initial_temp(x0, f, Proba_start, It_Pre, neigh_func_default,[]
```

See Also

optim_sa , neigh_func_default , temp_law_default

Authors

collette
: Yann COLLETTE (ycollet@freesurf.fr)

Name

neigh_func_csa — The classical neighborhood relationship for the simulated annealing

```
x_neigh = neigh_func_csa(x_current,T,param)
```

Parameters

x_current
: the point for which we want to compute a neighbor

T
: the current temperature

param
: a vector with the same size than x_current. A normalisation vector which allows to distort the shape of the neighborhood. This parameter allows to take into account the differences of interval of variation between variables. By default, this parameter is set to a vector of ones.

x_neigh
: the computed neighbor

Description

- This function implements the classical neighborhood relationship for the simulated annealing. The neighbors distribution is a gaussian distribution which is more and more peaked as the temperature decrease.

See Also

neigh_func_default , temp_law_huang , optim_sa

Authors

collette
: Yann COLLETTE (ycollet@freesurf.fr)

Name

neigh_func_default — A SA function which computes a neighbor of a given point

```
x_neigh = neigh_func_default(x_current,T,param)
```

Parameters

x_current
: the point for which we want to compute a neighbor

T
: the current temperature

param
: a two columns vector. The first column correspond to the negative amplitude of variation and the second column corresponds to the positive amplitude of variation of the neighborhood. By default, the first column is a column of -0.1 and the second column is a column of 0.1.

x_neigh
: the computed neighbor

Description

- This function computes a neighbor of a given point. For example, for a continuous vector, a neighbor will be produced by adding some noise to each component of the vector. For a binary string, a neighbor will be produced by changing one bit from 0 to 1 or from 1 to 0.

Examples

```
// We produce a neighbor by adding some noise to each component of a given v
function x_neigh = neigh_func_default(x_current, T)
  sa_min_delta = -0.1*ones(size(x_current,1),size(x_current,2));
  sa_max_delta = 0.1*ones(size(x_current,1),size(x_current,2));
  x_neigh = x_current + (sa_max_delta - sa_min_delta).*rand(size(x_current,
endfunction
```

See Also

optim_sa , compute_initial_temp , temp_law_default

Authors

collette
: Yann COLLETTE (ycollet@freesurf.fr)

Name

neigh_func_fsa — The Fast Simulated Annealing neghborhood relationship

```
x_neigh = neigh_func_fsa(x_current,T,param)
```

Parameters

x_current
: the point for which we want to compute a neighbor

T
: the current temperature

param
: a vector with the same size than x_current. A normalisation vector which allows to distort the shape of the neighborhood. This parameter allows to take into account the differences of interval of variation between variables. By default, this parameter is set to a vector of ones.

x_neigh
: the computed neighbor

Description

- This function computes the FSA neighborhood of a given point. The corresponding distribution is a Cauchy distribution which is more and more peaked as the temperature decrease.

See Also

optim_sa , temp_law_fsa , neigh_func_default

Authors

collette
: Yann COLLETTE (ycollet@freesurf.fr)

Name

neigh_func_vfsa — The Very Fast Simulated Annealing neighborhood relationship

```
x_neigh = neigh_func_vfsa(x_current,T,param)
```

Parameters

x_current
: the point for which we want to compute a neighbor

T
: the current temperature

param
: a ones column vector. The column correspond to the amplitude of variation of the neighborhood. By default, the column is a column of 0.1.

x_neigh
: the computed neighbor

Description

- This function implements the Very Fast Simulated Annealing relationship. This distribution is more and more peaked as the temperature decrease.

See Also

optim_sa , neigh_func_vfsa , temp_law_huang

Authors

collette
: Yann COLLETTE (ycollet@freesurf.fr)

Name

optim_sa — A Simulated Annealing optimization method

```
[x_best,f_best,mean_list,var_list,f_history,temp_list,x_history] = optim_sa(x0,
```

Parameters

x0
: the initial solution

f
: the objective function to be optimized (the prototype if f(x))

ItExt
: the number of temperature decrease

ItInt
: the number of iterations during one temperature stage

T0
: the initial temperature (see compute_initial_temp to compute easily this temperature)

Log
: if %T, some information will be displayed during the run of the simulated annealing

temp_law
: the temperature decrease law (see temp_law_default for an example of such a function)

param_temp_law
: a structure (of any kind - it depends on the temperature law used) which is transmitted as a parameter to temp_law

neigh_func
: a function which computes a neighbor of a given point (see neigh_func_default for an example of such a function)

param_neigh_func
: a structure (of any kind like vector, list, it depends on the neighborhood function used) which is transmitted as a parameter to neigh_func

x_best
: the best solution found so far

f_best
: the objective function value corresponding to x_best

mean_list
: the mean of the objective function value for each temperature stage. A vector of float (optional)

var_list
: the variance of the objective function values for each temperature stage. A vector of float (optional)

f_history
: the computed objective function values for each iteration. Each input of the list corresponds to a temperature stage. Each input of the list is a vector of float which gathers all the objective function values computed during the corresponding temperature stage - (optional)

temp_list
: the list of temperature computed for each temperature stage. A vector of float (optional)

x_history
: the parameter values computed for each iteration. Each input of the list corresponds to a temperature stage. Each input of the list is a vector of input variables which corresponds to all the variables computed during the corresponding temperature stage - (optional - can slow down a lot the execution of optim_sa)

Description

- A Simulated Annealing optimization method.

Examples

```
function y = rastrigin(x)
  y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
endfunction

x0          = [2 2];
Proba_start = 0.7;
It_Pre      = 100;
It_extern   = 100;
It_intern   = 1000;
x_test = neigh_func_default(x0);

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, neigh_func_de

[x_opt, f_opt, sa_mean_list, sa_var_list] = optim_sa(x0, rastrigin, It_exte

printf('optimal solution:\n'); disp(x_opt);
printf('value of the objective function = %f\n', f_opt);

t = 1:length(sa_mean_list);
plot(t,sa_mean_list,'r',t,sa_var_list,'g');
```

See Also

compute_initial_temp , neigh_func_default , temp_law_default

Authors

collette
: Yann COLLETTE (ycollet@freesurf.fr)

Name

temp_law_csa — The classical temperature decrease law

```
T_out = temp_law_csa(T_in,step_mean,step_var,temp_stage,n,param)
```

Parameters

T_in
: the temperature at the current stage

step_mean
: the mean value of the objective function computed during the current stage

step_var
: the variance value of the objective function computed during the current stage

temp_stage
: the index of the current temperature stage

n
: the dimension of the decision variable (the x in f(x))

param
: not used for this temperature law

T_out
: the temperature for the temperature stage to come

Description

- This function implements the classical annealing temperature schedule (the one for which the convergence of the simulated annealing has been proven).

Examples

```
function y = rastrigin(x)
  y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
endfunction

x0 = [-1, -1];
Proba_start = 0.8;
It_intern = 1000;
It_extern = 30;
It_Pre    = 100;

printf('SA: the CSA algorithm\n');

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, neigh_func_defa
printf('Initial temperatore T0 = %f\n', T0);

[x_opt, f_opt, sa_mean_list, sa_var_list, temp_list] = optim_sa(x0, rastrigin

printf('optimal solution:\n'); disp(x_opt);
printf('value of the objective function = %f\n', f_opt);

scf();
```

```
subplot(2,1,1);
xtitle('Classical simulated annealing','Iteration','Mean / Variance');
t = 1:length(sa_mean_list);
plot(t,sa_mean_list,'r',t,sa_var_list,'g');
legend(['Mean','Variance']);
subplot(2,1,2);
xtitle('Temperature evolution','Iteration','Temperature');
plot(t,temp_list,'k-');
```

See Also

optim_sa , temp_law_huang , neigh_func_default

Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

Name

temp_law_default — A SA function which computed the temperature of the next temperature stage

```
T_next = temp_law_default(T,step_mean,step_var,temp_stage,n,param)
```

Parameters

T
: the temperature applied during the last temperature stage

step_mean
: the mean of the objective function values computed during the last temperature stage

step_var
: the variance of the obejective function values computed during the last temperature stage

temp_stage
: the index of the current temperature stage

n
: the dimension of the decision variable (the x in f(x))

param
: a float between 0 and 1. Corresponds to the decrease in temperature of the geometric law (0.9 by default)

T_next
: the new temperature to be applied for the next temperature stage

Description

- A SA function which computed the temperature of the next temperature stage

Examples

```
// This function implements the simple geometric temperature law
function T = temp_law_default(T, step_mean, step_var)
  _alpha = 0.9;
  T = _alpha*T;
endfunction
```

See Also

optim_sa , compute_initial_temp , neigh_func_default

Authors

collette
: Yann COLLETTE (ycollet@freesurf.fr)

Name

temp_law_fsa — The Szu and Hartley Fast simulated annealing

```
T_out = temp_law_fsa(T_in,step_mean,step_var,temp_stage,n,param)
```

Parameters

T_in
: the temperature at the current stage

step_mean
: the mean value of the objective function computed during the current stage

step_var
: the variance value of the objective function computed during the current stage

temp_stage
: the index of the current temperature stage

n
: the dimension of the decision variable (the x in f(x))

param
: not used for this temperature law

T_out
: the temperature for the temperature stage to come

Description

- This function implements the Fast simulated annealing of Szu and Hartley.

Examples

```
function y = rastrigin(x)
  y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
endfunction

x0 = [-1, -1];
Proba_start = 0.8;
It_intern = 1000;
It_extern = 30;
It_Pre    = 100;

printf('SA: the FSA algorithm\n');

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, neigh_func_defa
printf('Initial temperatore T0 = %f\n', T0);

[x_opt, f_opt, sa_mean_list, sa_var_list, temp_list] = optim_sa(x0, rastrigin

printf('optimal solution:\n'); disp(x_opt);
printf('value of the objective function = %f\n', f_opt);

scf();
subplot(2,1,1);
```

```
xtitle('Fast simulated annealing','Iteration','Mean / Variance');
t = 1:length(sa_mean_list);
plot(t,sa_mean_list,'r',t,sa_var_list,'g');
legend(['Mean','Variance']);
subplot(2,1,2);
xtitle('Temperature evolution','Iteration','Temperature');
plot(t,temp_list,'k-');
```

See Also

optim_sa , temp_law_huang , neigh_func_default

Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

Name

temp_law_huang — The Huang temperature decrease law for the simulated annealing

```
T_out = temp_law_huang(T_in,step_mean,step_var,temp_stage,n,param)
```

Parameters

T_in
: the temperature at the current stage

step_mean
: the mean value of the objective function computed during the current stage

step_var
: the variance value of the objective function computed during the current stage

temp_stage
: the index of the current temperature stage

n
: the dimension of the decision variable (the x in f(x))

param
: a float corresponding to the lambda parameter of the Huang temperature decrease law (0.01 by default)

T_out
: the temperature for the temperature stage to come

Description

- This function implements the Huang temperature decrease law for the simulated annealing.

Examples

```
function y = rastrigin(x)
  y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
endfunction

x0 = [-1, -1];
Proba_start = 0.8;
It_intern = 1000;
It_extern = 30;
It_Pre    = 100;

printf('SA: the Huang temperature decrease law\n');

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, neigh_func_defa
printf('Initial temperatore T0 = %f\n', T0);

[x_opt, f_opt, sa_mean_list, sa_var_list, temp_list] = optim_sa(x0, rastrigin

printf('optimal solution:\n'); disp(x_opt);
printf('value of the objective function = %f\n', f_opt);

scf();
```

```
subplot(2,1,1);
xtitle('Huang simulated annealing','Iteration','Mean / Variance');
t = 1:length(sa_mean_list);
plot(t,sa_mean_list,'r',t,sa_var_list,'g');
legend(['Mean','Variance']);
subplot(2,1,2);
xtitle('Temperature evolution','Iteration','Temperature');
plot(t,temp_list,'k-');
```

See Also

optim_sa , temp_law_csa , neigh_func_csa

Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

Name

temp_law_vfsa — This function implements the Very Fast Simulated Annealing from L. Ingber

```
T_out = temp_law_vfsa(T_in,step_mean,step_var,temp_stage,n, param)
```

Parameters

T_in
: the temperature at the current stage

step_mean
: the mean value of the objective function computed during the current stage

step_var
: the variance value of the objective function computed during the current stage

temp_stage
: the index of the current temperature stage

n
: the dimension of the decision variable (the x in f(x))

param
: a float: the 'c' parameter of the VFSA method (0.01 by default)

T_out
: the temperature for the temperature stage to come

Description

- This function implements the Very Fast Simulated Annealing from L. Ingber.

Examples

```
function y = rastrigin(x)
  y = x(1)^2+x(2)^2-cos(12*x(1))-cos(18*x(2));
endfunction

x0 = [-1, -1];
Proba_start = 0.8;
It_intern = 1000;
It_extern = 30;
It_Pre    = 100;

printf('SA: the VFSA algorithm\n');

T0 = compute_initial_temp(x0, rastrigin, Proba_start, It_Pre, neigh_func_defa
printf('Initial temperatore T0 = %f\n', T0);

[x_opt, f_opt, sa_mean_list, sa_var_list, temp_list] = optim_sa(x0, rastrigin

printf('optimal solution:\n'); disp(x_opt);
printf('value of the objective function = %f\n', f_opt);

scf();
subplot(2,1,1);
```

```
xtitle('VFSA simulated annealing','Iteration','Mean / Variance');
t = 1:length(sa_mean_list);
plot(t,sa_mean_list,'r',t,sa_var_list,'g');
legend(['Mean','Variance']);
subplot(2,1,2);
xtitle('Temperature evolution','Iteration','Temperature');
plot(t,temp_list,'k-');
```

See Also

optim_sa , neigh_func_vfsa , temp_law_huang

Authors

collette

Yann COLLETTE (ycollet@freesurf.fr)

Sound file handling

Name

analyze — frequency plot of a sound signal

Parameters

fmin,fmax,rate,points
: scalars. default values fmin=100,fmax=1500,rate=22050,points=8192;

Description

Make a frequency plot of the signal `w` with sampling rate `rate`. The data must be at least `points` long. The maximal frequency plotted will be `fmax`, the minimal `fmin`.

Examples

```
// At first we create 0.5 seconds of sound parameters.
t=soundsec(0.5);
// Then we generate the sound.
s=sin(440*t)+sin(220*t)/2+sin(880*t)/2;
[nr,nc]=size(t);
s(nc/2:nc)=sin(330*t(nc/2:nc));
analyze(s);
```

Name

auread — load .au sound file

```
y=auread(aufile)
y=auread(aufile,ext)
[y,Fs,bits]=auread(aufile)
[y,Fs,bits]=auread(aufile,ext)
```

Parameters

aufile
: string (The .au extension is appended if no extension is given)

Fs
: ...

[]
: integer, frequency sampling in Hz.

ext
: string (`'size'` or `'snd'`) or integer (to read n samples) or 1 x 2 integer vector [n1,n2] (to read from n1 to n2).

Description

Utility function to read `.au` sound file. `auread(aufile)` loads a sound file specified by the string aufile, returning the sampled data in y. Amplitude values are in the range [-1,+1].

Supports multi-channel data in the following formats: 8-bit mu-law, 8-, 16-, and 32-bit linear, and floating point.

`[y,Fs,bits]=auread(aufile)` returns the sample rate (Fs) in Hertz and the number of bits per sample used to encode the data in the file.

`auread(aufile,n)` returns the first n samples from each channel.

`auread(aufile,[n1,n2])` returns samples n1 to n2.

`auread(aufile,'size')` returns the size of the audio data contained in the file in place of the actual audio data, returning the vector as [samples channels].

`auread(aufile,'snd')` returns information about the sample and data as a tlist.

Examples

```
y=wavread('SCI/modules/sound/demos/chimes.wav');
// default is 8-bits mu-law
auwrite(y,TMPDIR+'/tmp.au');
y1=auread(TMPDIR+'/tmp.au');
maxi(abs(y-y1))
```

See Also

savewave , analyze , mapsound

Name

auwrite — writes .au sound file

```
auwrite(y,aufile)
auwrite(y,Fs,aufile)
auwrite(y,Fs,bits,aufile)
auwrite(y,Fs,bits,method,aufile)
```

Parameters

y
: real vector or matrix with entries in [-1,1].

aufile
: string (The .au extension is appended if no extension is given)

Fs
: integer, frequency sampling in Hz.

bits
: integer, number of bits in the encoding.

method
: string , `'mu'` (default) or `'linear'`, encoding method.

Description

Utility function to save `.au` sound file. `auwrite(y,aufile)` writes a sound file specified by the string aufile. The data should be arranged with one channel per column. Amplitude values outside the range [-1,+1] are ignored. Supports multi-channel data for 8-bit mu-law, and 8, 16, 32, 64 bits linear formats.

`auwrite(y,Fs,aufile)` specifies in `Fs` the sample rate of the data in Hertz.

`auwrite(y,Fs,bits,aufile)` selects the number of bits in the encoder. Allowable settings are bits in [8,16,32,64]. `auwrite(y,Fs,bits,method,aufile)` allows selection of the encoding method, which can be either 'mu' or 'linear'. Note that bits must be 8 for 'mu' choice. The default method is 8-bits mu-law enconding.

Examples

```
A=matrix(1:6,2,3);
auwrite(A/6,22050,64,'linear',TMPDIR+'/foo.au');
B=auread(TMPDIR+'/foo.au');
maxi(abs(A- round(B*6)))
```

See Also

auread , wavread , savewave , analyze , mapsound

Name

beep — Produce a beep sound

```
beep();
beep('on')
beep('off')
s=beep()
```

Description

`beep()` produces your computer's default beep sound.

`beep('on')` turns the beep on

`beep('off')` turns the beep off

`s=beep()` returns the current beep mode (on or off).

Authors

A.C

Name

lin2mu — linear signal to mu-law encoding

```
mu=lin2mu(y)
```

Parameters

y
: real vector

mu
: real vector

Description

Utility fct: converts linear signal to mu-law encoding. `mu = lin2mu(y)` converts linear audio signal amplitudes in the range -1 <= y <= 1 to mu-law in the range 0 <= mu <= 255.

See Also

mu2lin

Name

loadwave — load a sound wav file into scilab

```
x=loadwave(filename);
[x,y]=loadwave(filename);
```

Parameters

filename
: a string. The path of the wav file to be loaded

x
: a matrix one line for each channel

y
: vector as [data format, number of channels, samples per second per channel, estimate of bytes per second needed, byte alignment of a basic sample block, bits per sample, length of sound data in bytes, bytes per sample (per channel)].

Description

Reads a `.wav` sound file into Scilab as a matrix. If y is given, it is filled with information about the samples (See the message sent by `loadwave`).

Examples

```
// At first we create 0.5 seconds of sound parameters.
t=soundsec(0.5);
// Then we generate the sound: a two channels sound.
s=[sin(2*%pi*440*t);sin(2*%pi*350*t)];
savewave(TMPDIR+'/foo.wav',s);
s1=loadwave(TMPDIR+'/foo.wav');
max(abs(s1-s))
```

See Also

savewave, analyze, mapsound

Name

mapsound — Plots a sound map

```
mapsound (w,dt,fmin,fmax,simpl,rate)
```

Parameters

dt,fmin,fmax,simpl,rate
: scalars. default values dt=0.1,fmin=100,fmax=1500,simpl=1,rate=22050;

Description

Plots a sound map for a sound. It does FFT at time increments dt. `rate` is the sampling rate. `simpl` points are collected for speed reasons. `fmin` and `fmax` are used for graphic boundaries.

Examples

```
// At first we create 0.5 seconds of sound parameters.
t=soundsec(0.5);
// Then we generate the sound.
s=sin(440*t)+sin(220*t)/2+sin(880*t)/2;
[nr,nc]=size(t);
s(nc/2:nc)=sin(330*t(nc/2:nc));
mapsound(s);
```

Name

mu2lin — mu-law encoding to linear signal

```
mu=lin2mu(y)
```

Parameters

y
: real vector

mu
: real vector

Description

Utility fct: `y=mu2lin(mu)` converts mu-law encoded 8-bit audio signals, stored in the range 0 <= mu <= 255, to linear signal amplitude in the range -s < y < s where s = 32124/32768 ~= .9803. The input mu is often obtained using mget(...,'uc') to read byte-encoded audio files. Translation of C program by Craig Reese: IDA/Supercomputing Research Center Joe Campbell: Department of Defense

See Also

mu2lin

Name

playsnd — sound player facility

```
[]=playsnd(y)
[]=playsnd(y,rate,bits [,command])
```

Parameters

y
: A matrix. Each line descibe a channel

fs
: real number, sampling frequency (default value is 22050).

bits
: real number, number of bits (usually 8 or 16). Unused yet.

command
: Only used on Unix systems it gives the name of the command to use for playing sound (wav) files. The defaut value is `play`. If set `/dev/audio` then a 8 bits mu-law raw sound file is created and send to `/dev/audio`

Description

Plays a multi channel signal given by a Scilab matrix were sound is sampled at rate given by `rate`.

Examples

```
// a two channel signal
y=loadwave("SCI/modules/sound/demos/chimes.wav");
playsnd(y)
```

See Also

lin2mu , wavread

Name

savewave — save data into a sound wav file.

```
savewave(filename,x [, rate , nbits]);
```

Parameters

filename
: a string. The path of the output wav file

x
: a mxn matrix where m is the number of channels and n the number of samples for each channel

rate
: a scalar giving the sampling rate (number of sample per second) 22050 is the default value.

Description

save x into a wav sound file. you can transform other sound files into wav file with the `sox` program.

Examples

```
// At first we create 0.5 seconds of sound parameters.
t=soundsec(0.5);
// Then we generate the sound.
s=sin(2*%pi*440*t)+sin(2*%pi*220*t)/2+sin(2*%pi*880*t)/2;
[nr,nc]=size(t);
s(nc/2:nc)=sin(2*%pi*330*t(nc/2:nc));
savewave(TMPDIR+'/foo.wav',s);
```

See Also

loadwave, analyze, mapsound

Name

sound — sound player facility

```
sound(y [,fs,bits,command)
```

Parameters

y
: real vector

fs
: real number, sampling frequency in sample per second (default value is 22050)

bits
: real number, number of bits (unused)

command
: Only used on Unix systems it gives the name of the command to use for playing sound (wav) files. The defaut value is `aplay`. If set `/dev/audio` then a 8 bits mu-law raw sound file is created and send to `/dev/audio`

Description

`sound(y,fs)`plays the sound signal given by matrix `y` (with sample frequency fs). In fact this function is just a wrapper for `playsnd`. Values in y are assumed to be in the range -1.0 <= y <= 1.0. Values outside that range are truncated. The number of rows of `y` gives the number of channels. `sound(y)` plays the sound at the default sample rate of 22050 sample per second. `sound(y,fs,nbits)` plays the sound using `nbits` bits/sample if possible (it is in fact unused). Most platforms support bits=8 or 16.

Examples

```
// a two channel signal
y=loadwave("SCI/modules/sound/demos/chimes.wav");
sound(y)
```

See Also

playsnd

Name

soundsec — generates n sampled seconds of time parameter

```
t=soundsec(n [,rate])
```

Parameters

n
: an integer, the number of seconds to generate.

rate
: an integer, the number of samples per second.

Description

generates a vector coding time from 0 to nseconds at sampled rate `rate`.

Examples

```
// At first we create 0.5 seconds of sound parameters.
t=soundsec(0.5);
// Then we generate the sound.
s=sin(2*%pi*440*t);
analyze(s,200,600,22050);
```

See Also

playsnd , analyze

Name

wavread — load .wav sound file

```
y=wavread(wavfile)
y=wavread(wavfile,ext)
[y,Fs,bits]=wavread(wavfile)
[y,Fs,bits]=wavread(wavfile,ext)
```

Parameters

wavfile
: string (The .wav extension is appended if no extension is given)

Fs
: integer, frequency sampling in Hz (number of samples per second).

ext
: string (`'size'`) or string('info') or integer (to read n samples) or 1 x 2 integer vector [n1,n2] (to read from n1 to n2).

Description

Utility function to read `.wav` sound file. `wavread(wavfile)` loads a sound file specified by the string wavfile, returning the sampled data in y. Amplitude values are in the range [-1,+1]. Supports multi-channel data in the following formats: 8-bit mu-law, 8-, 16-, and 32-bit linear, and floating point.

`[y,Fs,bits]=wavread(wavfile)` returns the sample rate (Fs) in Hertz and the number of bits per sample used to encode the data in the file.

`wavread(wavfile,n)` returns the first n samples from each channel.

`wavread(wavfile,[n1,n2])` returns samples n1 to n2.

`wavread(wavfile,'size')` returns the size of the audio data contained in the file in place of the actual audio data, returning the vector as [channels samples].

`wavread(wavfile,'info')` returns information about the audio data contained in the file in place of the actual audio data, returning the vector as [data format, number of channels, samples per second per channel, estimate of bytes per second needed, byte alignment of a basic sample block, bits per sample, length of sound data in bytes, bytes per sample (per channel)].

Examples

```
wavread("SCI/modules/sound/demos/chimes.wav","size")
[y,Fs,bits]=wavread("SCI/modules/sound/demos/chimes.wav");Fs,bits
subplot(2,1,1)
plot2d(y(1,:)) // first channel
subplot(2,1,2)
plot2d(y(2,:)) // second channel
y=wavread("SCI/modules/sound/demos/chimes.wav",[1 5]) //the first five samples
```

See Also

auread, savewave, analyze, mapsound

Name

wavwrite — writes .wav sound file

```
wavwrite(y, wavfile)
wavwrite(y, Fs, wavfile)
wavwrite(y, Fs, nbits, wavfile)
```

Parameters

y
: real vector or matrix with entries in [-1,1].

wavfile
: string (The .wav extension is appended if no extension is given)

Fs
: integer, frequency sampling in Hz. 22500 is the default value.

nbits
: bit-depth 8, 16, 24, 32 bits. it describes the number of bits of information recorded for each sample. 16 is the default value.

Description

Utility function to save `.wav` sound file. `wavwrite(y,wavfile)` writes a sound file specified by the string wavfile. The data should be arranged with one channel per column. Amplitude values outside the range [-1,+1] are ignored.

`wavwrite(y,Fs,wavfile)` specifies in `Fs` the sample rate of the data in Hertz.

Examples

```
A=matrix(1:6,2,3);
wavwrite(A/6,TMPDIR+'/foo.wav');
B=wavread(TMPDIR+'/foo.wav');
```

See Also

auread, wavread, savewave, analyze, mapsound

Sparses Matrix

Name

full — sparse to full matrix conversion

```
X=full(sp)
```

Parameters

sp
: real or complex sparse (or full) matrix

X
: full matrix

Description

`X=full(sp)` converts the sparse matrix `sp` into its full representation. (If `sp` is already full then `X` equals `sp`).

Examples

```
sp=sparse([1,2;5,4;3,1],[1,2,3]);
A=full(sp)
```

See Also

sparse , sprand , speye

Name

gmres — Generalized Minimum RESidual method

```
[x,flag,err,iter,res] = gmres(A,b,rstr,tol,maxi,M,x0)
```

Parameters

A
: n-by-n matrix or function returning `A*x`

b
: right hand side vector

x0
: initial guess vector (default: zeros(n,1))

M
: preconditioner: matrix or function returning `M*x` (In the first case, default: eye(n,n))

rstr
: number of iterations between restarts (default: 10)

maxi
: maximum number of iterations (default: n)

tol
: error tolerance (default: 1e-6)

x
: solution vector

err
: final residual norm

iter
: number of iterations performed

flag

0 =
: `gmres` converged to the desired tolerance within `maxi` iterations

1 =
: no convergence given `maxi`

res
: residual vector

Description

GMRES
: solves the linear system `Ax=b` using the Generalized Minimal residual method with restarts.

Details
: of this algorithm are described in :

  "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", Barrett, Berry, Chan, Demmel, Donato, Dongarra, Eijkhout, Pozo, Romine, and Van der Vorst, SIAM Publications, 1993 (ftp netlib2.cs.utk.edu; cd linalg; get templates.ps).

"Iterative Methods for Sparse Linear Systems, Second Edition" Saad, SIAM Publications, 2003 (ftp ftp.cs.umn.edu; cd dept/users/saad/PS; get all_ps.zip).

Examples

```
// GMRES call x=gmres(A,b);
```

See Also

pcg , qmr

Authors

Sage Group (IRISA, 2005)

Name

ludel — utility function used with lufact

```
ludel(hand)
```

Parameters

hand
: handle to sparse lu factors (output of lufact)

Description

This function is used in conjunction with `lufact`. It clears the internal memory space used to store the result of `lufact`.

The sequence of commands `[p,r]=lufact(A);x=lusolve(p,b);ludel(p);` solves the sparse linear system `A*x = b` and clears `p`.

See Also

sparse , lufact , luget

Name

lufact — sparse lu factorization

```
[hand,rk]=lufact(A,prec)
```

Parameters

A
: square sparse matrix

hand
: handle to sparse lu factors

rk
: integer (rank of A)

prec
: a vector of size two `prec=[eps,reps]` giving the absolute and relative thresolds.

Description

`[hand,rk]=lufact(A)` performs the lu factorization of sparse matrix `A`. `hand` (no display) is used by `lusolve` (for solving linear system) and `luget` (for retrieving the factors). `hand` should be cleared by the command: `ludel(hand)`;

The A matrix needs not be full rank but must be square (since A is assumed sparse one may add zeros if necessary to squaring down A).

eps :
: The absolute magnitude an element must have to be considered as a pivot candidate, except as a last resort. This number should be set significantly smaller than the smallest diagonal element that is is expected to be placed in the matrix. the default value is `%eps`.

reps :
: This number determines what the pivot relative threshold will be. It should be between zero and one. If it is one then the pivoting method becomes complete pivoting, which is very slow and tends to fill up the matrix. If it is set close to zero the pivoting method becomes strict Markowitz with no threshold. The pivot threshold is used to eliminate pivot candidates that would cause excessive element growth if they were used. Element growth is the cause of roundoff error. Element growth occurs even in well-conditioned matrices. Setting the reps large will reduce element growth and roundoff error, but setting it too large will cause execution time to be excessive and will result in a large number of fill-ins. If this occurs, accuracy can actually be degraded because of the large number of operations required on the matrix due to the large number of fill-ins. A good value seems to be 0.001 which is the default value. The default is chosen by giving a value larger than one or less than or equal to zero. This value should be increased and the matrix resolved if growth is found to be excessive. Changing the pivot threshold does not improve performance on matrices where growth is low, as is often the case with ill-conditioned matrices. reps was choosen for use with nearly diagonally dominant matrices such as node- and modified-node admittance matrices. For these matrices it is usually best to use diagonal pivoting. For matrices without a strong diagonal, it is usually best to use a larger threshold, such as 0.01 or 0.1.

Examples

```
a=rand(5,5);b=rand(5,1);A=sparse(a);
[h,rk]=lufact(A);
```

```
x=lusolve(h,b);a*x-b
ludel(h)
```

See Also

sparse , lusolve , luget

Name

luget — extraction of sparse LU factors

```
[P,L,U,Q]=luget(hand)
```

Parameters

hand
: handle, output of `lufact`

P
: sparse permutation matrix

L
: sparse matrix, lower triangular if `hand` is obtained from a non singular matrix

U
: square non singular upper triangular sparse matrix with ones along the main diagonal

Q
: sparse permutation matrix

Description

`[P,L,U,Q]=luget(hand)` with `hand` obtained by the command `[hand,rk]=lufact(A)` with `A` a sparse matrix returns four sparse matrices such that `P*L*U*Q=A`.

The A matrix needs not be full rank but must be square (since A is assumed sparse one may add zeros if necessary to squaring down A).

If `A` is singular, the `L` matrix is column compressed (with `rk` independent nonzero columns): the nonsingular sparse matrix `Q'*inv(U)` column compresses `A`.

Examples

```
a=rand(5,2)*rand(2,5);A=sparse(a);
[hand,rk]=lufact(A);[P,L,U,Q]=luget(hand);
full(L), P*L*U*Q-A
clean(P*L*U*Q-A)
ludel(hand)
```

See Also

sparse , lusolve , luget , clean

Name

lusolve — sparse linear system solver

```
lusolve(hand,b)
lusolve(A,b)
```

Parameters

b
: full real matrix

A
: real square sparse invertible matrix

hand
: handle to a previously computed sparse lu factors (output of lufact)

Description

`x=lusolve(hand,b)` solves the sparse linear system `A*x = b`.

`[hand,rk]=lufact(A)` is the output of lufact.

`x=lusolve(A,b)` solves the sparse linear system `A*x = b`

Examples

```
non_zeros=[1,2,3,4];rows_cols=[1,1;2,2;3,3;4,4];
sp=sparse(rows_cols,non_zeros);
[h,rk]=lufact(sp);x=lusolve(h,[1;1;1;1]);ludel(h)
rk,sp*x

non_zeros=[1,2,3,4];rows_cols=[1,1;2,2;3,3;4,4];
sp=sparse(rows_cols,non_zeros);
x=lusolve(sp,-ones(4,1));
sp*x
```

See Also

sparse , lufact , slash , backslash

Name

mtlb_sparse — convert sparse matrix

```
Y=mtlb_sparse(X)
```

Parameters

X
: sparse matrix

Y
: sparse matrix in Matlab format

Description

`Y=mtlb_sparse(X)` is used to convert `X`, a Scilab sparse matrix, to Matlab format. `Y` is the a variable with type 7, i.e. `type(Y)` is equal to 7. This function should be used in mexfiles (a Matlab mexfile containing sparse matrices can be used only if the Scilab sparse matrices are converted to that format). The functions `full` and `spget` work with this format.

Other operations and functions using this format can be overloaded with Scilab functions using the prefix "%msp". For instance the function `%msp_p(x)` (see SCIDIR/macros/percent directory) is used to display such "type 7" objects.

Examples

```
X=sparse(rand(2,2)); Y=mtlb_sparse(X);
Y, full(Y), [ij,v,mn]=spget(Y)
```

See Also

full , spget

Name

nnz — number of non zero entries in a matrix

```
n=nnz(X)
```

Parameters

X
: real or complex sparse (or full) matrix

n
: integer, the number of non zero elements in X

Description

`nnz` counts the number of non zero entries in a sparse or full matrix

Examples

```
sp=sparse([1,2;4,5;3,10],[1,2,3]);
nnz(sp)
a=[1 0 0 0 2];
nnz(a)
```

See Also

spget

Name

pcg — precondioned conjugate gradient

```
[x, flag, err, iter, res] = pcg(A, b [, tol [, maxIter [, M [, M2 [, x0 [, verb
[x, flag, err, iter, res] = pcg(A, b [key=value,...])
```

Parameters

A,
: a matrix, or a function, or a list computing `A*x` for each given `x`. The following is a description of the computation of A*x depending on the type of A.

  - `matrix.` If A is a matrix, it can be dense or sparse
  - `function.` If A is a function, it must have the following header :

    ```
    function y = A ( x )
    ```

  - `list.` If A is a list, the first element of the list is expected to be a function and the other elements in the list are the arguments of the function, from index 2 to the end. When the function is called, the current value of x is passed to the function as the first argument. The other arguments passed are the one given in the list.

b
: right hand side vector (size: nx1)

tol
: error relative tolerance (default: 1e-8). The termination criteria is based on the 2-norm of the residual r=b-Ax, divided by the 2-norm of the right hand side b.

maxIter
: maximum number of iterations (default: n)

M
: preconditioner: full or sparse matrix or function returning `M\x` (default: none)

M2
: preconditioner: full or sparse matrix or function returning `M2\x` for each `x` (default: none)

x0
: initial guess vector (default: zeros(n,1))

verbose
: set to 1 to enable verbose logging (default 0)

x
: solution vector

flag
: 0 if `pcg` converged to the desired tolerance within `maxi` iterations, 1 else

err
: final relative norm of the residual (the 2-norm of the right-hand side b is used)

iter
: number of iterations performed

res
: vector of the residual relative norms

Description

Solves the linear system `Ax=b` using the conjugate gradient method with or without preconditioning. The preconditionning should be defined by a symmetric positive definite matrix `M`, or two matrices `M1` and `M2` such that `M=M1*M2`. in the case the function solves `inv(M)*A*x = inv(M)*b` for `x`. `M`, `M1` and `M2` can be Scilab functions with calling sequence `y=Milx(x)` which computes the corresponding left division `y=Mi\x`.

The `A` matrix must be a symmetric positive definite matrix (full or sparse) or a function with calling sequence `y=Ax(x)` which computes `y=A*x`

Example with well conditionned and ill conditionned problems

In the following example, two linear systems are solved. The first maxtrix has a condition number equals to ~0.02, which makes the algorithm converge in exactly 10 iterations. Since this is the size of the matrix, it is an expected behaviour for a gradient conjugate method. The second one has a low condition number equals to 1.d-6, which makes the algorithm converge in a larger 22 iterations. This is why the parameter maxIter is set to 30. See below for other examples of the "key=value" syntax.

```
//Well conditionned problem
A=[ 94  0   0   0    0   28  0   0   32  0
    0   59  13  5    0   0   0   10  0   0
    0   13  72  34   2   0   0   0   0   65
    0   5   34  114  0   0   0   0   0   55
    0   0   2   0    70  0   28  32  12  0
    28  0   0   0    0   87  20  0   33  0
    0   0   0   0    28  20  71  39  0   0
    0   10  0   0    32  0   39  46  8   0
    32  0   0   0    12  33  0   8   82  11
    0   0   65  55   0   0   0   0   11  100];

b=ones(10,1);
[x, fail, err, iter, res]=pcg(A,b,1d-12,15);
mprintf("      fail=%d, iter=%d, errrel=%e\n",fail,iter,err)

//Ill contionned one
A=[ 894     0   0     0   0   28  0   0   1000  70000
     0      5   13    5   0   0   0   0   0     0
     0      13  72    34  0   0   0   0   0     6500
     0      5   34    1   0   0   0   0   0     55
     0      0   0     0   70  0   28  32  12    0
     28     0   0     0   0   87  20  0   33    0
     0      0   0     0   28  20  71  39  0     0
     0      0   0     0   32  0   39  46  8     0
     1000   0   0     0   12  33  0   8   82    11
     70000  0   6500  55  0   0   0   0   11    100];

 [x, fail, err, iter, res]=pcg(A,b,maxIter=30,tol=1d-12);
```

```
mprintf("      fail=%d, iter=%d, errrel=%e\n",fail,iter,err)
```

Examples with A given as a sparse matrix, or function, or list

The following example shows that the method can handle sparse matrices as well. It also shows the case where a function, computing the right-hand side, is given to the "pcg" primitive. The final case shown by this example, is when a list is passed to the primitive.

```
//Well conditionned problem
A=[ 94  0   0   0    0   28  0   0   32  0
  0   59  13  5    0   0   0   10  0   0
  0   13  72  34   2   0   0   0   0   65
  0   5   34  114  0   0   0   0   0   55
  0   0   2   0    70  0   28  32  12  0
  28  0   0   0    0   87  20  0   33  0
  0   0   0   0    28  20  71  39  0   0
  0   10  0   0    32  0   39  46  8   0
  32  0   0   0    12  33  0   8   82  11
  0   0   65  55   0   0   0   0   11  100];
b=ones(10,1);

// Convert A into a sparse matrix
Asparse=sparse(A);
[x, fail, err, iter, res]=pcg(Asparse,b,maxIter=30,tol=1d-12);
mprintf("      fail=%d, iter=%d, errrel=%e\n",fail,iter,err)

// Define a function which computes the right-hand side.
function y=Atimesx(x)
  A=[ 94  0   0   0    0   28  0   0   32  0
    0   59  13  5    0   0   0   10  0   0
    0   13  72  34   2   0   0   0   0   65
    0   5   34  114  0   0   0   0   0   55
    0   0   2   0    70  0   28  32  12  0
    28  0   0   0    0   87  20  0   33  0
    0   0   0   0    28  20  71  39  0   0
    0   10  0   0    32  0   39  46  8   0
    32  0   0   0    12  33  0   8   82  11
    0   0   65  55   0   0   0   0   11  100];
  y=A*x
endfunction

// Pass the script Atimesx to the primitive
[x, fail, err, iter, res]=pcg(Atimesx,b,maxIter=30,tol=1d-12);
mprintf("      fail=%d, iter=%d, errrel=%e\n",fail,iter,err)

// Define a function which computes the right-hand side.
function y=Atimesxbis(x,A)
  y=A*x
endfunction

// Pass a list to the primitive
Alist = list(Atimesxbis,Asparse);
[x, fail, err, iter, res]=pcg(Alist,b,maxIter=30,tol=1d-12);
mprintf("      fail=%d, iter=%d, errrel=%e\n",fail,iter,err)
```

Examples with key=value syntax

The following example shows how to pass arguments with the "key=value" syntax. This allows to set non-positionnal arguments, that is, to set arguments which are not depending on their order in the list of arguments. The available keys are the names of the optional arguments, that is : tol, maxIter, %M, %M2, x0, verbose. Notice that, in the following example, the verbose option is given before the maxIter option. Without the "key=value" syntax, the positionnal arguments would require that maxIter come first and verbose after.

```
// Example of an argument passed with key=value syntax
A=[100,1;1,10];
b=[101;11];
[xcomputed, flag, err, iter, res]=pcg(A,b,verbose=1);

// With key=value syntax, the order does not matter
[xcomputed, flag, err, iter, res]=pcg(A,b,verbose=1,maxIter=0);
```

See Also

backslash, qmr, gmres

Authors

Sage Group, IRISA, 2004

Serge Steer, INRIA, 2006

Michael Baudin, INRIA, 2008-2009

References

"Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", Barrett, Berry, Chan, Demmel, Donato, Dongarra, Eijkhout, Pozo, Romine, and Van der Vorst, SIAM Publications, 1993, ftp netlib2.cs.utk.edu/linalg/templates.ps

"Iterative Methods for Sparse Linear Systems, Second Edition", Saad, SIAM Publications, 2003, ftp ftp.cs.umn.edu/dept/users/saad/PS/all_ps.zip

Name

qmr — quasi minimal resiqual method with preconditioning

```
[x,flag,err,iter,res] = qmr(A,b,x0,M1,M1p,M2,M2p,maxi,tol)
```

Parameters

A
: matrix of size n-by-n or function returning `A*x`

b
: right hand side vector

x0
: initial guess vector (default: zeros(n,1))

M1
: left preconditioner: matrix or function returning `M1*x` (In the first case, default: eye(n,n))

M1p
: must only be provided when `M1` is a function. In this case `M1p` is the function which returns `M1'*x`

M2
: right preconditioner: matrix or function returning `M2*x` (In the first case, default: eye(n,n))

M2p
: must only be provided when `M2` is a function. In this case `M2p` is the function which returns `M2'*x`

maxi
: maximum number of iterations (default: n)

tol
: error tolerance (default: 1000*%eps)

x
: solution vector

flag

  0 =
  : `gmres` converged to the desired tolerance within `maxi` iterations

  1 =
  : no convergence given `maxi`

res
: residual vector

err
: final residual norm

iter
: number of iterations performed

Description

Solves the linear system `Ax=b` using the Quasi Minimal Residual Method with preconditioning.

See Also

gmres

Authors

SAGE Group, IRISA 2005

Name

readmps — reads a file in MPS format

```
mps= readmps (file-name,bounds [,maxsizes]);
```

Parameters

file-name

bounds
: 2-vector `[lowbound,upbound]` , default lower ans upper bounds

maxsizes
: 3-vector `[maxm,maxn,maxnza]` Maximum number of contraints and variables, maximum number of nonzeros entries in the LP constraint matrix. If omitted readmps reads the file once just to compute these numbers.

mps
: tlist with following fields

  irobj
  : integer (index of the objective row).

  namec
  : character string (Name of the objective).

  nameb
  : character string (Name of the right hand side).

  namran
  : character string (Name of the ranges section).

  nambnd
  : character string (Name of the bounds section).

  name
  : character string (Name of the LP problem).

  rownames
  : character string column vector (Name of the rows). colnames : character string row vector (Name of the columns).

  rowstat
  : integer vector, row types:

    1
    : row type is "="

    2
    : row type is ">="

    3
    : row type is "<="

    4
    : objective row

    5
    : other free row

rowcode
real matrix `[hdrowcd,lnkrow]` with

hdrowcd
real vector (Header to the linked list of rows with the same codes).

lnkrow
integer vector (Linked list of rows with the same codes).

colcode
real matrix `[hdcolcd,lnkcol]` with

hdcolcd
integer vector (Header to the linked list of columns with the same codes).

lnkcol
integer vector (Linked list of columns with the same codes).

rownmbs
integer vector (Row numbers of nonzeros in columns of matrix A.)

colpnts
integer vector (Pointers to the beginning of columns of matrix A).

acoeff
real vector (Array of nonzero elements for each column).

rhs
:real vector (Right hand side of the linear program).

ranges
real vector of constraint ranges.

bounds
real matrix `[lbounds,ubounds]` with

ubounds
full column vector of upper bounds

lbounds
full column vector of lower bounds

stavar
full column vector of variable status

0
:standard (non negative) variable

1
upper bounded variable

2
lower bounded variable

3
lower and upper bounded variable

4
minus infinity type variable i.e -inf<x<=u

5
plus infinity type variable i.e l<=x< inf

6
: fixed type variable i.e l=x=u

-k
: free variable

Description

`readmps`. Utility function: reads a file containing description of an LP problem given in MPS format. It is an interface with the program `rdmps1.f` of hopdm (J. Gondzio). For a description of the variables, see the file rdmps1.f.

MPS format is a standard ASCII medium for LP codes. MPS format is described in more detail in Murtagh's book:

Murtagh B. (1981). Advanced Linear Programming, McGrew-Hill, New York, 1981.

Examples

```
//Let the LP problem:
//objective:
//   min     XONE + 4 YTWO + 9 ZTHREE
//constraints:
//  LIM1:    XONE +   YTWO            < = 5
//  LIM2:    XONE +            ZTHREE > = 10
// MYEQN:         -   YTWO  +  ZTHREE   = 7
//Bounds
//  0 < = XONE < = 4
// -1 < = YTWO < = 1

//Generate MPS file
txt=['NAME          TESTPROB'
     'ROWS'
     ' N  COST'
     ' L  LIM1'
     ' G  LIM2'
     ' E  MYEQN'
     'COLUMNS'
     '    XONE      COST                 1   LIM1                 1'
     '    XONE      LIM2                 1'
     '    YTWO      COST                 4   LIM1                 1'
     '    YTWO      MYEQN               -1'
     '    ZTHREE    COST                 9   LIM2                 1'
     '    ZTHREE    MYEQN                1'
     'RHS'
     '    RHS1      LIM1                 5   LIM2                10'
     '    RHS1      MYEQN                7'
     'BOUNDS'
     ' UP BND1      XONE                 4'
     ' LO BND1      YTWO                -1'
     ' UP BND1      YTWO                 1'
     'ENDATA'];
mputl(txt,TMPDIR+'/test.mps')
//Read the MPS file
P=readmps(TMPDIR+'/test.mps',[0 10^30])
```

```
//Convert it to linpro format
LP=mps2linpro(P)
//Solve it with linpro
[x,lagr,f]=linpro(LP(2:$))
```

See Also

mps2linpro

Name

sparse — sparse matrix definition

```
sp=sparse(X)
sp=sparse(ij,v [,mn])
```

Parameters

X
: real or complex full (or sparse) matrix

ij
: two columns integer matrix (indices of non-zeros entries)

v
: vector

mn
: integer vector with two entries (row-dimension, column-dimension)

sp
: sparse matrix

Description

`sparse` is used to build a sparse matrix. Only non-zero entries are stored.

`sp = sparse(X)` converts a full matrix to sparse form by squeezing out any zero elements. (If `X` is already sparse `sp` is `X`).

`sp=sparse(ij,v [,mn])` builds an `mn(1)`-by-`mn(2)` sparse matrix with `sp(ij(k,1),ij(k,2))=v(k)`. `ij` and `v` must have the same column dimension. If optional `mn` parameter is not given the `sp` matrix dimensions are the max value of `ij(:,1)` and `ij(:,2)` respectively.

Operations (concatenation, addition, etc,) with sparse matrices are made using the same syntax as for full matrices.

Elementary functions are also available (`abs,maxi,sum,diag,...`) for sparse matrices.

Mixed operations (full-sparse) are allowed. Results are full or sparse depending on the operations.

Examples

```
sp=sparse([1,2;4,5;3,10],[1,2,3])
size(sp)
x=rand(2,2);abs(x)-full(abs(sparse(x)))
```

See Also

full , spget , sprand , speye , lufact

Name

spchol — sparse cholesky factorization

```
[R,P] = spchol(X)
```

Parameters

X
: symmetric positive definite real sparse matrix

P
: permutation matrix

R
: cholesky factor

Description

`[R,P] = spchol(X)` produces a lower triangular matrix `R` such that `P*R*R'*P' = X`.

Examples

```
X=[
3.,  0.,  0.,  2.,  0.,  0.,  2.,  0.,  2.,  0.,  0. ;
0.,  5.,  4.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0. ;
0.,  4.,  5.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0. ;
2.,  0.,  0.,  3.,  0.,  0.,  2.,  0.,  2.,  0.,  0. ;
0.,  0.,  0.,  0. , 5.,  0.,  0.,  0.,  0.,  0.,  4. ;
0.,  0.,  0.,  0.,  0.,  4.,  0.,  3.,  0.,  3.,  0. ;
2.,  0.,  0.,  2.,  0.,  0.,  3.,  0.,  2.,  0.,  0. ;
0.,  0.,  0.,  0.,  0.,  3.,  0.,  4.,  0.,  3.,  0. ;
2.,  0.,  0.,  2.,  0.,  0.,  2.,  0.,  3.,  0.,  0. ;
0.,  0.,  0.,  0.,  0.,  3.,  0.,  3.,  0.,  4.,  0. ;
0.,  0.,  0.,  0.,  4.,  0.,  0.,  0.,  0.,  0.,  5.];
X=sparse(X);[R,P] = spchol(X);
max(P*R*R'*P'-X)
```

See Also

sparse , lusolve , luget , chol

Name

spcompack — converts a compressed adjacency representation

Parameters

xadj
: integer vector of length (n+1).

xlindx
: integer vector of length n+1 (pointers).

lindx
: integer vector

adjncy
: integer vector

Description

```
Utility fonction spcompak is used to convert a compressed adjacency
representation into standard adjacency representation.
```

Examples

```
// A is the sparse matrix:
A=[1,0,0,0,0,0,0;
   0,1,0,0,0,0,0;
   0,0,1,0,0,0,0;
   0,0,1,1,0,0,0;
   0,0,1,1,1,0,0;
   0,0,1,1,0,1,0;
   0,0,1,1,0,1,1];
A=sparse(A);
//For this matrix, the standard adjacency representation is given by:
xadj=[1,2,3,8,12,13,15,16];
adjncy=[1, 2, 3,4,5,6,7, 4,5,6,7, 5, 6,7, 7];
//(see sp2adj).
// increments in vector xadj give the number of non zero entries in each column
// ie there is 2-1=1 entry in the column 1
//    there is 3-2=1 entry in the column 2
//    there are 8-3=5 entries in the column 3
//              12-8=4                      4
//etc
//The row index of these entries is given by the adjncy vector
// for instance,
// adjncy (3:7)=adjncy(xadj(3):xadj(4)-1)=[3,4,5,6,7]
// says that the 5=xadj(4)-xadj(3) entries in column 3 have row
// indices 3,4,5,6,7.
//In the compact representation, the repeated sequences in adjncy
//are eliminated.
//Here in adjncy the sequences 4,5,6,7  and 7 are eliminated.
```

```
//The standard structure (xadj,adjncy) takes the compressed form (lindx,xlindx)
lindx=[1, 2, 3,4,5,6,7, 5, 6,7];
xlindx=[1,2,3,8,9,11];
//(Columns 4 and 7 of A are eliminated).
//A can be reconstructed from (xadj,xlindx,lindx).
[xadj,adjncy,anz]= sp2adj(A);
adjncy-spcompack(xadj,xlindx,lindx)
```

See Also

sp2adj , adj2sp , spget

Name

spget — retrieves entries of sparse matrix

```
[ij,v,mn]=spget(sp)
```

Parameters

sp
: real or complex sparse matrix

ij
: two columns integer matrix (indices of non-zeros entries)

mn
: integer vector with two entries (row-dimension, column-dimension)

v
: column vector

Description

`spget` is used to convert the internal representation of sparse matrices into the standard `ij, v` representation.

Non zero entries of `sp` are located in rows and columns with indices in `ij`.

Examples

```
sp=sparse([1,2;4,5;3,10],[1,2,3])
[ij,v,mn]=spget(sp);
```

See Also

sparse , sprand , speye , lufact

Special Functions

Name

besseli — Modified Bessel functions of the first kind (I sub alpha).
besselj — Bessel functions of the first kind (J sub alpha).
besselk — Modified Bessel functions of the second kind (K sub alpha).
bessely — Bessel functions of the second kind (Y sub alpha).
besselh — Bessel functions of the third kind (aka Hankel functions)

```
y = besseli(alpha,x [,ice])
y = besselj(alpha,x [,ice])
y = besselk(alpha,x [,ice])
y = bessely(alpha,x [,ice])
y = besselh(alpha,x)
y = besselh(alpha,K,x [,ice])
```

Parameters

x
: real or complex vector.

alpha
: real vector

ice
: integer flag, with default value 0

K
: integer, with possible values 1 or 2, the Hankel function type.

Description

Warning: the semantics of these functions changes between Scilab-3.0 and Scilab-3.1. The old semantics is available for compatibility using the oldbesseli, oldbesselj, oldbesselk, oldbessely functions.

- `besseli(alpha,x)` computes modified Bessel functions of the first kind (I sub alpha), for real order `alpha` and argument `x`. `besseli(alpha,x,1)` computes `besseli(alpha,x).*exp(-abs(real(x)))`.
- `besselj(alpha,x)` computes Bessel functions of the first kind (J sub alpha), for real order `alpha` and argument `x`. `besselj(alpha,x,1)` computes `besselj(alpha,x).*exp(-abs(imag(x)))`.
- `besselk(alpha,x)` computes modified Bessel functions of the second kind (K sub alpha), for real order `alpha` and argument `x`. `besselk(alpha,x,1)` computes `besselk(alpha,x).*exp(x)`.
- `bessely(alpha,x)` computes Bessel functions of the second kind (Y sub alpha), for real order `alpha` and argument `x`. `bessely(alpha,x,1)` computes `bessely(alpha,x).*exp(-abs(imag(x)))`.
- `besselh(alpha [,K] ,x)` computes Bessel functions of the third kind (Hankel function H1 or H2 depending on `K`), for real order `alpha` and argument `x`. If omitted `K` is supposed to be equal to 1. `besselh(alpha,1,x,1)` computes `besselh(alpha,1,x).*exp(-%i*x)` and `besselh(alpha,2,x,1)` computes `besselh(alpha,2,x).*exp(%i*x)`

Remarks

If `alpha` and `x` are arrays of the same size, the result `y` is also that size. If either input is a scalar, it is expanded to the other input's size. If one input is a row vector and the other is a column vector, the result`y` is a two-dimensional table of function values.

Y_alpha and J_alpha Bessel functions are 2 independant solutions of the Bessel 's differential equation :

$$x^2 \cdot \frac{d^2y}{dx^2} + x \cdot \frac{dy}{dx} + (x^2 - \alpha^2) \cdot y = 0, \alpha \geq 0$$

K_alpha and I_alpha modified Bessel functions are 2 independant solutions of the modified Bessel 's differential equation :

$$x^2 \cdot \frac{d^2y}{dx^2} + x \cdot \frac{dy}{dx} - (x^2 + \alpha^2) \cdot y = 0, \alpha \geq 0$$

H^1_alpha and H^2_alpha, the Hankel functions of first and second kind, are linear linear combinations of Bessel functions of the first and second kinds:

$$H_\alpha^1(z) = J_\alpha(z) + i \cdot Y_\alpha(z)$$
$$H_\alpha^2(z) = J_\alpha(z) - i \cdot Y_\alpha(z)$$

Examples

```
//  besselI functions
// ==================
   x = linspace(0.01,10,5000)';
   xbasc()
   subplot(2,1,1)
   plot2d(x,besseli(0:4,x), style=2:6)
   legend('I'+string(0:4),2);
   xtitle("Some modified Bessel functions of the first kind")
   subplot(2,1,2)
   plot2d(x,besseli(0:4,x,1), style=2:6)
   legend('I'+string(0:4),1);
   xtitle("Some modified scaled Bessel functions of the first kind")

// besselJ functions
// =================
   x = linspace(0,40,5000)';
   xbasc()
   plot2d(x,besselj(0:4,x), style=2:6, leg="J0@J1@J2@J3@J4")
   legend('I'+string(0:4),1);
   xtitle("Some Bessel functions of the first kind")

// use the fact that J_(1/2)(x) = sqrt(2/(x pi)) sin(x)
// to compare the algorithm of besselj(0.5,x) with a more direct formula
   x = linspace(0.1,40,5000)';
   y1 = besselj(0.5, x);
   y2 = sqrt(2 ./(%pi*x)).*sin(x);
   er = abs((y1-y2)./y2);
   ind = find(er > 0 & y2 ~= 0);
```

```
   xbasc()
   subplot(2,1,1)
   plot2d(x,y1,style=2)
   xtitle("besselj(0.5,x)")
   subplot(2,1,2)
   plot2d(x(ind), er(ind), style=2, logflag="nl")
   xtitle("relative error between 2 formulae for besselj(0.5,x)")

// besselK functions
// =================
   x = linspace(0.01,10,5000)';
   xbasc()
   subplot(2,1,1)
   plot2d(x,besselk(0:4,x), style=0:4, rect=[0,0,6,10])
   legend('K'+string(0:4),1);
   xtitle("Some modified Bessel functions of the second kind")
   subplot(2,1,2)
   plot2d(x,besselk(0:4,x,1), style=0:4, rect=[0,0,6,10])
   legend('K'+string(0:4),1);
   xtitle("Some modified scaled Bessel functions of the second kind")

// besselY functions
// =================
   x = linspace(0.1,40,5000)'; // Y Bessel functions are unbounded  for x -> 0+
   xbasc()
   plot2d(x,bessely(0:4,x), style=0:4, rect=[0,-1.5,40,0.6])
   legend('Y'+string(0:4),4);
   xtitle("Some Bessel functions of the second kind")


// besselH functions
// =================
   x=-4:0.025:2; y=-1.5:0.025:1.5;
   [X,Y] = ndgrid(x,y);
   H = besselh(0,1,X+%i*Y);
   clf();f=gcf();
   xset("fpf"," ")
   f.color_map=jetcolormap(16);
   contour2d(x,y,abs(H),0.2:0.2:3.2,strf="034",rect=[-4,-1.5,3,1.5])
   legends(string(0.2:0.2:3.2),1:16,"ur")
   xtitle("Level curves of |H1(0,z)|")
```

Authors

Amos, D. E., (SNLA)
Daniel, S. L., (SNLA)
Weston, M. K., (SNLA)

Used Functions

The source codes can be found in routines/calelm

Slatec : dbesi.f, zbesi.f, dbesj.f, zbesj.f, dbesk.f, zbesk.f, dbesy.f, zbesy.f, zbesh.f

Drivers to extend definition area (Serge Steer INRIA): dbesig.f, zbesig.f, dbesjg.f, zbesjg.f, dbeskg.f, zbeskg.f, dbesyg.f, zbesyg.f, zbeshg.f

Name

beta — beta function

```
z = beta(x,y)
```

Parameters

x, y
: 2 positive reals or 2 matrices (or vectors) of positive reals of same size.

z
: a real or a matrix of the same size than `x` with `z(i,j) = beta(x(i,j),y(i,j))`.

Description

Computes the complete beta function :

$$B(x,y) = \int_0^1 t^{x-1} \cdot (1-t)^{y-1} \cdot dt = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$$

For small `x` and `y` the algorithm uses the expression in function of the gamma function, else it applies the exponential function onto the result of the `betaln` function provided with the DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameter (see cdfbet for more information about DCDFLIB).

Examples

```
// example 1 :
beta(5,2) - beta(2,5)   // symetry (must be exactly 0)
beta(0.5,0.5)           // exact value is pi

// example 2 : an error study based on the relation  B(1,x) = 1/x
// (computing 1/x must lead to only a relative error of eps_m, so
//  it may be used near as a reference to evaluate the error in B(1,x))
x = logspace(-8,8,20000)';
e = beta(ones(x),x) - (1)./x;
er = abs(e) .* x;
ind = find(er ~= 0);
eps = ones(x(ind))*number_properties("eps");
xbasc()
plot2d(x(ind),[er(ind) eps 2*eps],style=[1 2 3],logflag="ll",leg="er@eps_m@2 eps
xtitle("approximate relative error in computing beta(1,x)")
xselect()

// example 3 : plotting the beta function
t = linspace(0.2,10,60);
X = t'*ones(t); Y = ones(t')*t;
Z = beta(X,Y);
xbasc()
plot3d(t, t, Z, flag=[2 4 4], leg="x@y@z", alpha=75, theta=30)
xtitle("The beta function on [0.2,10]x[0.2,10]")
xselect()
```

See Also

gamma, cdfbet

Name

calerf — computes error functions.

Parameters

x
: real vector or matrix

flag
: integer indicator

y
: real vector or matrix (of same size than x)

Description

`calerf(x,0)` computes the error function `erf(x)`

`calerf(x,1)` computes the complementary error function `erfc(x)`

`calerf(x,2)` computes the scaled complementary error function `erfcx(x)`

Examples

```
deff('y=f(t)','y=exp(-t^2)');
calerf(1,0)
2/sqrt(%pi)*intg(0,1,f)
```

See Also

erf, erfc, erfcx

Authors

W. J. Cody (code from Netlib (specfun))

Name

dlgamma — derivative of gammaln function, psi function

```
y = dlgamma(x)
```

Parameters

x
: real vector

y
: real vector with same size.

Description

`dlgamma(x)` evaluates, at all the elements of `x` the logarithmic derivative of the gamma function which corresponds also to the derivative of the gammaln function :

$$\frac{1}{\Gamma(x)}\frac{d\Gamma(x)}{dx} = \frac{d}{dx}(\ln(\Gamma(x)))$$

`x` must be real. Also known as the psi function.

Examples

```
dlgamma(0.5)
```

See Also

gamma, gammaln

Authors

W. J. Cody (code from Netlib (specfun))

Name

erf — The error function.

```
y = erf(x)
```

Parameters

x
: real vector or matrix

y
: real vector or matrix (of same size than x)

Description

`erf` computes the error function:

$$y = \frac{2}{\sqrt{\pi}} \cdot \int_0^x e^{-t^2} \cdot dt$$

Examples

```
deff('y=f(t)','y=exp(-t^2)');
erf(0.5)-2/sqrt(%pi)*intg(0,0.5,f)
```

See Also

erfc, erfcx, calerf, cdfnor

Authors

W. J. Cody (code from Netlib (specfun))

Name

erfc — The complementary error function.

```
y = erfc(x)
```

Parameters

x
: real vector or matrix

y
: real vector or matrix (of same size than x)

Description

`erfc` computes the complementary error function:

$$y = \frac{2}{\sqrt{\pi}} \int_{x}^{+\infty} e^{-t^2} \cdot dt$$

$$y = 1 - \mathrm{erf}(x)$$

Examples

```
erf([0.5,0.2])+erfc([0.5,0.2])
```

See Also

erf, erfcx, calerf

Authors

W. J. Cody (code from Netlib (specfun))

Name

erfcx — scaled complementary error function.

```
y = erfcx(x)
```

Parameters

x
: real vector or matrix

y
: real vector or matrix (of same size than x)

Description

`erfcx` computes the scaled complementary error function:

$$y = \exp(x^2) \cdot \mathrm{erfc}(x)$$

$$y \longrightarrow \frac{1}{x\sqrt{\pi}} \quad \text{when} \quad x \to +\infty$$

See Also

erf, erfc, calerf

Authors

W. J. Cody (code from Netlib (specfun))

Name

erfinv — The inverse of the error function.

```
y = erfinv(x)
```

Parameters

x
: real vector or matrix

y
: real vector or matrix (of same size than x)

Description

`erfinv` computes the inverse of the error function `erf`. `x = erfinv(y)` satisfies `y = erf(x)`, -1 <= `y` < 1, -inf <= `x` <= inf.

Examples

```
x=linspace(-0.99,0.99,100);
y=erfinv(x);
plot2d(x,y)
norm(x-erf(y),'inf')
```

See Also

erfc , cdfnor

References

Milton Abramowitz and Irene A. Stegun, eds. Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. New York: Dover, 1972.

Name

gamma — The gamma function.

```
y = gamma(x)
```

Parameters

x
: real vector or matrix

y
: real vector or matrix with same size than x.

Description

`gamma(x)` evaluates the gamma function at all the elements of `x`. The gamma function is defined by :

$$\Gamma(x) = \int_0^{+\infty} t^{x-1} \cdot e^{-t} \cdot dt$$

and generalizes the factorial function for real numbers (`gamma(n+1) = n!`).

Examples

```
// simple examples
gamma(0.5)
gamma(6)-prod(1:5)

// the graph of the Gamma function on [a,b]
a = -3; b = 5;
x = linspace(a,b,40000)';
y = gamma(x);
xbasc()
c=xget("color")
xset("color",2)
plot2d(x, y, style=0, axesflag=5, rect=[a, -10, b, 10])
xset("color",c)
xtitle("The gamma function on ["+string(a)+","+string(b)+"]")
xselect()
```

See Also

gammaln, dlgamma

Authors

W. J. Cody and L. Stoltz (code from Netlib (specfun))

Name

gammaln — The logarithm of gamma function.

```
y = gammaln(x)
```

Parameters

x
: real vector

y
: real vector with same size.

Description

`gammaln(x)` evaluates the logarithm of gamma function at all the elements of `x`, avoiding underflow and overflow. `x` must be real.

Examples

```
gammaln(0.5)
```

See Also

gamma, dlgamma

Authors

W. J. Cody and L. Stoltz (code from Netlib (specfun))

Name

legendre — associated Legendre functions

```
y = legendre(n,m,x [,normflag])
```

Parameters

n
: non negative integer or vector of non negative integers regularly spaced with increment equal to 1

m
: non negative integer or vector of non negative integers regularly spaced with increment equal to 1

x
: real (row) vector (elements of `x` must be in the `(-1,1)` interval)

normflag
: (optional) scalar string

Description

When `n` and `m` are scalars, `legendre(n,m,x)` evaluates the associated Legendre function Pnm(x) at all the elements of `x`. The definition used is :

$$P_n^m(x) = (-1)^m \cdot (1-x^2)^{m/2} \cdot \frac{d^m}{dx^m} P_n(x)$$

where `Pn` is the Legendre polynomial of degree `n`. So `legendre(n,0,x)` evaluates the Legendre polynomial Pn(x) at all the elements of `x`.

When the normflag is equal to "norm" you get a normalized version (without the `(-1)^m` factor), precisely :

$$P_n^m(x,\mathrm{norm}) = \sqrt{\left(\frac{2n+1}{2} \cdot \frac{(n-m)!}{(n+m)!}\right)} \cdot (1-x^2)^{m/2} \cdot \frac{d^m}{dx^m} P_n(x)$$

which is useful to compute spherical harmonic functions (see Example 3):

For efficiency, one of the two first arguments may be a vector, for instance `legendre(n1:n2,0,x)` evaluates all the Legendre polynomials of degree *n1*, *n1+1*, ..., *n2* at the elements of `x` and `legendre(n,m1:m2,x)` evaluates all the Legendre associated functions Pnm for *m=m1*, *m1+1*, ..., *m2* at `x`.

Output format

In any case, the format of `y` is :

```
max(length(n),length(m)) x length(x)
```

and :

```
y(i,j) = P(n(i),m;x(j))   if n is a vector
y(i,j) = P(n,m(i);x(j))   if m is a vector
y(1,j) = P(n,m;x(j))      if both n and m are scalars
```

so that `x` is preferably a row vector but any `mx x nx` matrix is excepted and considered as an `1 x (mx * nx)` matrix, reshaped following the column order.

Examples

```
// example 1 : plot of the 6 first Legendre polynomials on (-1,1)
l = nearfloat("pred",1);
x = linspace(-l,l,200)';
y = legendre(0:5, 0,  x);
xbasc()
plot2d(x,y', leg="p0@p1@p2@p3@p4@p5@p6")
xtitle("the 6 th first Legendre polynomials")

// example 2 : plot of the associated Legendre functions of degree 5
l = nearfloat("pred",1);
x = linspace(-l,l,200)';
y = legendre(5, 0:5, x, "norm");
xbasc()
plot2d(x,y', leg="p5,0@p5,1@p5,2@p5,3@p5,4@p5,5")
xtitle("the (normalised) associated Legendre functions of degree 5")

// example 3 : define then plot a spherical harmonic
// 3-1 : define the function Ylm
function [y] = Y(l,m,theta,phi)
   // theta may be a scalar or a row vector
   // phi may be a scalar or a column vector
   if m >= 0 then
      y = (-1)^m/(sqrt(2*%pi))*exp(%i*m*phi)*legendre(l, m, cos(theta), "norm")
   else
      y = 1/(sqrt(2*%pi))*exp(%i*m*phi)*legendre(l, -m, cos(theta), "norm")
   end
endfunction

// 3.2 : define another useful function
function [x,y,z] = sph2cart(theta,phi,r)
   // theta row vector      1 x nt
   // phi   column vector  np x 1
   // r     scalar or np x nt matrix (r(i,j) the length at phi(i) theta(j))
   x = r.*(cos(phi)*sin(theta));
   y = r.*(sin(phi)*sin(theta));
   z = r.*(ones(phi)*cos(theta));
endfunction

// 3-3 plot Y31(theta,phi)
l = 3; m = 1;
theta = linspace(0.1,%pi-0.1,60);
phi = linspace(0,2*%pi,120)';
f = Y(l,m,theta,phi);
[x1,y1,z1] = sph2cart(theta,phi,abs(f));       [xf1,yf1,zf1] = nf3d(x1,y1,z1);
[x2,y2,z2] = sph2cart(theta,phi,abs(real(f))); [xf2,yf2,zf2] = nf3d(x2,y2,z2);
[x3,y3,z3] = sph2cart(theta,phi,abs(imag(f))); [xf3,yf3,zf3] = nf3d(x3,y3,z3);

xbasc()
subplot(1,3,1)
plot3d(xf1,yf1,zf1,flag=[2 4 4]); xtitle("|Y31(theta,phi)|")
```

```
subplot(1,3,2)
plot3d(xf2,yf2,zf2,flag=[2 4 4]); xtitle("|Real(Y31(theta,phi))|")
subplot(1,3,3)
plot3d(xf3,yf3,zf3,flag=[2 4 4]); xtitle("|Imag(Y31(theta,phi))|")
```

Authors

Smith, John M. (code dxlegf.f from Slatec)
B. Pincon (scilab interface)

Name

oldbesseli — Modified Bessel functions of the first kind (I sub alpha).
oldbesselj — Bessel functions of the first kind (J sub alpha).
oldbesselk — Modified Bessel functions of the second kind (K sub alpha).
oldbessely — Bessel functions of the second kind (Y sub alpha).

```
y = oldbesseli(alpha,x)
y = oldbesseli(alpha,x,ice)
y = oldbesselj(alpha,x)
y = oldbesselk(alpha,x)
y = oldbesselk(alpha,x,ice)
y = oldbessely(alpha,x)
```

Parameters

x
: real vector with non negative entries

alpha
: real vector with non negative entries regularly spaced with increment equal to one `alpha=alpha0+(n1:n2)`

ice
: integer flag, with default value 1

Description

These functions are obsolete, use besseli, besselj, besselk, bessely instead. Note however that the semantics of these two sets of functions are different.

`oldbesseli(alpha,x)` computes modified Bessel functions of the first kind (I sub alpha), for real, non-negative order `alpha` and real non negative argument `x`. `besseli(alpha,x,2)` computes `besseli(alpha,x).*exp(-x)`.

`oldbesselj(alpha,x)` computes Bessel functions of the first kind (J sub alpha), for real, non-negative order `alpha` and real non negative argument `x`.

`oldbesselk(alpha,x)` computes modified Bessel functions of the second kind (K sub alpha), for real, non-negative order `alpha` and real non negative argument `x`. `besselk(alpha,x,2)` computes `besselk(alpha,x).*exp(x)`.

`oldbessely(alpha,x)` computes Bessel functions of the second kind (Y sub alpha), for real, non-negative order `alpha` and real non negative argument `x`.

`alpha` and `x` may be vectors. The output is `m`-by-`n` with `m = size(x,'*')`, `n = size(alpha,'*')` whose `(i,j)` entry is `oldbessel?(alpha(j),x(i))`.

Remarks

Y_alpha and J_alpha Bessel functions are 2 independant solutions of the Bessel 's differential equation :

$$x^2 \cdot \frac{d^2y}{dx^2} + x \cdot \frac{dy}{dx} + (x^2 - \alpha^2) \cdot y = 0, \alpha \geq 0$$

K_alpha and I_alpha modified Bessel functions are 2 independant solutions of the modified Bessel 's differential equation :

$$x^2 \cdot \frac{d^2y}{dx^2} + x \cdot \frac{dy}{dx} - (x^2 + \alpha^2) \cdot y = 0, \alpha \geq 0$$

Examples

```
// example #1: display some I Bessel functions
x = linspace(0.01,10,5000)';
y = oldbesseli(0:4,x);
ys = oldbesseli(0:4,x,2);
xbasc()
subplot(2,1,1)
   plot2d(x,y, style=2:6, leg="I0@I1@I2@I3@I4", rect=[0,0,6,10])
   xtitle("Some modified Bessel functions of the first kind")
subplot(2,1,2)
   plot2d(x,ys, style=2:6, leg="I0s@I1s@I2s@I3s@I4s", rect=[0,0,6,1])
   xtitle("Some modified scaled Bessel functions of the first kind")

// example #2 : display some J Bessel functions
x = linspace(0,40,5000)';
y = besselj(0:4,x);
xbasc()
plot2d(x,y, style=2:6, leg="J0@J1@J2@J3@J4")
xtitle("Some Bessel functions of the first kind")

// example #3 : use the fact that J_(1/2)(x) = sqrt(2/(x pi)) sin(x)
//              to compare the algorithm of besselj(0.5,x) with
//              a more direct formula
x = linspace(0.1,40,5000)';
y1 = besselj(0.5, x);
y2 = sqrt(2 ./(%pi*x)).*sin(x);
er = abs((y1-y2)./y2);
ind = find(er > 0 & y2 ~= 0);
xbasc()
subplot(2,1,1)
   plot2d(x,y1,style=2)
   xtitle("besselj(0.5,x)")
subplot(2,1,2)
   plot2d(x(ind), er(ind), style=2, logflag="nl")
   xtitle("relative error between 2 formulae for besselj(0.5,x)")

// example #4: display some K Bessel functions
x = linspace(0.01,10,5000)';
y = besselk(0:4,x);
ys = besselk(0:4,x,1);
xbasc()
subplot(2,1,1)
   plot2d(x,y, style=0:4, leg="K0@K1@K2@K3@K4", rect=[0,0,6,10])
   xtitle("Some modified Bessel functions of the second kind")
subplot(2,1,2)
   plot2d(x,ys, style=0:4, leg="K0s@K1s@K2s@K3s@K4s", rect=[0,0,6,10])
   xtitle("Some modified scaled Bessel functions of the second kind")

// example #5: plot severals Y Bessel functions
x = linspace(0.1,40,5000)'; // Y Bessel functions are unbounded  for x -> 0+
y = bessely(0:4,x);
xbasc()
```

```
plot2d(x,y, style=0:4, leg="Y0@Y1@Y2@Y3@Y4", rect=[0,-1.5,40,0.6])
xtitle("Some Bessel functions of the second kind")
```

Authors

W. J. Cody, L. Stoltz (code from Netlib (specfun))

Spreadsheet

Name

excel2sci — reads ascii Excel files

```
M=excel2sci(fname [,sep])
```

Parameters

fname
: character string. The file path

sep
: character string. Excel separator used, default value is ","

M
: matrix of strings

Description

Given an ascii file created by Excel using "Text and comma" format `excel2sci(fname)` returns the corresponding Scilab matrix of strings. Use `excel2sci(fname,sep )` for an other choice of separator.

Note: You may eval all or part ot `M` using function `evstr`.

See Also

read , evstr

Name

readxls — reads an Excel file

```
sheets = readxls(file_path)
```

Parameters

file_path
: a character string: the path of the Excel file.

sheets
: an `mlist` of type `xls`, with one field named `sheets`

Description

Given an Excel file path this function returns an `mlist` data structure of type `xls`, with one field named `sheets`. The `sheets` field itself contains a list of sheet data structure.

`sheet=mlist(['xlssheet','name','text','value'],sheetname,Text,Value)` where `sheetname` is a character string containing the name of the sheet, `Text` is a matrix of string which contains the cell's strings and `Value` is a matrix of numberswhich contains the cell's values.

Warning only BIFF8 Excel files (last Excel file version) are handled

Examples

```
Sheets = readxls('SCI/modules/spreadsheet/demos/xls/t1.xls')
// some basic operations on Sheets
typeof(Sheets)
s1=Sheets(1) //get the first sheet
typeof(s1)
s1.value //get the first sheet value field
s1.text  //get the first sheet text field
s1(2,:)  //get the 2 row of the sheet
typeof(s1(2,:))

editvar s1
```

See Also

xls_open , xls_read

Authors

Pierrick Mode
: INRIA

Serge Steer
: INRIA

Used Functions

This function is based on the Scilab functions `xls_open` and `xls_read`

Name

xls_open — Open an Excel file for reading

```
[fd,SST,Sheetnames,Sheetpos] = xls_open(file_path)
```

Parameters

file_path
: a character string: the path of the Excel file.

fd
: a number, the logical unit on the Excel stream.

SST
: A vector of all character strings which appear in the Excel sheets.

Sheetnames
: a vector of strings: the sheet names.

Sheetpos
: a vector of numbers: the position of the beginning of sheets in the Excel stream.

Description

This function first analyzes the ole2 data structure associated with the given file to extract the Excel stream which is included in. After that the Excel stream is saved in the `TMDIR` directory and opened. The `fd` logical unit points to this temporary file. Then the first sheet in this stream is read to get the global information like number of sheets, sheet names `Sheetnames`, sheet adresses within the stream `Sheetpos` and the `SST` which contains all the strings used in the following sheets.

The `fd` and `Sheetpos` data have to be passed to xls_read to read the data sheets.

The readxls function can be used to read all an Excel file in one function with a single function call.

Warning only BIFF8 Excel files (last Excel file version (2003)) are handled

Examples

```
//Decode ole file, extract and open Excel stream
 [fd,SST,Sheetnames,Sheetpos] = xls_open('SCI/modules/spreadsheet/demos/xls/
 //Read first data sheet
 [Value,TextInd] = xls_read(fd,Sheetpos(1))
 //close the spreadsheet stream
 mclose(fd)
```

See Also

xls_read , readxls

Authors

Pierrick Mode
: INRIA

Serge Steer
: INRIA

Bibliography

This function is based on the Microsoft ole2 file documentation (http://chicago.sourceforge.net/devel/docs/ole/) and on Excel stream description from OpenOffice (http://sc.openoffice.org/spreadsheetfileformat.pdf).

Used Functions

The ripole-0.1.4 procedure (http://www.pldaniels.com/ripole) is used to extract the spreadsheet stream out of the ole file.

Name

xls_read — read a sheet in an Excel file

```
[Value,TextInd] = xls_read(fd,Sheetpos)
```

Parameters

fd
: a number, the logical unit on the Excel stream returned by xls_open.

Sheetpos
: a number: the position of the beginning of the sheet in the Excel stream. This position is one of those returned by xls_open.

Value
: a matrix of numbers, the numerical data found in the sheet. The cell without numerical data are represented by `NaN` values.

TextInd
: a matrix of indices with the same size as `Value`. The 0 indices indicates that no string exists in the correspondin Excel cell. a positive index `i` points to the string `SST(i)` where `SST` is given by xls_open.

Description

This function reads an Excel sheet given a logical unit on an Excel stream ant the position of the beginning of the sheet within this stream. It returns the numerical data and the strings contained by the Excel cells.

The readxls function can be used to read all an Excel file in one function with a single function call.

Warning only BIFF8 Excel files (last Excel file version) are handled

Examples

```
//Decode ole file, extract and open Excel stream
 [fd,SST,Sheetnames,Sheetpos] = xls_open('SCI/modules/spreadsheet/demos/xls/
 //Read first data sheet
 [Value,TextInd] = xls_read(fd,Sheetpos(1))
 //close the spreadsheet stream
 mclose(fd)
```

See Also

xls_open , readxls

Authors

Pierrick Mode
: INRIA

Serge Steer
: INRIA

Bibliography

This function is based on Excel stream description from OpenOffice (http://sc.openoffice.org/spreadsheetfileformat.pdf).

Used Functions

This function uses the xls.c file which can be found in a Scilab source version in the directory SCIDIR/modules/spreadsheet/src/c

Statistics

Name

cdfbet — cumulative distribution function Beta distribution

```
[P,Q]=cdfbet("PQ",X,Y,A,B)
[X,Y]=cdfbet("XY",A,B,P,Q)
[A]=cdfbet("A",B,P,Q,X,Y)
[B]=cdfbet("B",P,Q,X,Y,A)
```

Parameters

P,Q,X,Y,A,B
: five real vectors of the same size.

P,Q (Q=1-P)
: The integral from 0 to X of the beta distribution (Input range: [0, 1].)

Q
: 1-P

X,Y (Y=1-X)
: Upper limit of integration of beta density (Input range: [0,1], Search range: [0,1]) A,B : The two parameters of the beta density (input range: (0, +infinity), Search range: [1D-300,1D300])

Description

Calculates any one parameter of the beta distribution given values for the others (The beta density is proportional to `t^(A-1) * (1-t)^(B-1)`.

Cumulative distribution function (P) is calculated directly by code associated with the following reference.

DiDinato, A. R. and Morris, A. H. Algorithm 708: Significant Digit Computation of the Incomplete Beta Function Ratios. ACM Trans. Math. Softw. 18 (1993), 360-373.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdfbin — cumulative distribution function Binomial distribution

```
[P,Q]=cdfbin("PQ",S,Xn,Pr,Ompr)
[S]=cdfbin("S",Xn,Pr,Ompr,P,Q)
[Xn]=cdfbin("Xn",Pr,Ompr,P,Q,S)
[Pr,Ompr]=cdfbin("PrOmpr",P,Q,S,Xn)
```

Parameters

P,Q,S,Xn,Pr,Ompr
: six real vectors of the same size.

P,Q (Q=1-P)
: The cumulation from 0 to S of the binomial distribution. (Probablility of S or fewer successes in XN trials each with probability of success PR.) Input range: [0,1].

S
: The number of successes observed. Input range: [0, XN] Search range: [0, XN]

Xn
: The number of binomial trials. Input range: (0, +infinity). Search range: [1E-300, 1E300]

Pr,Ompr (Ompr=1-Pr)
: The probability of success in each binomial trial. Input range: [0,1]. Search range: [0,1]

Description

Calculates any one parameter of the binomial distribution given values for the others.

Formula 26.5.24 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the binomial distribution to the cumulative incomplete beta distribution.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdfchi — cumulative distribution function chi-square distribution

```
[P,Q]=cdfchi("PQ",X,Df)
[X]=cdfchi("X",Df,P,Q);
[Df]=cdfchi("Df",P,Q,X)
```

Parameters

P,Q,Xn,Df
: four real vectors of the same size.

P,Q (Q=1-P)
: The integral from 0 to X of the chi-square distribution. Input range: [0, 1].

X
: Upper limit of integration of the non-central chi-square distribution. Input range: [0, +infinity). Search range: [0,1E300]

Df
: Degrees of freedom of the chi-square distribution. Input range: (0, +infinity). Search range: [1E-300, 1E300]

Description

Calculates any one parameter of the chi-square distribution given values for the others.

Formula 26.4.19 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the chi-square distribution to the incomplete distribution.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdfchn — cumulative distribution function non-central chi-square distribution

```
[P,Q]=cdfchn("PQ",X,Df,Pnonc)
[X]=cdfchn("X",Df,Pnonc,P,Q);
[Df]=cdfchn("Df",Pnonc,P,Q,X)
[Pnonc]=cdfchn("Pnonc",P,Q,X,Df)
```

Parameters

P,Q,X,Df,Pnonc
: five real vectors of the same size.

P,Q (Q=1-P)
: The integral from 0 to X of the non-central chi-square distribution. Input range: [0, 1-1E-16).

X
: Upper limit of integration of the non-central chi-square distribution. Input range: [0, +infinity). Search range: [0,1E300]

Df
: Degrees of freedom of the non-central chi-square distribution. Input range: (0, +infinity). Search range: [1E-300, 1E300]

Pnonc
: Non-centrality parameter of the non-central chi-square distribution. Input range: [0, +infinity). Search range: [0,1E4]

Description

Calculates any one parameter of the non-central chi-square distribution given values for the others.

Formula 26.4.25 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to compute the cumulative distribution function.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

The computation time required for this routine is proportional to the noncentrality parameter (PNONC). Very large values of this parameter can consume immense computer resources. This is why the search range is bounded by 10,000.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdff — cumulative distribution function F distribution

```
[P,Q]=cdff("PQ",F,Dfn,Dfd)
[F]=cdff("F",Dfn,Dfd,P,Q);
[Dfn]=cdff("Dfn",Dfd,P,Q,F);
[Dfd]=cdff("Dfd",P,Q,F,Dfn)
```

Parameters

P,Q,F,Dfn,Dfd
: five real vectors of the same size.

P,Q (Q=1-P)
: The integral from 0 to F of the f-density. Input range: [0,1].

F
: Upper limit of integration of the f-density. Input range: [0, +infinity). Search range: [0,1E300]

Dfn
: Degrees of freedom of the numerator sum of squares. Input range: (0, +infinity). Search range: [1E-300, 1E300]

Dfd
: Degrees of freedom of the denominator sum of squares. Input range: (0, +infinity). Search range: [1E-300, 1E300]

Description

Calculates any one parameter of the F distribution given values for the others.

Formula 26.6.2 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the computation of the cumulative distribution function for the F variate to that of an incomplete beta.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

The value of the cumulative F distribution is not necessarily monotone in either degrees of freedom. There thus may be two values that provide a given CDF value. This routine assumes monotonicity and will find an arbitrary one of the two values.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdffnc — cumulative distribution function non-central f-distribution

```
[P,Q]=cdffnc("PQ",F,Dfn,Dfd,Pnonc)
[F]=cdffnc("F",Dfn,Dfd,Pnonc,P,Q);
[Dfn]=cdffnc("Dfn",Dfd,Pnonc,P,Q,F);
[Dfd]=cdffnc("Dfd",Pnonc,P,Q,F,Dfn)
[Pnonc]=cdffnc("Pnonc",P,Q,F,Dfn,Dfd);
```

Parameters

P,Q,F,Dfn,Dfd,Pnonc
: six real vectors of the same size.

P,Q (Q=1-P)
: The integral from 0 to F of the non-central f-density. Input range: [0,1-1E-16).

F
: Upper limit of integration of the non-central f-density. Input range: [0, +infinity). Search range: [0,1E300]

Dfn
: Degrees of freedom of the numerator sum of squares. Input range: (0, +infinity). Search range: [1E-300, 1E300]

Dfd
: Degrees of freedom of the denominator sum of squares. Must be in range: (0, +infinity). Input range: (0, +infinity). Search range: [1E-300, 1E300]

Pnonc
: The non-centrality parameter Input range: [0,infinity) Search range: [0,1E4]

Description

Calculates any one parameter of the Non-central F distribution given values for the others.

Formula 26.6.20 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to compute the cumulative distribution function.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

The computation time required for this routine is proportional to the noncentrality parameter (PNONC). Very large values of this parameter can consume immense computer resources. This is why the search range is bounded by 10,000.

The value of the cumulative noncentral F distribution is not necessarily monotone in either degrees of freedom. There thus may be two values that provide a given CDF value. This routine assumes monotonicity and will find an arbitrary one of the two values.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdfgam — cumulative distribution function gamma distribution

```
[P,Q]=cdfgam("PQ",X,Shape,Scale)
[X]=cdfgam("X",Shape,Scale,P,Q)
[Shape]=cdfgam("Shape",Scale,P,Q,X)
[Scale]=cdfgam("Scale",P,Q,X,Shape)
```

Parameters

P,Q,X,Shape,Scale
: five real vectors of the same size.

P,Q (Q=1-P)
: The integral from 0 to X of the gamma density. Input range: [0,1].

X
: The upper limit of integration of the gamma density. Input range: [0, +infinity). Search range: [0,1E300]

Shape
: The shape parameter of the gamma density. Input range: (0, +infinity). Search range: [1E-300,1E300]

Scale
: The scale parameter of the gamma density. Input range: (0, +infinity). Search range: (1E-300,1E300]

Description

Calculates any one parameter of the gamma distribution given values for the others.

Cumulative distribution function (P) is calculated directly by the code associated with:

DiDinato, A. R. and Morris, A. H. Computation of the incomplete gamma function ratios and their inverse. ACM Trans. Math. Softw. 12 (1986), 377-393.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

The gamma density is proportional to `T**(SHAPE - 1) * EXP(- SCALE * T)`

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdfnbn — cumulative distribution function negative binomial distribution

```
[P,Q]=cdfnbn("PQ",S,Xn,Pr,Ompr)
[S]=cdfnbn("S",Xn,Pr,Ompr,P,Q)
[Xn]=cdfnbn("Xn",Pr,Ompr,P,Q,S)
[Pr,Ompr]=cdfnbn("PrOmpr",P,Q,S,Xn)
```

Parameters

P,Q,S,Xn,Pr,Ompr
: six real vectors of the same size.

P,Q (Q=1-P)
: The cumulation from 0 to S of the negative binomial distribution. Input range: [0,1].

S
: The upper limit of cumulation of the binomial distribution. There are F or fewer failures before the XNth success. Input range: [0, +infinity). Search range: [0, 1E300]

Xn
: The number of successes. Input range: [0, +infinity). Search range: [0, 1E300]

Pr
: The probability of success in each binomial trial. Input range: [0,1]. Search range: [0,1].

Ompr
: 1-PR Input range: [0,1]. Search range: [0,1] PR + OMPR = 1.0

Description

Calculates any one parameter of the negative binomial distribution given values for the others.

The cumulative negative binomial distribution returns the probability that there will be F or fewer failures before the XNth success in binomial trials each of which has probability of success PR.

The individual term of the negative binomial is the probability of S failures before XN successes and is Choose `( S, XN+S-1 ) * PR^(XN) * (1-PR)^S`

Formula 26.5.26 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce calculation of the cumulative distribution function to that of an incomplete beta.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdfnor — cumulative distribution function normal distribution

```
[P,Q]=cdfnor("PQ",X,Mean,Std)
[X]=cdfnor("X",Mean,Std,P,Q)
[Mean]=cdfnor("Mean",Std,P,Q,X)
[Std]=cdfnor("Std",P,Q,X,Mean)
```

Parameters

P,Q,X,Mean,Std
: six real vectors of the same size.

P,Q (Q=1-P)
: The integral from -infinity to X of the normal density. Input range: (0,1].

X
: :Upper limit of integration of the normal-density. Input range: (-infinity, +infinity)

Mean
: The mean of the normal density. Input range: (-infinity, +infinity)

Sd
: Standard Deviation of the normal density. Input range: (0, +infinity).

Description

Calculates any one parameter of the normal distribution given values for the others.

A slightly modified version of ANORM from Cody, W.D. (1993). "ALGORITHM 715: SPECFUN - A Portabel FORTRAN Package of Special Function Routines and Test Drivers" acm Transactions on Mathematical Software. 19, 22-32. is used to calulate the cumulative standard normal distribution.

The rational functions from pages 90-95 of Kennedy and Gentle, Statistical Computing, Marcel Dekker, NY, 1980 are used as starting values to Newton's Iterations which compute the inverse standard normal. Therefore no searches are necessary for any parameter.

For X < -15, the asymptotic expansion for the normal is used as the starting value in finding the inverse standard normal. This is formula 26.2.12 of Abramowitz and Stegun.

The normal density is proportional to exp(- 0.5 * ((X - MEAN)/SD)**2)

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdfpoi — cumulative distribution function poisson distribution

```
[P,Q]=cdfpoi("PQ",S,Xlam)
[S]=cdfpoi("S",Xlam,P,Q)
[Xlam]=cdfpoi("Xlam",P,Q,S);
```

Parameters

P,Q,S,Xlam
: four real vectors of the same size.

P,Q (Q=1-P)
: The cumulation from 0 to S of the poisson density. Input range: [0,1].

S
: :Upper limit of cumulation of the Poisson. Input range: [0, +infinity). Search range: [0,1E300]

Xlam
: Mean of the Poisson distribution. Input range: [0, +infinity). Search range: [0,1E300]

Description

Calculates any one parameter of the Poisson distribution given values for the others.

Formula 26.4.21 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the computation of the cumulative distribution function to that of computing a chi-square, hence an incomplete gamma function.

Cumulative distribution function (P) is calculated directly. Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

From DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters (February, 1994) Barry W. Brown, James Lovato and Kathy Russell. The University of Texas.

Name

cdft — cumulative distribution function Student's T distribution

```
[P,Q]=cdft("PQ",T,Df)
[T]=cdft("T",Df,P,Q)
[Df]=cdft("Df",P,Q,T)
```

Parameters

P,Q,T,Df
: six real vectors of the same size.

P,Q (Q=1-P)
: The integral from -infinity to t of the t-density. Input range: (0,1].

T
: Upper limit of integration of the t-density. Input range: (-infinity, +infinity). Search range: [-1E150, 1E150]

DF:
: Degrees of freedom of the t-distribution. Input range: (0 , +infinity). Search range: [1e-300, 1E10]

Description

Calculates any one parameter of the T distribution given values for the others.

Formula 26.5.27 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the computation of the cumulative distribution function to that of an incomplete beta.

Computation of other parameters involve a seach for a value that produces the desired value of P. The search relies on the monotinicity of P with the other parameter.

Name

center — center

```
s=center(x)
s=center(x,'r') or s=center(x,1)
s=center(x,'c') or s=center(x,2)
```

Parameters

x: real or complex vector or matrix

Description

This function computes `s`, the centred version of the numerical matrix `x`. For a vector or a matrix `x`, `s=center(x)` returns in the `(i,j)` coefficient of the matrix `s` the value `(x(i,j)-xbar)`, where `xbar` is the mean of the values of the coefficients of `x`. `s=center(x,'r')` (or, equivalently, `s=center(x,1)`) is the rowwise center reduction of the values of `x`. It returns in the entry `s(i,j)` the value `(x(i,j)-xbarv(j))(j)` with `xbarv(j)` the mean of the values of the `j` column. `s=center(x,'c')` (or, equivalently, `s=center(x,2)`) is the columnwise center reduction of the values of `x`. It returns in the entry `s(i,j)` the value `(x(i,j)-xbarh(i)))` with `xbarh(i)` the mean of the values of the `i` row.

Examples

```
x=[0.2113249  0.0002211 0.6653811;
   0.7560439  0.3303271 0.6283918]
s=center(x)
s=center(x,'r')
s=center(x,'c')
```

See Also

wcenter

Authors

Carlos Klimann

Name

wcenter — center and weight

```
s=wcenter(x)
s=wcenter(x,'r') or s=wcenter(x,1)
s=wcenter(x,'c') or s=wcenter(x,2)
```

Parameters

x: real or complex vector or matrix

Description

This function computes `s`, the weigthed and centred version of the numerical matrix `x`.

For a vector or a matrix `x`, `s=wcenter(x)` returns in the `(i,j)` coefficient of the matrix `s` the value `(x(i,j)-xbar)/sigma`, where `xbar` is the mean of the values of the coefficients of `x` and `sigma` his standard deviation.

`s=wcenter(x,'r')` (or, equivalently, `s=wcenter(x,1)`) is the rowwise centre reduction of the values of `x`. It returns in the entry `s(i,j)` the value `(x(i,j)-xbarv(j))/sigmav(j)` with `xbarv(j)` the mean of the values of the `j` column and `sigmav(j)` the standard deviation of the `j` column of `x`.

`s=wcenter(x,'c')` (or, equivalently, `s=wcenter(x,2)`) is the columnwise centre reduction of the values of `x`. It returns in the entry `s(i,j)` the value `(x(i,j)-xbarh(i))/sigmah(i)` with `xbarh(i)` the mean of the values of the `i` row and `sigmah(i)` the standard deviation of the `i` row of `x`.

Examples

```
x=[0.2113249 0.0002211 0.6653811;
   0.7560439 0.3303271 0.6283918]
s=wcenter(x)
s=wcenter(x,'r')
s=wcenter(x,'c')
```

See Also

center

Authors

Carlos Klimann

Name

cmoment — central moments of all orders

```
mom=cmoment(x,ord)
mom=cmoment(x,ord,'r') or mom=cmoment(x,ord,1)
mom=cmoment(x,ord,'c') or mom=cmoment(x,ord,2)
```

Parameters

x
: real or complex vector or matrix

ord
: positive integer

Description

cmoment`(x,ord)` is the central moment or order ord of the elements of `x`. If a third argument of type string `'r'` (or `1`) or `'c'` (or `2`) is used, we get in the first case, a row vector `mom` such that `mom(j)` contains the central moment of order `ord` of the `j` column of `x`. `cmoment(x,ord,'c')` is used in the same way for the central moments in the rows.

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

Examples

```
x=[0.2113249 0.0002211 0.6653811;
   0.7560439 0.3303271 0.6283918]
mom=cmoment(x,3)
mom=cmoment(x,2,'r')
mom=cmoment(x,3,'c')
```

See Also

sum , median , st_deviation , mean , meanf , moment , nanmean , nanmeanf , stdev , stdevf , variance , variancef , nanstdev

Authors

Carlos Klimann

Name

correl — correlation of two variables

```
rho=correl(x,y,fre)
```

Parameters

x
: real or complex vector

y
: real or complex vector

fre
: matrix of type length(x) x length(y)

Description

`correl(x,y,fre)` computes the correlation of two variables x and y. fre is a matrix of dimensions length(x) x length(y). In fre the element of indices (i,j) corresponds to the value or number or frequences of x_i&y_j.

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

Examples

```
x=[2.5 7.5 12.5 17.5]
h=[0 1 2]
fre=[.03 .12 .07;.02 .13 .11;.01 .13 .14;.01 .09 .14]
rho=correl(x,h,fre)
```

See Also

covar

Authors

Carlos Klimann

Name

covar — covariance of two variables

```
s=covar(x,y,fre)
```

Parameters

x
: real or complex vector

y
: real or complex vector

fre
: matrix of type length(x) x length(y)

Description

`covar(x,y,fre)` computes the covariance of two variables x and y. fre is a matrix of dimensions length(x) x length(y). In fre the element of indices (i,j) corresponds to the value or number or frequences of x_i&y_j.

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

Examples

```
x=[10 20 30 40]
y=[10 20 30 40]
fre=[.20 .04 .01  0;
     .10 .36 .09  0;
       0 .05 .10  0;
       0   0   0 .05]
s=covar(x,y,fre)
```

Authors

Carlos Klimann

Name

ftest — Fischer ratio

```
f=ftest(samples)
[f,p]=ftest(samples)
```

Parameters

samples
: real or complex matrix of type nr X nc

Description

`f=ftest(samples)` computes the Fischer ratio of the nc samples whose values are in the columns of the matrix `samples`. Each one of these samples is composed of nr values. (The Fischer ratio is the ratio between nr times the variance of the means of samples and the mean of variances of each sample)

`[f,p]=ftest(samples)` gives in p the p-value of the computed Fischer ratio f.

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

Examples

```
samples=[46 55 54;
         53 54 50;
         49 58 51;
         50 61 51;
         46 52 49]
[f,p]=ftest(samples)
```

See Also

ftuneq

Authors

Carlos Klimann

Name

ftuneq — Fischer ratio for samples of unequal size.

```
f=ftuneq(sample1[,sample2[,sample3]...]])
[f,p]=ftuneq(sample1[,sample2[,sample3]...]])
```

Parameters

sample1, sample2, sample3,...
: real or complex matrix of any type

Description

This function computes the F ratio for samples of unequal size.

"The most efficient design is to make all samples the same size n. However when this is nor feasible, it still is possible to modify the ANOVA calculations." Note that the definition of xbarbar is no longer mean(xbar), but rather a weighted average with weights ni. Additionnally it gives (in p) the p-value of the computed Fischer ratio.

Given a number a of samples each of them composed of n_i (i from 1 to a) observations this fonction computes in f the Fischer ratio (it is the ratio between nr times the variance of the means of samples and the mean of the variances of each sample).

`f=ftest(samples)` computes the Fischer ratio of the nc samples whose values are in the columns of the matrix `samples`. Each one of these samples is composed of nr values. (The Fischer ratio is the ratio between nr times the variance of the means of samples and the mean of variances of each sample)

`[f,p]=ftest(samples)` gives in p the p-value of the computed Fischer ratio f.

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

Examples

```
samples=[46 55 54;53 54 50; 49 58 51;50 61 51;46 52 49]
[f,p]=ftest(samples)
```

See Also

ftuneq

Authors

Carlos Klimann

Name

geomean — geometric mean

```
gm=geomean(x)
gm=geomean(x,'r')(or, equivalently, gm=geomean(x,1))
gm=geomean(x,'c')(or, equivalently, gm=geomean(x,2))
```

Parameters

x
:real or complex vector or matrix

Description

This function computes the geometric mean of a vector or matrix `x`. For a vector or matrix `x`, `gm=geomean(x)` returns in scalar `gm` the geometric mean of all the entries of `x`.

`gm=geomean(x,'r')` (or, equivalently, `gm=gmean(x,1)`) returns in each entry of the row vector `gm` the geometric mean of each column of `x`.

`gm=geomean(x,'c')` (or, equivalently, `gm=gmean(x,2)`) returns in each entry of the column vector `gm` the geometric mean of each row of `x` .

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

harmean — harmonic mean

```
hm=harmean(x)
hm=harmean(x,'r')(or, equivalently, hm=harmean(x,1))
hm=harmean(x,'c')(or, equivalently, hm=harmean(x,2))
```

Parameters

x
: real or complex vector or matrix

Description

This function computes the harmonic mean of a vector or matrix `x`. For a vector or matrix `x`, `hm=harmean(x)` returns in scalar `hm` the harmonic mean of all the entries of `x`.

`hm=harmean(x,'r')` (or, equivalently, `hm=harmean(x,1)`) returns in each entry of the row vector `hm` the harmonic mean of each column of `x`.

`hm=harmean(x,'c')` (or, equivalently, `hm=harmean(x,2)`) returns in each entry of the column vector `hm` the harmonic mean of each row of `x` .

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

iqr — interquartile range

```
q=iqr(x)
q=iqr(x,'r') (or, equivalently, q=iqr(x,1))
q=iqr(x,'c') (or, equivalently, q=iqr(x,2))
```

Parameters

x
: real or complex vector or matrix

Description

This function computes the interquartile range IQR= upper quartile - lower quartile of a vector or a matrix `x` .

For a vector or a matrix `x` , `q=iqr(x)` returns in the scalar `q` the interquartile range of all the entries of `x`.

`q=iqr(x,'r')` (or, equivalently, `q=iqr(x,1)`) is the rowwise interquartile range. It returns in each entry of the row vector `q` the interquartile range of each column of `x`.

`q=iqr(x,'c')` (or, equivalently, `q=iqr(x,2)`) is the columnwise interquartile range. It returns in each entry of the column vector `q` the interquartile range of each row of `x`.

Authors

Carlos Klimann

Bibliography

Wonacott, T.H.Wonacott, R.J.; Introductory Statistics, J.Wiley-Sons, 1990.

Name

labostat — Statistical toolbox for Scilab

Contents

```
centre: centering variables
centrered: centering and reducing variables
cmoment: central moments of all orders
correl: correlation
covar: covariance
ftest: fischer test and his p-value
geomean: geometric mean
harmean: harmonic mean
iqr: interquartile range
mad: mean absolute deviation
meanf:  arithmetic mean of a vector or matrix with a table of frequences
median: 50th percentile of a sample
mn: arithmetic mean of a vector or matrix
moment: moments of all orders
msd:  mean squared  deviation
mvvacov : multivariable matrix of variance-covariance
nand2mean: estimate of the difference of means of two independent samples
nanmax:  maximum ignoring NaNs
nanmean: mean ignoring NaNs
nanmeanf:  mean with frequency table ignoring NaNs
nanmedian: 50th percentile of a sample ignoring NaNs
nanmin: minimum ignoring NANs
nanstdev:  standard deviation ignoring NaNs
nanstdevf: standard deviation with frequency table ignoring NaNs
nansum: sum ignoring NaNs
nfreq: frequency of the values of a sample
pca: principal component analysys
pctl: vector of  percentiles of a sample in decreasing order
perctl: vector of  percentiles of a sample in decreasing order
quart: quartils
stdev: standard deviation
stdevf: standard deviation with frequences
strange: distance between largest and smallest value
tabul: frequences of values
var: variance
varf: variance with frequence table
```

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, 5th edition, J.Wiley & Sons, 1990.

Saporta, Gilbert, Probabilites, Analyse des Donnees et Statistique, Editions Technip, Paris, 1990.

Name

mad — mean absolute deviation

```
s2=mad(x)
s2=mad(x,'r') or s2=mad(x,1)
s2=mad(x,'c') or s2=mad(x,2)
```

Parameters

x
: real or complex vector or matrix

Description

This function computes the mean absolute deviation of a real or complex vector or matrix `x`.

For a vector or matrix `x`, `s2=mad(x)` returns in scalar `s2` the mean absolute deviation of all the entries of `x`.

`s2=mad(x,'r')` (or, equivalently, `s2=mad(x,1)`) returns in each entry of the column vector `s2` the mean absolute deviation of each column of `x`.

`s2=mad(x,'c')` (or, equivalently, `s2=mad(x,2)`) returns in each entry of the column vector `s2` the mean absolute deviation of each row of `x`.

Bibliography

Reference: Wonacott T.H.& Wonacott R.J. .- Introductory Statistics, 5th edition, John Wiley, 1990.

Name

mean — mean (row mean, column mean) of vector/matrix entries

```
y=mean(x)
y=mean(x,'r')
y=mean(x,'c')
y=mean(x,'m')
```

Parameters

x
: real vector or matrix

y
: scalar or vector

Description

For a vector or a matrix `x`, `y=mean(x)` returns in the scalar `y` the mean of all the entries of `x`.

`y=mean(x,'r')` (or, equivalently, `y=mean(x,1)`) is the rowwise mean. It returns a row vector: `y(j)= mean(x(:,j))`

`y=mean(x,'c')` (or, equivalently, `y=mean(x,2)`) is the columnwise mean. It returns a column vector: `y(i)= mean(x(i,:))`

`y=mean(x,'m')` is the mean along the first non singleton dimension of x (for compatibility with Matlab).

Examples

```
A=[1,2,10;7,7.1,7.01];
mean(A)
mean(A,'r')
mean(A,'c')
A=matrix(1:12,[1,1,2,3,2]);
// in this case mean(A,'m') is equivalent to mean(A,3), the first non singleton
y=mean(A,'m')
```

See Also

sum , median , st_deviation

Name

meanf — weighted mean of a vector or a matrix

```
m=meanf(val,fre)
m=meanf(val,fre,'r') or m=meanf(val,fre,1)
m=meanf(val,fre,'c') or m=meanf(val,fre,2)
```

Parameters

?

Description

This function computes the mean of a vector or matrix `x`. For a vector or matrix `x`, `m=mn(x)` returns in scalar `m` the mean of all the entries of `x`. `m=mn(x,'r')` (or, equivalently, `m=mn(x,1)`) returns in each entry of the row vector `m` the mean of each column of `x`. `m=mn(x,'c')` (or, equivalently, `m=mn(x,2)`) returns in each entry of the column vector `m` the mean of each row of `x`.

Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.3303271 0.6283918]
m=meanf(x,rand(x))
m=meanf(x,[10 10 10;1 1 1],'r')
m=meanf(x,[10 10 10;1 1 1],'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

median — median (row median, column median,...) of vector/matrix/array entries

```
y=median(x)
y=median(x,'r')
y=median(x,'c')
y=median(x,'m')
y=median(x,dim)
```

Parameters

x
: real vector, matrix or an array

y
: scalar,vector, matrix or an array

dim
: positive integer

Description

For a vector or a matrix `x`, `y=median(x)` returns in the scalar `y` the median of all the entries of `x`.

`y=median(x,'r')` (or, equivalently, `y=median(x,1)`) is the median along the row index. It returns in each entry of the column vector `y` the median of each column of `x`.

`y=median(x,'c')` (or, equivalently, `y=median(x,2)`) is the median along the column index. It returns in each entry of the row vector `y` the median of each row of `x`.

`y=median(x,'m')` is the median along the first non singleton dimension of `x` (for compatibility with matlab).

`y=median(x,dim)` is the median along the dimension `dim` of `x` (for compatibility with matlab).

Examples

```
A=[1,2,10;7,7.1,7.01];
median(A)
median(A,'r')
median(A,'c')
A=matrix([-9 3 -8 6 74 39 12 -6 -89 23 65 34],[2,3,2]);
median(A,3)
median(A,'m')
```

See Also

sum , mean

Name

moment — non central moments of all orders

```
mom=moment(x,ord)
mom=moment(x,ord,'r') or mom=moment(x,ord,1)
mom=moment(x,ord,'c') or mom=moment(x,ord,2)
```

Parameters

x
: real or complex vector or matrix

ord
: positive integer

Description

moment`(x,ord)` is the non central moment or order ord of the elements of `x`.

If a third argument of type string `'r'` (or `1`) or `'c'` (or `2`) is used, we get in the first case, a row vector `mom` such that `mom(j)` contains the non central moment of order `ord` of the `j` column of `x`. `moment(x,ord,'c')` is used in the same way for the non central moments in the rows.

Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.3303271 0.6283918]
mom=moment(x,3)
mom=moment(x,2,'r')
mom=moment(x,3,'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

msd — mean squared deviation

```
y=msd(x)
y=msd(x,'r') or m=msd(x,1)
y=msd(x,'c') or m=msd(x,2)
```

Parameters

x
: real or complex vector or matrix

Description

This function computes the mean squared deviation of the values of a vector or matrix `x`.

For a vector or a matrix `x`, `y=msd(x)` returns in the scalar `y` the mean squared deviation of all the entries of `x`.

`y=msd(x,'r')` (or, equivalently, `y=msd(x,1)`) is the rowwise mean squared deviation. It returns in each entry of the row vector `y` the mean squared deviation of each column of `x` .

`y=msd(x,'c')` (or, equivalently, `m=msd(x,2)`) is the columnwise mean squared deviation. It returns in each entry of the column vector `y` the mean squared deviation of each row of `x`.

Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.3303271 0.6283918]
m=msd(x)
m=msd(x,'r')
m=msd(x,'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

mvvacov — computes variance-covariance matrix

```
v=mvvacov(x)
```

Parameters

x
: real or complex vector or matrix

Description

This function computes v, the matrix of variance-covariance of the "tableau" x (x is a numerical matrix nxp) who gives the values of p variables for n individuals: the (i,j) coefficient of v is v(i,j)=E(xi-xibar)(xj-xjbar), where E is the first moment of a variable, xi is the i-th variable and xibar the mean of the xi variable.

Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.4453586 0.6283918]
v=mvvacov(x)
```

Authors

Carlos Klimann

Bibliography

Saporta, Gilbert, Probabilites, Analyse des Donnees et Statistique, Editions Technip, Paris, 1990. Mardia, K.V., Kent, J.T. & Bibby, J.M., Multivariate Analysis, Academic Press, 1979.

Name

nancumsum — Thos function returns the cumulative sum of the values of a matrix

```
s = nancumsum(x,orient)
```

Parameters

x
: x is a numerical vector or matrix.

orient
: is an optional parameter. The possible values are '*', 1, 2, 'r' or 'c'.

s
: numerical scalar or vector. It contains the cumulative sum of the values of x, ignoring the NAN's.

Description

This function returns in scalar or vector s the cumulative sum of the values (ignoring the NANs) of a vector or matrix (real or complex) x.

This function for a vector or a matrix x, s=nancumsum(x) (or, equivalently s=nancumsum(x,'*') returns in scalar s the cumulative sum (ignoring the NANs) of all the entries of x taken columnwise.

s=nancumsum(x,'r') (or, equivalently, s=nancumsum(x,1)) returns in the cols(x) sized vector s the cumulative sum (ignoring the NANs) of the rows of x: s(:,i)=nancumsum(x(:,i))

s=nancumsum(x,'c') (or, equivalently, s=nancumsum(x,2)) returns in the rows(x) sized vector s the cumulative sum (ignoring NANs) of the columns of x: s(i,:)=nancumsum(x(i,:))

For the last two cases, if a row or column is in whole composed of NAN, the corresponding place of s will contain a NAN.

Examples

```
a=[1 2 3;4 5 6]
s=nancumsum(a)
s=nancumsum(a,'r')
s=nancumsum(a,'c')
```

See Also

nansum , cumsum

Authors

Carlos Klimann

Name

nand2mean — difference of the means of two independent samples

```
[dif]=nand2mean(sample1,sample2)
[dif]=nand2mean(sample1,sample2,conf)
```

Parameters

sample1
: real or complex vector or matrix

sample2
: real or complex vector or matrix

conf
: real scalar between 0 and 1

Description

This function computes an estimate (dif(1)) for the difference of the means of two independent samples (arrays sample1 and sample2) and gives the half amplitude of the range of variability of dif with an indicated confidence level (dif(2)). The choice of the normal or t fonctions as the probability fonction depends on the sizes of sample1 and sample2. We suppose that the underlying variances of both populations are equal. NAN values are not counted.

In Labostat, NAN values stand for missing values in tables.

In absence of the confidence parameter a confidence level of 95% is assumed.

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, 5th edition, J.Wiley & Sons, 1990.

Name

nanmax — max (ignoring Nan's)

```
[m,index]=nanmax(x)
[m,index]=nanmax(x,'r')
[m,index]=nanmax(x,'c')
```

Parameters

x
: real or complex vector or matrix

Description

This function gives for a real or a numerical matrix `x` his largest element `m` (but ignoring the NANs).

For `x`, a numerical vector or matrix, `m=nanmax(x)` returns in scalar `m` the largest element of `x` (ignoring the NANs). The form `[m,index] =nanmax(x,orient)` gives in addition of the value of the largest element of `x` (ignoring the NANs) in scalar `m`, the index of this element in `x`, as a 2-vector.

`m=nanmax(x,'r')` gives in the 1xsize(x,2) matrix `m` the largest elements (ignoring the NANs) of each column of `x`. If the form `[m,index]=nanmax(x,'r')` is used, the elements of the 1xsize(x,2) matrix index are the indexes of the largest elements (ignoring the NANs) of each column of x in the corresponding column.

m=nanmax(x,'c') gives in the size(x,2)x1 matrix m the largest elements (ignoring the NANs) of each row of x. If the form [m,index]=nanmax(x,'c') is used, the elements of the size(x,2)x1 matrix index are the indexes of the largest elements (ignoring the NANs) of each row of x in the corresponding row.

In Labostat, NAN values stand for missing values in tables.

Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
m=nanmax(x)
m=nanmax(x,'r')
m=nanmax(x,'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

nanmean — mean (ignoring Nan's)

```
m=nanmean(val)
m=nanmean(val,'r') (or m=nanmean(val,1))
m=nanmean(val,'c') (or m=nanmean(val,2))
```

Parameters

val
: real or complex vector or matrix

Description

This function returns in scalar `m` the mean of the values (ignoring the NANs) of a vector or matrix `val`.

For a vector or matrix `val`, `m=nanmean(val)` or `m=nanmean(val,'*')` returns in scalar `m` the mean of all the entries (ignoring the NANs) of `val`.

`m=nanmean(val,'r')` (or, equivalently, `m=nanmean(val,1)`) returns in each entry of the row vector `m` of type 1xsize(val,'c') the mean of each column of `val` (ignoring the NANs).

`m=nanmeanf(val,'c')` (or, equivalently, `m=nanmean(val,2)`) returns in each entry of the column vector `m` of type size(val,'c')x1 the mean of each row of `val` (ignoring the NANs).

In Labostat, NAN values stand for missing values in tables.

Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
m=nanmean(x)
m=nanmean(x,1)
m=nanmean(x,2)
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

nanmeanf — mean (ignoring Nan's) with a given frequency.

```
m=nanmean(val,fre)
m=nanmean(val,fre,'r') (or m=nanmean(val,fre,1))
m=nanmean(val,fre,'c') (or m=nanmean(val,fre,2))
```

Parameters

val
: real or complex vector or matrix

fre
: integer vector or matrix with same dimensions than val

Description

This function returns in scalar `m` the mean of the values (ignoring the NANs) of a vector or matrix `val`, each counted with a frequency signaled by the corresponding values of the integer vector or matrix `fre` with the same type of `val`.

For a vector or matrix `val`, `m=nanmeanf(val,fre)` or `m=nanmeanf(val,fre,'*')` returns in scalar `m` the mean of all the entries (ignoring the NANs) of `val`, each value counted with the multiplicity indicated by the corresponding value of `fre`.

`m=nanmeanf(val,fre,'r')` (or, equivalently, `m=nanmeanf(val,fre,1)`) returns in each entry of the row vector `m` of type 1xsize(val,'c') the mean of each column of val (ignoring the NANs), each value counted with the multiplicity indicated by the corresponding value of `fre`.

`m=nanmeanf(val,fre,'c')` (or, equivalently, `m=nanmeanf(val,fre,2)`) returns in each entry of the column vector m of type size(val,'c')x1 the mean of each row of val (ignoring the NANs), each value counted with the multiplicity indicated by the corresponding value of fre.

In Labostat, NAN values stand for missing values in tables.

Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
fre=[34 12 25;12 23 5]
m=nanmeanf(x,fre)
m=nanmeanf(x,fre,1)
m=nanmeanf(x,fre,2)
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

nanmedian — median of the values of a numerical vector or matrix

```
m=nanmedian(x)
m=nanmedian(x,'r') (or m=nanmedian(x,1))
m=nanmedian(x,'c') (or m=nanmedian(x,2))
```

Parameters

x
: real or complex vector or matrix

Description

For a vector or a matrix `x`, `[m]=nanmedian(x)` returns in the vector `m` the median of the values (ignoring the NANs) of vector `x`.

`[m]=nanmedian(x,'r')` (or, equivalently, `[m]=nanmedian(x,1)`) are the rowwise medians. It returns in each position of the row vector `m` the medians of data (ignoring the NANs) in the corresponding column of `x`.

`[m]=nanmedian(x,'c')` (or, equivalently, `[m]=nanmedian(x,2)`) are the columnwise madians. It returns in each position of the column vector `m` the medians of data (ignoring the NANs) in the corresponding row of `x`.

In Labostat, NAN values stand for missing values in tables.

Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
m=nanmedian(x)
m=nanmedian(x,1)
m=nanmedian(x,2)
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

nanmin — min (ignoring Nan's)

```
[m,index]=nanmin(x)
[m,index]=nanmin(x,'r')
[m,index]=nanmin(x,'c')
```

Parameters

x
: real or complex vector or matrix

Description

This function gives for a real or a numerical matrix `x` his largest element `m` (but ignoring the NANs).

For `x`, a numerical vector or matrix, `m=nanmin(x)` returns in scalar `m` the largest element of `x` (ignoring the NANs). The form `[m,index] =nanmin(x,orient)` gives in addition of the value of the largest element of `x` (ignoring the NANs) in scalar `m`, the index of this element in `x`, as a 2-vector.

`m=nanmin(x,'r')` gives in the 1xsize(x,2) matrix `m` the largest elements (ignoring the NANs) of each column of `x`. If the form `[m,index]=nanmin(x,'r')` is used, the elements of the 1xsize(x,2) matrix index are the indexes of the largest elements (ignoring the NANs) of each column of x in the corresponding column.

m=nanmin(x,'c') gives in the size(x,2)x1 matrix m the largest elements (ignoring the NANs) of each row of x. If the form [m,index]=nanmin(x,'c') is used, the elements of the size(x,2)x1 matrix index are the indexes of the largest elements (ignoring the NANs) of each row of x in the corresponding row.

In Labostat, NAN values stand for missing values in tables.

Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
m=nanmin(x)
m=nanmin(x,'r')
m=nanmin(x,'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

nanstdev — standard deviation (ignoring the NANs).

```
s=nanstdev(x)
s=nanstdev(x,'r') or m=nanstdev(x,1)
s=nanstdev(x,'c') or m=nanstdev(x,2)
```

Parameters

x
: real or complex vector or matrix

Description

This function computes the standard deviation of the values of a vector or matrix `x` (ignoring the NANs).

For a vector or a matrix `x`, `s=nanstdev(x)` returns in the scalar `s` the standard deviation of all the entries of `x` (ignoring the NANs).

`s=nanstdev(x,'r')` (or, equivalently, `s=nanstdev(x,1)`) is the rowwise standard deviation. It returns in each entry of the row vector `s` the standard deviation of each column of `x` (ignoring the NANs).

`s=nanstdev(x,'c')` (or, equivalently, `s=nanstdev(x,2)`) is the columnwise standard deviation. It returns in each entry of the column vector `s` the standard deviation of each row of `x` (ignoring the NANs).

In Labostat, NAN values stand for missing values in tables.

Examples

```
x=[0.2113249 0.0002211 0.6653811;
   0.7560439 %nan      0.6283918;
   0.3       0.2       0.5      ];
s=nanstdev(x)
s=nanstdev(x,'r')
s=nanstdev(x,'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

nansum — Sum of values ignoring NAN's

```
s = nansum(x,orient)
```

Parameters

x
: numerical vector or matrix.

orient
: nothing or '*'. 'r' or 1. 'c' or 2.

s
: Numerical scalar or vector containig the value of the adding operation.

Description

This function returns in s the sum of the values (ignoring the NAN's) of a numerical vector or matrix x.

For a vector or matrix x, s=nansum(x) (or s=nansum(x,'*')) returns in scalar s the sum of values of all entries (ignoring the NAN's) of a vector or matrix x.

s=nansum(x,'r')(or, equivalently, s=nansum(x,1)) returns in each entry of the row vector s of type 1xsize(x,'c') the sum of each column of x (ignoring the NANs).

s=nansum(x,'c')(or, equivalently, s=nansum(x,2)) returns in each entry of the column vector s of type size(x,'c')x1 the sum of each row of x (ignoring the NANs).

For the last two cases, if a row or column is in whole composed of NAN, the corresponding place of s will contain a NAN.

Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
m=nansum(x)
m=nansum(x,1)
m=nansum(x,2)
```

See Also

nancumsum , sum

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. and Wonacott, R.J.; Introductory Statistics, 5th edition, J.Wiley and Sons, 1990.

Name

nfreq — frequence of the values in a vector or matrix

```
m=nfreq(x)
```

Parameters

x
: real or complex vector or matrix

Description

Frequence of the values in a real or complex vector or a real or complex matrix `x`.

For a real or complex vector or a real or complex matrix `x`, `m=freq(x)` returns in the first column of the `size(x,'*')x2` matrix `m` the values of `x` and in the second column of this matrix the frequences of the corresponding values.

Note that the tabul function is more efficient, applies also to vector of strings and returns a sorted `m`.

Examples

```
x=[2 8 0 3 7 6 8 7 9 1]
m=nfreq(x)
```

See Also

tabul , dsearch , histplot

Authors

Carlos Klimann

Name

pca — Computes principal components analysis with standardized variables

```
[lambda,facpr,comprinc] = pca(x)
```

Parameters

x
: is a nxp (n individuals, p variables) real matrix. Note that `pca` center and normalize the columns of `x` to produce principal components analysis with standardized variables.

lambda
: is a p x 2 numerical matrix. In the first column we find the eigenvalues of V, where V is the correlation p x p matrix and in the second column are the ratios of the corresponding eigenvalue over the sum of eigenvalues.

facpr
: are the principal factors: eigenvectors of V. Each column is an eigenvector element of the dual of `R^p`.

comprinc
: are the principal components. Each column (c_i=Xu_i) of this n x n matrix is the M-orthogonal projection of individuals onto principal axis. Each one of this columns is a linear combination of the variables x1, ...,xp with maximum variance under condition `u'_i M^(-1) u_i=1`

Description

This function performs several computations known as "principal component analysis".

The idea behind this method is to represent in an approximative manner a cluster of n individuals in a smaller dimensional subspace. In order to do that, it projects the cluster onto a subspace. The choice of the k-dimensional projection subspace is made in such a way that the distances in the projection have a minimal deformation: we are looking for a k-dimensional subspace such that the squares of the distances in the projection is as big as possible (in fact in a projection, distances can only stretch). In other words, inertia of the projection onto the k dimensional subspace must be maximal.

Warning, the graphical part of the old version of `pca` as been removed. It can now be performed using the show_pca function.

Examples

```
a=rand(100,10,'n');
[lambda,facpr,comprinc] = pca(a);
show_pca(lambda,facpr)
```

See Also

show_pca , princomp

Authors

Carlos Klimann

Bibliography

Saporta, Gilbert, Probabilites, Analyse des Donnees et Statistique, Editions Technip, Paris, 1990.

Name

perctl — computation of percentils

```
p=perctl(x,y)
```

Parameters

x
: real or complex vector or matrix

y
: vector of positif values between 0 and 100.

Description

Compute the matrix p of percentils (in increasing order, column first) of the real vector or matrix x indicated by the entries of y, the values of entries of y must be positive integers between 0 and 100. p is a matrix whose type is length(y) x 2 and the content of its first column are the percentils values. The contents of its second column are the places of the computed percentiles in the input matrix x.

The minimum or maximum values in x are assigned to percentiles for percent values outside that range.

Examples

```
x=[6 7 0 7 10 4 2 2 7 1;
 6 0 5 5 5 2 0 6 8 10;
 8 6 4 3 5 9 8 3 4 7;
 1 3 2 7 6 1 1 4 8 2;
 6 3 5 1 6 5 9 9 5 5;
 1 6 4 4 5 4 0 8 1 8;
 7 1 3 7 8 0 2 8 10 8;
 3 6 1 9 8 5 5 3 2 1;
 5 7 6 2 10 8 7 4 0 8;
 10 3 3 4 8 6 9 4 8 3]
y=[10 20 30]
p=perctl(x,y)
```

Authors

Carlos Klimann

Bibliography

HYNDMAN,Rob J. and FAN Yanan, Sample Quantiles in Statistical Packages, The American Statistician, Nov.1996, Vol 50, No.4

Name

princomp — Principal components analysis

```
[facpr,comprinc,lambda,tsquare] = princomp(x,eco)
```

Parameters

x
: is a n-by-p (n individuals, p variables) real matrix.

eco
: a boolean, use to allow economy size singular value decomposition.

facpr
: A p-by-p matrix. It contains the principal factors: eigenvectors of the correlation matrix V.

comprinc
: a n-by-p matrix. It contains the principal components. Each column of this matrix is the M-orthogonal projection of individuals onto principal axis. Each one of this columns is a linear combination of the variables x1, ...,xp with maximum variance under condition `u'_i M^(-1) u_i=1`

lambda
: is a p column vector. It contains the eigenvalues of V, where V is the correlation matrix.

tsquare
: a n column vector. It contains the Hotelling's T^2 statistic for each data point.

Description

This function performs "principal component analysis" on the n-by-p data matrix x.

The idea behind this method is to represent in an approximative manner a cluster of n individuals in a smaller dimensional subspace. In order to do that, it projects the cluster onto a subspace. The choice of the k-dimensional projection subspace is made in such a way that the distances in the projection have a minimal deformation: we are looking for a k-dimensional subspace such that the squares of the distances in the projection is as big as possible (in fact in a projection, distances can only stretch). In other words, inertia of the projection onto the k dimensional subspace must be maximal.

To compute principal component analysis with standardized variables may use `princomp(wcenter(x,1))` or use the pca function.

Examples

```
a=rand(100,10,'n');
[facpr,comprinc,lambda,tsquare] = princomp(a);
```

See Also

wcenter , pca

Authors

Carlos Klimann

Bibliography

Saporta, Gilbert, Probabilites, Analyse des Donnees et Statistique, Editions Technip, Paris, 1990.

Name

quart — computation of quartiles

```
s=quart(x)
s=quart(x,'r') or m=quart(x,1)
s=quart(x,'c') or m=quart(x,2)
```

Parameters

x
: real or complex vector or matrix

Description

For a vector or a matrix x, [q]=quart(x,y) returns in the vector q the quartiles of x. [q]=quart(x,'r') (or, equivalently, [q]=quart(x,1)) are the rowwise percentiles. It returns in each column of the matrix q the quartiles of data in the corresponding column of x.

[q]=quart(x,'c') (or, equivalently, [q]=quart(x,2)) are the columnwise quartiles. It returns in each row of the matrix q the quartiles of data in the corresponding row of x.

Examples

```
x=[6 7 0 7 10 4 2 2 7 1;
   6 0 5 5 5 2 0 6 8 10;
   8 6 4 3 5 9 8 3 4 7;
   1 3 2 7 6 1 1 4 8 2;
   6 3 5 1 6 5 9 9 5 5;
   1 6 4 4 5 4 0 8 1 8;
   7 1 3 7 8 0 2 8 10 8;
   3 6 1 9 8 5 5 3 2 1;
   5 7 6 2 10 8 7 4 0 8;
   10 3 3 4 8 6 9 4 8 3]
q=quart(x)
q=quart(x,'r')
q=quart(x,'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

regress — regression coefficients of two variables

```
coefs=regress(x,y)
```

Parameters

x,y
: real or complex vector

Description

This function computes the regresion coefficients of two variables `x` and `y`, both numerical vectors of same number of elements `n`. `coefs=[a b]` be a 1x2 matrix such that `Y=a+bX` will be the equation of the ordinary least square approximation to our data.

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

Examples

```
x=[0.5608486 0.6623569 0.7263507 0.1985144 0.5442573 0.2320748 0.2312237]
y=[0.3616361 0.2922267 0.5664249 0.4826472 0.3321719 0.5935095 0.5015342]
coefs=regress(x,y)
```

See Also

covar

Authors

Carlos Klimann

Name

sample — Sampling with replacement

```
s = sample(n,X,orient)
```

Parameters

n
: positive integer (size of sample)

X
: matrix. Samples will be extracted from this matrix.

orient
: Optional parameter. Admissible values are 1, 2, 'r' or 'c'

s
: vector or matrix containing sample

Description

This function gives a vector (or matrix) nx1. It contains a random sample of n extractions, with replacement, from the matrix X.

s=sample(n,X) (or s=sample(n,X,'*')) returns a vector s whose values are a random sample of n values from X, extracted with replacement, from X .

s=sample(n,X,'r') (or, equivalently, s=sample(n,X,1)) returns a matrix of type size(X,'r')xn. It contains a random sample of n rows, extracted with replacement, from the rows of X.

s=sample(n,X,'c') (or, equivalently, s=sample(n,X,2)) returns a matrix of type nxsize(X,'c'). It contains a random sample of n columns, extracted with replacement from the columns of X.

Examples

```
X=['a' 'dd' 'arreu'; 'ber' 'car' 'zon']
s=sample(25,X)
s=sample(25,X,'r')
s=sample(25,X,'c')
```

See Also

samplef , samwr

Authors

Carlos Klimann

Name

samplef — sample with replacement from a population and frequences of his values.

```
s = samplef(n,X,f,orient)
```

Parameters

n
: positive integer (size of sample)

X
: matrix. Samples will be extracted from this matrix

f
: positive integer matrix with same type than X. It indicates frequences of corresponding values of X.

orient
: Optional parameter. Admissible values are 1, 2, 'r' or 'c'

s
: vector or matrix containing sample

Description

This function gives s, a vector of lenght n. It contains a sample of n extractions, with replacement, from the vector (or matrix) X, each element counted with the frequence given by the corresponding value in vector f.

s=samplef(n,X,f) (or s=samplef(n,X,f,'*')) returns a vector s whose values are a random sample of n values from X, each value with a probability to be sampled proportional to the corresponding value of f, extracted with replacement, from X. f must have same lenght than X.

s=samplef(n,X,f,'r') (or, equivalently, s=samplef(n,X,f,1)) returns a matrix of type size(X,'r')xn. It contains a random sample of n rows from X, each row with a probability to be sampled proportional to the corresponding value of f, extracted with replacement, from the rows of X. The lenght of f must be equal to the number of rows of X.

s=samplef(n,X,f,'c') (or, equivalently, s=samplef(n,X,f,2)) returns a matrix of type nxsize(X,'c'). It contains a random sample of n columns from X, each column with a probability to be sampled proportional to the corresponding value of f, extracted with replacement, from the columns of X. The lenght of f must be equal to the number of columns of X.

Examples

```
a=[3 7 9;22 4 2]
f1=[10 1 1 1 1 1]
f2=[1 ; 15]
f3=[10 1 1]
s=samplef(15,a,f1)
s=samplef(15,a,f2,'r')
s=samplef(15,a,f3,'c')
```

See Also

sample , samwr

Authors

Carlos Klimann

Name

samwr — Sampling without replacement

```
s = samwr(sizam,numsamp,X)
```

Parameters

sizam
: integer. Size of a sample. It must be less or equal than size of X.

numsamp
: integer. Number of samples to be extracted.

X
: column vector. It contains the population.

s
: matrix of type sizsam x numsamp. It contains numsamp random samples (the columns) each of sizam (size(X,'*')) extractions, without replacement, from the column vector X.

Description

Gives samples without replacement from a column vector.

Examples

```
a=[0.33 1.24 2.1 1.03]
s=samwr(4,12,a)
```

See Also

sample , samplef

Authors

Carlos Klimann

Name

show_pca — Visualization of principal components analysis results

```
show_pca(lambda,facpr,N)
```

Parameters

lambda
: is a p x 2 numerical matrix. In the first column we find the eigenvalues of V, where V is the correlation p x p matrix and in the second column are the ratios of the corresponding eigenvalue over the sum of eigenvalues.

facpr
: are the principal factors: eigenvectors of V. Each column is an eigenvector element of the dual of `R^p`.

N
: Is a 2x1 integer vector. Its coefficients point to the eigenvectors corresponding to the eigenvalues of the correlation matrix `p` by `p` ordered by decreasing values of eigenvalues. If `N`. is missing, we suppose `N=[1 2]`..

Description

This function visualize the pca results.

Examples

```
a=rand(100,10,'n');
[lambda,facpr,comprinc] = pca(a);
show_pca(lambda,facpr)
```

See Also

pca , princomp

Authors

Carlos Klimann

Bibliography

Saporta, Gilbert, Probabilites, Analyse des Donnees et Statistique, Editions Technip, Paris, 1990.

Name

st_deviation — standard deviation (row or column-wise) of vector/matrix entries
stdev — standard deviation (row or column-wise) of vector/matrix entries

```
y=st_deviation(x)
y=st_deviation(x,'r')
y=st_deviation(x,'c')
y=stdev(x)
y=stdev(x,'r')
y=stdev(x,'c')
```

Parameters

x
: real vector or matrix

y
: scalar or vector

Description

st_deviation computes the "sample" standard deviation, that is, it is normalized by N-1, where N is the sequence length.

For a vector or a matrix `x`, `y=st_deviation(x)` returns in the scalar `y` the standard deviation of all the entries of `x`.

`y=st_deviation(x,'r')` (or, equivalently, `y=st_deviation(x,1)`) is the rowwise standard deviation. It returns in each entry of the column vector `y` the standard deviation of each row of `x`.

`y=st_deviation(x,'c')` (or, equivalently, `y=st_deviation(x,2)`) is the columnwise st_deviation. It returns in each entry of the row vector `y` the standard deviation of each column of `x`.

Examples

```
A=[1,2,10;7,7.1,7.01];
st_deviation(A)
st_deviation(A,'r')
st_deviation(A,'c')
```

See Also

sum , median , mean , nanstdev , stdevf

Name

stdevf — standard deviation

```
s=stdevf(x,fre)
s=stdevf(x,fre,'r') or s=stdevf(x,fre,1)
s=stdevf(x,fre,'c') or s=stdevf(x,fre,2)
```

Parameters

x
: real or complex vector or matrix

Description

This function computes the standard deviation of the values of a vector or matrix `x`, each of them counted with a frequency given by the corresponding values of the integer vector or matrix `fre` who has the same type of `x`.

For a vector or matrix `x`, s=stdevf(x,fre) (or `s=stdevf(x,fre,'*')` returns in scalar `s` the standard deviation of all the entries of `x`, each value counted with the multiplicity indicated by the corresponding value of `fre`.

`s=stdevf(x,fre,'r')` (or, equivalently, `s=stdevf(x,fre,1)`) returns in each entry of the row vector `s` of type 1xsize(x,'c') the standard deviation of each column of `x`, each value counted with the multiplicity indicated by the corresponding value of `fre`.

`s=stdevf(x,fre,'c')` (or, equivalently, `s=stdevf(x,fre,2)`) returns in each entry of the column vector `s` of type size(x,'c')x1 the standard deviation of each row of `x`, each value counted with the multiplicity indicated by the corresponding value of `fre`.

Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.9546254 0.6283918]
fre=[1 2 3;3 4 3]
m=stdevf(x,fre)
m=stdevf(x,fre,'r')
m=stdevf(x,fre,'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

strange — range

```
[r]=strange(x)
[r]=strange(x,'r') (or, equivalently, [r]=strange(x,1))
[r]=strange(x,'c') (or, equivalently, [r]=strange(x,2))
```

Parameters

x
: real or complex vector or matrix

Description

The range is the distance between the largest and smaller value, [r]=strange(x) computes the range of vector or matrix x.

[r]=strange(x,'r') (or equivalently [r]=strange(x,1)) give a row vector with the range of each column.

[r]=strange(x,'c') (or equivalently [r]=strange(x,2)) give a column vector with the range of each row.

References

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, J.Wiley & Sons, 1990.

Authors

Carlos klimann

Name

tabul — frequency of values of a matrix or vector

```
[m]=tabul(X [,order])
```

Parameters

X
: vector or matrix (of real or complex numbers or strings)

order
: (optionnal) a character equal to "d" or "i" (default value "d")

m
: a 2 columns matrix (if `X` is a numerical vector or matrix) or a list with 2 members (if `X` is a string vector or matrix).

Description

This function computes the frequency of values of the components of a vector or matrix `X` of numbers or string characters :

if X is a numerical vector or matrix
: then `m` is a two column matrix who contains in the first column the distinct values of `X` and in the other column the number of occurrences of those values (m(i,2) is the number of occurrences of m(i,1)).

if X is a string vector or matrix
: then `m` is a list whose first member is a string (column) vector composed with the distinct values of `X` and the second member is a (column) vector whose components are the number of occurrences of those values (m(i)(2) is the number of occurrences of the string m(i)(1)).

The optional parameter `order` must be "d" or "i" (by default order="d") and gives the order (decreasing or increasing) the distinct values of X will be sorted.

Examples

```
// first example
X = [2 8 0 3 7 6 8 7 9 1 6 7 7 2 5 2 2 2 9 7]
m1 = tabul(X)
m2 = tabul(X, "i")

// second example
X = ["ba" "baba" "a" "A" "AA" "a" "aa" "aa" "aa" "A" "ba"]
m = tabul(X,"i")

// third example
n = 50000;
X = grand(n,1,"bin",70,0.5);
m = tabul(X,"i");
xbasc()
plot2d3(m(:,1), m(:,2)/n)
xtitle("empiral probabilities of B(70,0.5)")
```

```
// last example : computes the occurrences of words of the scilab license
text = read(SCI+"/license.txt",-1,1,"(A)"); // read the scilab license
bigstr = strcat(text," ");                  // put all the lines in a big strin
sep =  [" " "," "." ";" "*" ":" "-" """"];  // words separators
words = tokens(bigstr, sep);                // cut the big string into words
m = tabul(words);                           // computes occurrences of each wor
[occ , p] = sort(m(2));                     // sort by decreasing frequencies
results = [m(1)(p) string(occ)]             // display result
```

See Also

dsearch , histplot

Authors

Carlos Klimann (original author)
J.S. Giet and B. Pincon (new version)

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

thrownan — eliminates nan values

```
[nonan,numb]=thrownan(x)
```

Parameters

x
: real or complex vector or matrix

Description

This function returns in vector `nonan` the values (ignoring the NANs) of a vector or matrix `x` and in the corresponding places of vector `numb` the indexes of the value.

For a vector or matrix `x`, `[nonan,numb]=thrownan(x)` considers `x`, whatever his dimensions are, like a vector (columns first).

In Labostat, NAN values stand for missing values in tables.

Examples

```
x=[0.2113249 %nan 0.6653811;0.7560439 0.3303271 0.6283918]
[nonan numb]=thrownan(x)
```

Authors

Carlos Klimann

Name

trimmean — trimmed mean of a vector or a matrix

```
m=trimmean(x[,discard [,flag [,verbose]]])
```

Parameters

x
: real or complex vector or matrix

discard
: Optional real value between 0 and 100 representing the part of the data to discard. It discard is not in the [0,100] range, an error is generated. Default value for discard=50.

flag
: Optional string or real parameter which controls the behaviour when x is a matrix. Available values for flag are : "all", 1, 2, r or c (default is flag="all"). The two values flag=r and flag=1 are equivalent. The two values flag=c and flag=2 are equivalent.

verbose
: Optional integer. If set to 1, then enables verbose logging. Default is 0.

Description

A trimmed mean is calculated by discarding a certain percentage of the lowest and the highest scores and then computing the mean of the remaining scores. For example, a mean trimmed 50% is computed by discarding the lower and higher 25% of the scores and taking the mean of the remaining scores.

The median is the mean trimmed 100% and the arithmetic mean is the mean trimmed 0%.

A trimmed mean is obviously less susceptible to the effects of extreme scores than is the arithmetic mean. It is therefore less susceptible to sampling fluctuation than the mean for extremely skewed distributions. The efficiency of a statistic is the degree to which the statistic is stable from sample to sample. That is, the less subject to sampling fluctuation a statistic is, the more efficient it is. The efficiency of statistics is measured relative to the efficiency of other statistics and is therefore often called the relative efficiency. If statistic A has a smaller standard error than statistic B, then statistic A is more efficient than statistic B. The relative efficiency of two statistics may depend on the distribution involved. For instance, the mean is more efficient than the median for normal distributions but not for some extremely skewed distributions. The efficiency of a statistic can also be thought of as the precision of the estimate: The more efficient the statistic, the more precise the statistic is as an estimator of the parameter. The trimmed mean is less efficient than the mean for normal distributions.

For a vector or matrix `x`, `t=trimmean(x,discard)` returns in scalar `t` the mean of all the entries of `x`, after discarding `discard/2` highest values and `discard/2` lowest values.

`t=trimmean(x,discard,'r')` (or, equivalently, `t=trimmean(x,discard,1)`) returns in each entry of the row vector `t` the trimmed mean of each column of `x`.

`t=trimmean(x,discard,'c')` (or, equivalently, `t=trimmean(x,discard,2)`) returns in each entry of the column vector `t` the trimmed mean of each row of `x`.

This function computes the trimmed mean of a vector or matrix `x`.

For a vector or matrix `x`, `m=trimmean(x,discard)` returns in scalar `m` the trimmed mean of all the entries of `x`.

`m=trimmean(x,'r')` (or, equivalently, `m=trimmean(x,1)`) returns in each entry of the row vector `m` the trimmed mean of each column of `x`.

`m=trimmean(x,'c')` (or, equivalently, `m=trimmean(x,2)`) returns in each entry of the column vector `m` the trimmed mean of each row of `x`.

Example with x as vector

In the following example, one computes the trimmed mean of one data vector, with the default discard value equals to 50 and verbose logging. The data is made of 9 entries. The algorithms sorts the vector and keeps only indices from 3 to 7, skipping indices 1, 2, 8 and 9. The value 4000, which is much larger than the others is not taken into account. The computed trimmed mean is therefore 50.

```
data = [10, 20, 30, 40, 50, 60, 70, 80, 4000];
computed = trimmean(data,verbose=1);
```

Example with x as matrix

In the following example, one computes the trimmed mean of one data matrix. The chosen discard value is 50. The orientation is "r", which means that the data is sorted row by row. For each column of the matrix, one computes a trimmed mean. The trimmed mean is the line vector [25 25 25 25].

```
data = [
  10 10 10 10
  20 20 20 20
  30 30 30 30
  4000 4000 4000 4000
];
computed = trimmean(data,50,orien="r");
```

References

Luis Angel Garcia-Escudero and Alfonso Gordaliza, Robustness Properties of Means and Trimmed Means, JASA, Volume 94, Number 447, Sept 1999, pp956-969

Trimmed Mean, http://davidmlane.com/hyperstat/A11971.html

Authors

Carlos Klimann

Name

variance — variance of the values of a vector or matrix

```
s=variance(x[,orien[,w]])
s=variance(x,'r') or m=variance(x,1)
s=variance(x,'c') or m=variance(x,2)
```

Parameters

x
: real or complex vector or matrix

orien
: the orientation of the computation. Valid values or the orien parameter are 1, "r", 2 and "c".

w
: w : type of normalization to use. Valid values are 0 and 1. This depends on the number of columns of x (if orien = 1 is chosen), the number of rows (if orien = 2 is chosen). If w = 0, normalizes with m-1, provides the best unbiased estimator of the variance (this is the default). If w = 1, normalizes with m, this provides the second moment around the mean. If no orien option is given, the normalization is done with n * m - 1, where n * m is the total number of elements in the matrix.

Description

This function computes the variance of the values of a vector or matrix `x`.

For a vector or a matrix `x`, `s=variance(x)` returns in the scalar `s` the variance of all the entries of `x`.

`s=variance(x,'r')` (or, equivalently, `s=variance(x,1)`) is the rowwise variance. It returns in each entry of the row vector s the variance of each column of `x`. The generalized formulae is used, which manages complex values.

`s=variance(x,'c')` (or, equivalently, `s=variance(x,2)`) is the columnwise standard deviation. It returns in each entry of the column vector `s` the variance of each row of `x`. The generalized formulae is used, which manages complex values.

Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.4453586 0.6283918]
s=variance(x)
s=variance(x,'r')
s=variance(x,'c')
```

See Also

mtlb_var

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Name

variancef — standard deviation of the values of a vector or matrix

```
s=variancef(x,fre)
s=variancef(x,fre,'r') or s=variancef(x,fre,1)
s=variancef(x,fre,'c') or s=variancef(x,fre,2)
```

Parameters

x
: real or complex vector or matrix

Description

This function computes the variance of the values of a vector or matrix `x`, each of them counted with a frequency signaled by the corresponding values of the integer vector or matrix `fre` with the same type of `x`.

For a vector or matrix `x`, `s=variancef(x,fre)` (or `s=variancef(x,fre,'*')`) returns in scalar `s` the variance of all the entries of `x`, each value counted with the multiplicity indicated by the corresponding value of `fre`.

`s=variancef(x,fre,'r')`(or, equivalently, `s=variancef(x,fre,1)`) returns in each entry of the row vector `s` of type 1xsize(x,'c') the variance of each column of `x`, each value counted with the multiplicity indicated by the corresponding value of `fre`.

`s=variancef(x,fre,'c')`(or, equivalently, `s=variancef(x,fre,2)`) returns in each entry of the column vector `s` of type size(x,'c') x1 the variance of each row of `x`, each value counted with the multiplicity indicated by the corresponding value of `fre`.

Examples

```
x=[0.2113249 0.0002211 0.6653811;0.7560439 0.9546254 0.6283918]
fre=[1 2 3;3 4 3]
m=variancef(x,fre)
m=variancef(x,fre,'r')
m=variancef(x,fre,'c')
```

Authors

Carlos Klimann

Bibliography

Wonacott, T.H. & Wonacott, R.J.; Introductory Statistics, fifth edition, J.Wiley & Sons, 1990.

Strings

Name

ascii — string ascii conversions

```
a=ascii(txt)
txt=ascii(a)
```

Parameters

txt
: A character string or a matrix of character strings.

a
: A vector of integer ascii codes

Description

This function convert Scilab string to a vector of ascii code or vector of ascii code to Scilab strings.

If `txt` is a matrix of string, `ascii(txt)` is equivalent to `ascii(strcat(txt))`

Examples

```
ascii(["hello";"world"])
ascii("scilab")
ascii([115 99 105 108 97 98])
```

See Also

code2str, str2code

Name

blanks — Create string of blank characters

```
txt=blanks(n)
```

Parameters

txt
: A single character string

n
: number of blanks

Description

blanks(n) is a string of n blanks.

Examples

```
disp(['xxx' blanks(20) 'yyy'])
```

Name

code2str — returns character string associated with Scilab integer codes.

```
str=code2str(c)
```

Parameters

str
: A character string

c
: vector of character integer codes

Description

Returns character string associated with Scilab integer codes.str is such that c(i) is the Scilab integer code of part(str,i))

Examples

```
code2str([-28 12 18 21 10 11])
str2code('Scilab')'
```

See Also

str2code, ascii

Name

convstr — case conversion

```
[y]=convstr(str, [flag])
```

Parameters

str, y
: A matrix of character strings

flag
: A character option with possible values

  'u'
  : for upper

  'l'
  : for lower

Description

converts the matrix of strings `str-matrix` into lower case (for `"l"` ;default value) or upper case (for `"u"`).

Examples

```
A=['this','is';'my','matrix'];
convstr(A,'u')
```

Name

emptystr — zero length string

```
s=emptystr()
s=emptystr(a)
s=emptystr(m,n)
```

Parameters

a
: Any type of matrix

s
: A matrix of character strings

m,n
: Integers

Description

Returns a matrix of zero length character strings

With no input argument returns a zero length character string.

With a matrix for input argument returns a zero length character strings matrix of the same size.

With two integer arguments returns a mxn zero length character strings matrix

Examples

```
x=emptystr();for k=1:10, x=x+','+string(k);end
```

See Also

part , length , string

Name

grep — find matches of a string in a vector of strings

```
row=grep(haystack,needle )
[row,which]=grep(haystack,needle )
row=grep(haystack,needle ,[flag])
[row,which]=grep(haystack,needle ,[flag])
```

Parameters

haystack
: A Row vector of character strings.

needle
: A character string or a row vector of character strings . The string(s) to search in `haystack.`

row
: vector of indices: row where a match has been found or an empty matrix if no match found.

which
: vector of indices: index of needle string found or an empty matrix if no match found.

flag
: Character ("r" for regular expression)

Description

Foreach entry of haystack , grep searches if at least a string in needle matches a substring. haystack entries index where at least a match has been found are returned in the row argument. while optionnal which argument gives the index of first string of needle found. When using the third parameters "r", the needle should be a string of regular expression. And then grep is going to match it with haystack according to the regular express rules.

Examples

```


txt=['find matches of a string in a vector of strings'
     'search position of a character string in an other string'
     'Compare Strings'];

grep(txt,'strings')
grep(txt,['strings' 'Strings'])

[r,w]=grep(txt,['strings' 'Strings'])

str = ["hat";"cat";"hhat";"chat";"hcat";"ccchat";"at";"dog"]

grep(str,'/[hc]+at/','r')
grep(str,'/[hc]?at/','r')
grep(str,'/cat|dog/','r')


```

See Also

strindex

Name

isalphanum — check that characters of a string are alphanumerics

```
res = isalphanum(str)
```

Parameters

str
: A character string.

res
: A boolean matrix.

Description

`res = isalphanum(str)` returns an array the same size as str containing logical %t (true) where the elements of str are alphanumerics and logical %f (false) where they are not.

Examples

```
s = 'A1,B2,C3';
isalphanum(s)
```

See Also

isletter , isdigit

Name

isascii — tests if character is a 7-bit US-ASCII character

```
res = isascii(str)
```

Parameters

str
: A character string.

res
: A Boolean matrix.

Description

`res = isascii(str)` returns TRUE (%T) if c is a 7-bit US-ASCII character code between 0 and octal 0177 inclusive. otherwise returns FALSE (%F)

Examples

```
isascii(code2str(300))
isascii(code2str(-11))
letters = [115.    99.    105.    108.    97.    98.]
isascii(letters)
ascii(letters)
isascii('scilab')
```

See Also

isalphanum

Name

isdigit — check that characters of a string are digits between 0 and 9

```
res = isdigit(str)
```

Parameters

str
: A character string.

res
: A boolean matrix.

Description

`res = isdigit(str)` returns an array the same size as str containing logical %T (TRUE) where the elements of str are digits and logical %F (FALSE) where they are not.

Examples

```
s = 'A1,B2,C3';
isdigit(s)
```

See Also

isalphanum , isletter

Name

isletter — check that characters of a string are alphabetics letters

```
res = isletter(str)
```

Parameters

str
: A character string.

res
: A boolean matrix.

Description

`res = isletter(str)` returns an array the same size as str containing logical %t (true) where the elements of str are letters of the alphabet and logical %f (false) where they are not.

Examples

```
s = 'A1,B2,C3';
isletter(s)
```

See Also

isalphanum , isdigit

Name

isnum — tests if a string represents a number

```
res = isnum(str)
```

Parameters

str
: A character string or a matrix of character strings.

res
: A boolean matrix.

Description

`res = isnum(str)` returns %T if str represents a number

Examples

```
isnum(['1'     , ..
       '-1.23' , ..
       '+1e+23', ..
       '1d+23' , ..
       '%pi'])
```

See Also

isletter , isdigit , isalphanum

Authors

P.M

Name

justify — Justify character array.

```
Tj=justify(T,opt)
```

Parameters

T
: A matrix of character string.

Tj
: A matrix of character string. the justified result

opt
: A character option with possible values

  'r'
  : or 'right' for right justification

  'l'
  : or 'left' for left justification

  'c'
  : or 'center' for centering justification

Description

`justify` justify the column of a matrix of string accdording to the given option.

Examples

```
t=['1234','x','adfdfgdfghfgj'
   '1','354556','dgf'
   'sdfgd','','sdfsf'];

justify(t,'l')
justify(t,'c')
justify(t,'r')
```

See Also

length , part

Name

length — length of object

```
n=length(M)
```

Parameters

M
: matrix (usual or polynomial or character string) or list

n
: integer or integer matrix

Description

For usual or polynomial matrix `n` is the integer equal to number of rows times number of columns of `M`. (Also valid for `M` a boolean matrix)

For matrices made of character strings (and in particular for a character string) `length` returns in `n` the length of entries of the matrix of character strings `M`.

The length of a list is the number of elements in the list (also given by `size`).

`length('123')` is `3`. `length([1,2;3,4])` is `4`.

WARNING : length of a sparse matrix returns the max of dimensions and not the product of the dimensions. (example : length(sparse(eye(12,2))) returns max(12,2) and not 24)

please use size(...,'*') with sparse matrix.

Examples

```
length([123 ; 456 ])
length(['hello world',SCI])
```

See Also

size

Name

part — extraction of strings

```
[strings_out] = part(strings_in, v)
```

Parameters

strings_in, strings_out
: Matrix of character strings.

v
: Integer row vector.

Description

Let `s[k]` stands for the `k` character of string `s` (or the white space character if `k >length(s)`).

`part` returns `strings_out`, a matrix of character strings, such that `strings_out(i,j)` is the string `"s[v(1)]...s[v(n)]"` (`s=strings_in(i,j)`).

Examples

```
// returns characters position 8 to 11
part("How to use ""part"" ?",8:11)

// returns characters position 2 to 4 for each element
// no characters replaced by ''
c = part(['a','abc','abcd'],2:4)

// returns character position 1 for each element and add characters position 4 t
c = part(['abcdefg','hijklmn','opqrstu'],[1,4:7]);

// returns character 4 for each element, add characters position 1 to 7 and add
c = part(['abcdefg','hijklmn','opqrstu'],[4,1:7,4]);

// returns character position 1,add again character position 1 and character po
c=part(['a','abc','abcd'],[1,1,2])

// a a a
part(['a','abc','abcd'],[1])

// aa aa aa
part(['a','abc','abcd'],[1,1])

// aa aab aab
part(['a','abc','abcd'],[1,1,2])
```

See Also

string, length

Name

regexp — find a substring that matches the regular expression string

```
[start]=regexp(input,pattern,[flag])
[start,end,match]=regexp(input,pattern,[flag])
[start,end]=regexp(input,pattern,[flag])
[start,end,match]=regexp(input,pattern,[flag])
```

Parameters

input
: a string.

pattern
: a character string (under the rules of regular expression)

start
: the starting index of each substring of str that matches the regular expression string pattern

end
: the ending index of each substring of str that matches the regular expression string pattern

match
: the text of each substring of str that matches `pattern`.

[flag]
: 'o' for matching the pattern once .

Description

The rules of regular expression are similar to perl language. For a quick start , see http://perldoc.perl.org/perlrequick.html. For a more in-depth tutorial on , see http://perldoc.perl.org/perlretut.html and for the reference page, see http://perldoc.perl.org/perlre.html

A difference with Perl is that matching a position but no character (for example, with /^/ or /(?=o)/) is a successful match in Perl but not in Scilab.

Examples

```
regexp('xabyabbbz','/ab*/','o')
regexp('a!','/((((((((((a))))))))))\041/')
regexp('ABCC','/^abc$/i')
regexp('ABC','/ab|cd/i')
[a b c]=regexp('XABYABBBZ','/ab*/i')
```

See Also

strindex

Name

sci2exp — converts an expression to a string

```
t=sci2exp(a [,nam] [,lmax])
```

Parameters

a
: a scilab expression, may be

  - constant,
  - polynomial
  - string matrix
  - list
  - boolean matrix

nam
: character string

t
: vector of string, contains the expression or the affectation instruction

lmax
: integer, contains the maximum line length. default value is 90, `lmax=0` indicate no line length control a single string is returned

Description

`sci2exp` converts expression to an instruction string if `nam` is given or to an expression string.

Examples

```
a=[1 2;3 4]
sci2exp(a,'aa')
sci2exp(a,'aa',0)
sci2exp(ssrand(2,2,2))
sci2exp(poly([1 0 3 4],'s'),'fi')
```

Name

str2code — return scilab integer codes associated with a character string

```
c=str2code(str)
```

Parameters

str
: A character string.

c
: A vector of character integer codes

Description

Return c such that c(i) is the scilab integer code of part(str,i))

Examples

```
str2code('Scilab')'
code2str([-28 12 18 21 10 11])
```

See Also

code2str, ascii

Name

strcat — concatenate character strings

```
txt=strcat(vector_of_strings [,string_added])
txt=strcat(vector_of_strings [,string_added],["flag"])
```

Parameters

vector_of_strings
: vector of strings

string_added
: string added, default value is the emptystr `""`

txt
: string

"flag"
: string ("r" for return a column matrix, "c" for return a row matrix)

Description

`txt=strcat(vector_of_strings)` concatenates character strings : `txt=vector_of_strings(1)+...+vector_of_strings(n)`

`txt=strcat(vector_of_strings,string_added)` returns `txt=vector_of_strings(1)+string_added+...+string_added +vector_of_strings(n)`.

The plus symbol does the same: `"a"+"b"` is the same as `strcat(["a","b"]).`

`If size of vector_of_strings is one, it returns txt=vector_of_strings(1);`

`strcat('A','B') returns 'A' and not 'AB' as strcat(['A','B'])`

Examples

```
strcat(string(1:10),',')
strcat(["a","b"])
strcat(["a","b"],'|')
strcat('A')
strcat('A','B')
strcat(['A','B'])
strcat(['A','B'],'')
```

See Also

string, strings

Name

strchr — Find the first occurrence of a character in a string

```
res = strchr(haystack,char)
```

Parameters

haystack
: A character string or matrix of character strings

char
: a character.

res
: A character string or matrix of character strings

Description

`res = strchr(haystack,char)` Returns the first occurrence of character in the string str.

num must have same dimensions than haystack or only one char.

Examples

```
strchr('This is a sample string','s')
strchr(['This is a sample string','in scilab'],'s')
strchr(['This is a sample string','in scilab'],['s','a'])
```

See Also

strrchr , strstr

Name

strcmp — compare character strings

```
res = strcmp(string_one,string_two,[,'i'])
```

Parameters

string_one
: A character string or matrix of character strings

string_two
: A character string or matrix of character strings

'i'
: parameter to do stricmp (case independent), default value is 's'

res
: matrix.

Description

`res = strcmp(string_one,string_two)` returns an integral value indicating the relationship between the strings.

A value greater than zero indicates that the first character that does not match has a greater value in `string_one` than in `string_two`

And a value less than zero indicates the opposite.

Examples

```
TXT1 = ['scilab','SciLab';'Strcmp','STRcmp'];
TXT2 = ['ScIlAb','sciLab';'sTrCmP','StrCMP'];
strcmp(TXT1,TXT2)
strcmp(TXT1,'scilab')
strcmp(TXT1,'SciLab')
strcmp(TXT1,TXT2,'i')
```

See Also

strcat, strcmpi

Name

strcmpi — compare character strings (case independent)

```
res = strcmpi(string_one,string_two)
```

Parameters

string_one
: A character string or matrix of character strings

string_two
: A character string or matrix of character strings

res
: matrix.

Description

`res = strcmpi(string_one,string_two)` returns an integral value indicating the relationship between the strings.

A value greater than zero indicates that the first character that does not match has a greater value in `string_one` than in `string_two`

And a value less than zero indicates the opposite.

Examples

```
TXT1 = ['scilab','SciLab';'Strcmp','STRcmp'];
TXT2 = ['ScIlAb','sciLab';'sTrCmP','StrCMP'];
strcmpi(TXT1,TXT2)
strcmpi(TXT1,'scilab')
```

See Also

strcat , strcmp

Name

strcspn — Get span until character in string

```
res = strcspn(string_one,string_two)
```

Parameters

string_one
: A character string or matrix of character strings

string_two
: A character string or matrix of character strings

res
: matrix.

Description

`res = strcspn(string_one,string_two)` Scans string_one for the first occurrence of any of the characters that are part of string_two, returning the number of characters of string_one read before this first occurrence.

string_one must have same dimensions than string_two or string_one must be a string.

Examples

```
strcspn("fcba73","1234567890")
strcspn(["fcba73","f7cba73"],"1234567890")
strcspn(["fcba73","f7cba73"],["312","34567890"])
```

See Also

strspn

Name

strindex — search position of a character string in an other string.

```
ind=strindex(haystack,needle,[flag])
[ind,which]=strindex(haystack,needle,[flag])
```

Parameters

haystack
: A character string. The string where to search occurrences of `needle`

needle
: A character string or character string vector . The string(s) to search in `haystack`

ind
: vector of indexes

which
: vector of indexes

flag
: string("r" for regular expression)

Description

`strindex` searches indexes where `needle (i)` is found in `haystack`

for each `k` it exist a `i` shuch that `part(haystack,ind(k)+(0:length(needle(i))-1))` is the same string than `needle(i)`. If `which` argument is required it contains these `i`. When using the third parameters "r", the needle should be a string of regular expression. And then strindex is going to match it with haystack according to the regular express rules.

`strindex` without regular expression argument is based on Knuth-Morris-Pratt algoritm.

This algorithm is more powerful than that used in scilab 4.x. In some special case, result can be different.

example:

// scilab 5.x

-->[k,w]=strindex('aab',['a','ab'])

w = 1. 1. 2. k = 1. 2. 2.

// scilab 4.x

-->[k,w]=strindex('aab',['a','ab'])

w = 1. 1. k = 1. 2.

The rules of regular expression are similar to perl language. For a quick start , see http://perldoc.perl.org/perlrequick.html. For a more in-depth tutorial on , see http://perldoc.perl.org/perlretut.html and for the reference page, see http://perldoc.perl.org/perlre.html

Examples

```
```

```
k=strindex('SCI/demos/scicos','/')
k=strindex('SCI/demos/scicos','SCI/')
k=strindex('SCI/demos/scicos','!')
k=strindex('aaaaa','aa')
k=strindex('SCI/demos/scicos',['SCI','sci'])
[k,w]=strindex('1+3*abc/2.33',['+','-','*','/'])
k=strindex('2' ,'/2(]*)?$\1/' ,'r')
```

See Also

string, strings, regexp, strsubst

Name

string — conversion to string

```
string(x)
[out,in,text]=string(x)
```

Parameters

x
: real matrix or function

Description

converts a matrix into a matrix of strings.

If `x` is a function `[out,in,text]=string(x)` returns three vectors strings : `out` is the vector of output variables, `in` is the vector of input variables, and `text` is the (column) vector of the source code of the function.

If `x` is a `lib` variable, text is a character string column vector. The first element contains the path of library file and the other the name of functions it defines.

Character strings are defined as `'string'` (between quotes) or `"string"` (between doublequotes); matrices of strings are defined as usual constant matrices.

Concatenation of strings is made by the + operation.

Examples

```
string(rand(2,2))
deff('y=mymacro(x)','y=x+1')
[out,in,text]=string(mymacro)
x=123.356; 'Result is '+string(x)
```

See Also

part , length , quote , evstr , execstr , strsubst , strcat , strindex , sci2exp

Name

strings — Scilab Object, character strings

Description

Strings are defined as `'string'` (between quotes) or `"string"` (between doublequotes); matrices of strings are defined as usual constant matrices.

Concatenation of two strings is made by a `+` : `string1+string2`.

Examples

```
['this','is'; 'a 2x2','matrix']
"matrix"=="mat"+"rix"
```

See Also

part , length , strcat

Name

stripblanks — strips leading and trailing blanks (and tabs) of strings

```
txt=stripblanks(txt[,tabs])
```

Parameters

txt
: A character string or matrix of character strings

tabs
: if TRUE then tabs are also stripped (default value is FALSE)

Description

stripblanks strips leading and trailing blanks (and tabs) of strings

Examples

```
a='  123   ';
'!'+a+'!'
'!'+stripblanks(a)+'!'
a=['  123   ',' xyz']
strcat(stripblanks(a))
```

Name

strncmp — Copy characters from strings

```
res = strncpy(str1,num)
```

Parameters

str1
: A character string or matrix of character strings

num
: matrix Maximum number of characters to be copied from source

res
: A character string or matrix of character strings

Description

`res = strncpy(str1,num)` Copies the first num characters of source to destination.

num must have same dimensions than str1 or str2 must be a number.

Examples

```
strncpy('scilab',3)
strncpy(['scilab','SciLab';'strncpy','strstr'],3)
strncpy(['scilab','SciLab';'strncpy','strstr'],[1,2;3,4])
```

See Also

strcat , strcmp

Name

strrchr — Find the last occurrence of a character in a string

```
res = strrchr(str1,char)
```

Parameters

str1
: A character string or matrix of character strings

char
: a character.

res
: A character string or matrix of character strings

Description

`res = strrchr(str1,char)` Returns the last occurrence of character in the string str.

num must have same dimensions than str1 or only one char.

Examples

```
strrchr('This is a sample string','s')
strrchr(['This is a sample string','in scilab'],'s')
strrchr(['This is a sample string','in scilab'],['s','a'])
```

See Also

strchr , strstr

Name

strrev — returns string reversed

```
res = strrev(str1)
```

Parameters

str1
: A character string or matrix of character strings

res
: A character string or matrix of character strings

Description

`res = strrev(str1)` Returns string reversed.

Examples

```
rev = strrev('This is a simple string')
strrev(rev)
strrev(['This is a simple string','scilab'])
```

Name

strsplit — split a string into a vector of strings

```
v = strsplit(str,ind)
```

Parameters

str
: A character string

ind
: a vector of stricly increasing indices in the interval `[1 length(str)-1]` .

v
: the resulting column vector of strings (dimension `size(ind,'*')+1`).

Description

`v = strsplit(str,ind)` splits the string `str` into a vector of strings at the points given by the indices in `ind` (after each characters pointed to by the index in ind).

Examples

```
S='strsplit splits a string into a vector of strings';
strsplit(S,[15 25 30])
ind=strindex(S,' ')
```

See Also

strcat , tokens

Authors

S. Steer
: INRIA

Name

strspn — Get span of character set in string

```
res = strspn(str1,str2)
```

Parameters

str1
: A character string or matrix of character strings

str2
: A character string or matrix of character strings

res
: matrix.

Description

`res = strspn(str1,str2)` Returns the length of the initial portion of str1 which consists only of characters that are part of str2.

str2 must have same dimensions than str2 or str2 can be a string.

Examples

```
i = strspn("129th","1234567890");
printf ("The length of initial number is %d.\n",i);
i = strspn(["129th","130th"],["1234567890","130t"])
```

See Also

strcspn

Name

strstr — Locate substring

```
res = strstr(haystack,needle)
```

Parameters

haystack
: A character string or matrix of character strings

needle
: A character string or matrix of character strings

res
: A character string or matrix of character strings

Description

`res = strstr(haystack,needle)` Returns a string matrix starting from where the first occurrence of needle in haystack to the end of haystack, or '' if there needle is not part of haystack.

Examples

```
strstr('This is a simple string','simple')
strstr('This is a simple string','sample')
strstr(['This is a simple string','in scilab'],'is')
strstr(['This is a sample string','in scilab'],['a','scilab'])
```

See Also

strrchr, strchr

Name

strsubst — substitute a character string by another in a character string.

```
string_out=strsubst(string_in,searchStr,replaceStr)
string_out=strsubst(string_in,searchStr,replaceStr,[flag])
```

Parameters

string_in
: a matrix of character string. The strings where to search occurrences of `searchStr`

searchStr
: A character string. The string to search in `string`.

replaceStr
: A character string. The replacement string.

str_out
: A matrix of character strings. The result of the substitution on `searchStr` by `replaceStr` in `string`

flag
: string("r" for regular expression)

Description

`strsubst` replaces all occurrences of `searchStr` in `string` by `replaceStr`.

When using the forth parameters "r", the searchStr should be a string of regular expression. And then strsubst is going to match it with string and replace according to the regular express rules.

Examples

```
strsubst('SCI/demos/scicos','SCI','.')
strsubst('SCI/demos/scicos','/',' ')
strsubst('2' ,'/2(]*)?$\1/' ,'0','r')
```

See Also

string, strings

Name

strtod — Convert string to double.

```
d =  strtod(str)
[d,endstr] = strtod(str)
```

Parameters

str
: A character string or matrix of character strings

d
: A real or matrix of reals

endstr
: A character string or matrix of character strings (next character in str after the numerical value).

Description

`[d,endstr] = strtod(str)` Parses strings str interpreting its content as a floating point number and returns its value as a real.

Examples

```
strtod('123.556This is a sample real')
[d,endstr] = strtod('123.556This is a sample real')
strtod(['123.556This is a sample real','888.666 here'])
[d,endstr] =strtod(['123.556This is a sample real','888.666 here'])
```

Name

strtok — Split string into tokens

```
res = strtok(str,delimiters)
```

Parameters

str
: A character string

delimiters
: A character string

res
: A character string

Description

`res = strtok(str,delimiters)` sequence of calls to this function split str into tokens, which are sequences of contiguous characters spearated by any of the characters that are part of delimiters.

Examples

```
TOKENS = [];
token = strtok("A string of ,,tokens and some  more tokens"," ,");
TOKENS = [TOKENS,token];
while( token <> '' )
  token = strtok(" ,");
  TOKENS = [TOKENS,token];
end
disp(TOKENS);
```

See Also

strrchr , strchr

Name

tokenpos — returns the tokens positions in a character string.

```
kdf=tokenpos(str [,delimiter])
```

Parameters

str
: A character string. The string where to search the tokens.

delimiter
: (optional) A character or a vector of characters. The tokens delimeters.

kdf
: Two columns matrix, first column gives the index of the beginning of the tokens, the second gives the index of the last character of the tokens.

Description

`kdf=tokenpos(str [,delimiter])` searches the tokens included in the string `str`. The `delimiter` default value is [" ",<Tab>] where <Tab> stands for `ascii(9)`. It returns the indices of the first and last characters of each found tokens.

Examples

```
str='This is a character string';
kdf=tokenpos(str)
first=part(str,kdf(1,1):kdf(1,2))
```

See Also

strindex , tokens

Name

tokens — returns the tokens of a character string.

```
T=tokens(str [,delimiter])
```

Parameters

str
: A character string. The string where to search the tokens.

delimiter
: (optional) a character or a vector of characters. The tokens delimeters.

T
: column vector of found tokens

Description

`T=tokens(str [,delimiter])` searches the tokens included in the string `str`. The `delimiter` default value is [" ",<Tab>] where <Tab> stands for `ascii(9)`.

Examples

```
tokens('This is a character string')

tokens('SCI/demos/scicos','/')

tokens('y=a+b*2',['=','+','*'])
```

See Also

strindex , tokenpos

Name

tree2code — generates ascii definition of a Scilab function

```
txt=tree2code(tree,prettyprint)
```

Parameters

tree
: a macro tree (coming from macr2tree)

prettyprint
: optional boolean value

  %T
  : generated code is indented and beautified

  %F
  : generated code is not beautified (default)

txt
: a column vector of strings, the text giving the Scilab instructions

Description

Given a loaded Scilab function "tree" (returned by macr2tree), tree2code allows to re-generate the code.

Examples

```
tree=macr2tree(cosh);
txt=tree2code(tree,%T);
write(%io(2),txt,'(a)');
```

See Also

macr2tree

Authors

V.C.

Symbolic

Name

addf — symbolic addition

```
addf("a","b")
```

Parameters

"a","b"
: character strings

Description

`addf("a","b")` returns the character string `"a+b"`. Trivial simplifications such as `addf("0","a")` or `addf("1","2")` are performed.

Examples

```
addf('0','1')
addf('1','a')
addf('1','2')
'a'+'b'
```

See Also

mulf , subf , ldivf , rdivf , eval , evstr

Name

ldivf — left symbolic division

```
ldivf('d','c')
```

Description

returns the string `'c\d'` Trivial simplifications such as `'1\c' = 'c'` are performed.

Examples

```
ldivf('1','1')
ldivf('a','0')
ldivf('a','x')
ldivf('2','4')
```

See Also

rdivf , addf , mulf , evstr

Name

mulf — symbolic multiplication

```
mulf('d','c')
```

Description

returns the string `'c*d'` Trivial simplifications such as `'1*c' = 'c'` are performed.

Examples

```
mulf('1','a')
mulf('0','a')
'a'+'b'   //Caution...
```

See Also

rdivf , addf , subf

Name

rdivf — right symbolic division

```
["r"]=ldivf("d","c")
```

Parameters

"d","c","r"
: strings

Description

returns the string `"c/d"` Trivial simplifications such as `"c/1" = "c"` are performed.

Examples

```
ldivf('c','d')
ldivf('1','2')
ldivf('a','0')
```

See Also

ldivf

Name

subf — symbolic subtraction

```
["c"]=subf("a","b")
```

Parameters

"a","b","c"
: strings

Description

returns the character string `c="a-b"` Trivial simplifications such as `subf("0","a")` or `subf("1","2")` are performed.

Examples

```
subf('0','a')
subf('2','1')
subf('a','0')
```

See Also

mulf , ldivf , rdivf , eval , evstr

Tcl/Tk Interface

Name

ScilabEval — tcl instruction : Evaluate a string with scilab interpreter

```
ScilabEval instruction
ScilabEval instruction "seq"
ScilabEval instruction "sync"
ScilabEval instruction "sync" "seq"
ScilabEval "flush"
```

Parameters

instruction
: tcl string character contains a Scilab instruction to evaluate with the current Scilab interpreter.

Description

This function must be called in a tcl/tk script executed from Scilab. It allows to associate Scilab actions to tcl/tk widgets (graphic objects) or to use Scilab to perform some computations within a tcl script.

ScilabEval instruction
: If the `ScilabEval instruction` syntax is used, the `instruction` is first stored in a FIFO queue, `ScilabEval` returns immediately. Scilab executes the queued `instructions` when possible (it should be at the prompt but also at the end of each instructions of the currently running function) in the order they were submitted. This syntax can be used to associate Scilab actions to tcl/tk widgets but not into a tcl script executed by TCL_EvalFile or TCL_EvalStr because in this situation the Scilab interpreter is blocked up to the end of the script. Note that with the `ScilabEval instruction` syntax, if there are many `ScilabEval` commands stored in the queue the execution of the second one can be started in the middle of the execution of the first one (in particular if the first one contains more than a simple expression).

  If the `"seq"` option is added, the associated instruction evaluation should be finished (or paused) before the next queued instruction evaluation can be started. The next callback stored in the command queue will only be taken into account when the current one will be finished or paused.

ScilabEval instruction "sync"
: If the `ScilabEval instruction "sync"` syntax is used, the instruction is executed immediately (not queued) and the `ScilabEval`returns when the `instruction` evaluation is finished. The scilab `instruction` evaluation may be interrupted by new or queued commands.

  If the `"seq"` option is added, the associated instruction evaluation should be finished (or paused) before any queued instruction evaluation can be started. The scilab `instruction` evaluation may not be interrupted by new or queued commands (except if it is paused).

ScilabEval "flush"
: If the `ScilabEval "flush"` syntax is used, all the previously queued `instructions` are executed immediately and the ScilabEval returns when the execution is finished. Each `instruction` is executed with the option used at the time of queuing up (i.e. `seq` or no option).

The evaluation context of all these cases is the current Scilab context when the`instruction` evaluation starts.

Examples

```
//Callbacks and "seq" option usage

//create tcl instructions
tcl_script=['toplevel .w1'
'button .w1.b -text ""Click here to execute without seq option""  -command Witho
'button .w1.b1 -text ""Click here to execute with seq option""  -command  WithS
'pack .w1.b .w1.b1'
'proc  WithoutSeq {} { ';
'  ScilabEval ""cont=%f;;cont=%t;"" '
'  ScilabEval ""if cont then disp(''ok''),else disp(''wrong'');end;cont=%f;"" '
'}'
'proc WithSeq {} { ';
'  ScilabEval ""cont=%f;;cont=%t;""  ""seq""'
'  ScilabEval ""if cont then disp(''ok''),else disp(''wrong'');end;cont=%f;"" '
'}'];
mputl(tcl_script,TMPDIR+'/test.tcl') //write them to a file
// Execute the tcl script
cont=%f;
TCL_EvalFile(TMPDIR+'/test.tcl');;

//scripts and "sync" option usage

//----------------without "sync"----------------
tcl_script=['  set t ""0""'
    '  while {$t != ""10""} {'
    '     ScilabEval ""a=$t;mprintf(''%d '',a);""'
    '     incr t'
    '  }'];

mputl(tcl_script,TMPDIR+'/test.tcl') //write them to a file
// Execute the tcl script
TCL_EvalFile(TMPDIR+'/test.tcl');mprintf('TCL_EvalFile finished\n');
// The ScilabEval are executed  after the and of TCL_EvalFile

//----------------with "sync"----------------
tcl_script=['  set t ""0""'
    '  while {$t != ""10""} {'
    '     ScilabEval ""a=$t;mprintf(''%d '',a);"" ""sync""'
    '     incr t'
    '  }'];

mputl(tcl_script,TMPDIR+'/test.tcl') //write them to a file
// Execute the tcl script
TCL_EvalFile(TMPDIR+'/test.tcl');mprintf('TCL_EvalFile finished\n');
// The ScilabEval are executed  synchronously with TCL_EvalFile
```

See Also

TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_SetVar

Authors

Bertrand Guiheneuf

Name

TCL_CreateSlave — Create a TCL slave interpreter

```
TCL_CreateSlave(slaveName[, isSafe])
```

Parameters

slaveName
: String: Name of the TCL slave interpreter to create.

isSafe
: Boolean: %T to create a safe slave interpreter, %F otherwise. The default value is %F. A safe slave is not allowed to perform some operations, see the TCL documentation for more information.

Description

This routine allows to create a TCL slave interpreter.

Examples

```
TCL_CreateSlave("TCLinterp")
TCL_SetVar("a","r","TCLinterp")
TCL_ExistVar("a","TCLinterp")
TCL_ExistVar("a")
TCL_DeleteInterp("TCLinterp")

TCL_CreateSlave("TCLinterp", %T)
TCL_SetVar("a","r","TCLinterp")
TCL_ExistVar("a","TCLinterp")
TCL_ExistVar("a")
TCL_DeleteInterp("TCLinterp")
```

See Also

TCL_SetVar , TCL_ExistVar , TCL_DeleteInterp

Authors

Allan CORNET

V.C.

Name

TCL_DeleteInterp — delete TCL interpreter

```
TCL_DeleteInterp(interp)
TCL_DeleteInterp()
```

Parameters

interp
: character string parameter. Name of the slave tcl interpreter to delete. If not provided, it defaults to the main tcl interpreter created by Scilab.

Description

This routine allows to delete a TCL slave interpreter or the main scilab TCL interpreter.

Examples

```
TCL_SetVar("Scilab","OK")
TCL_ExistVar("Scilab")
TCL_DeleteInterp()
TCL_ExistVar("Scilab")
TCL_CreateSlave('BisInterp')
TCL_ExistInterp('BisInterp')
TCL_SetVar("Scilab","OK",'BisInterp')
TCL_ExistVar("Scilab",'BisInterp')
TCL_DeleteInterp('BisInterp')
TCL_ExistInterp('BisInterp')
```

See Also

TCL_SetVar , TCL_ExistVar , TCL_CreateSlave , TCL_ExistInterp

Authors

Allan CORNET

Name

TCL_EvalFile — Reads and evaluate a tcl/tk file

```
TCL_EvalFile(filename [,interp])
```

Parameters

filename
: character string. Contains the name of the file to read and evaluate.

interp
: optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

Description

With this routine, one can read and evaluate the content of a file containing tcl/tk scripts. This allows to create powerful tk interfaces.

The filename might be relative or absolute.

Advantages and drawbacks of this functionality

This routines allows to use directly tcl/tk scripts. This thus allows, for instance to use Interface Builders such as SpecTcl to design the interface. The interfaces built directly with tcl/tk scripts are much faster than the ones built with the Scilab Graphic Object library provided with tksci (see uicontrol for example). Indeed, those Objects are warpings around tk graphic widgets. Nevertheless, this way of creating graphic user interface should only be used when one aims at addressing directly specific tk/tcl features. There are two main reasons for this. First of all, there is no simple way to manipulate Scilab objects from within a tcl/tk script. Thus, the interface designer has to write two sets of callbacks routines. One to describe the changes occuring in the interface when the user acts on the widgets. The second set of call routines will perform the (pure) Scilab reactions to the user actions.

Here is an example: Suppose you design a scrollbar corresponding to a spline tension value. You want the spline to be displayed in a graphic windows and updated each time the user moves the scrollbar. At the same time, you want the value of this tension parameter to be displayed within the Interface. You will have to write a first tcl/tk (callback) function which will be automatically called by the tk scrollbar ("-command" option). This callback function will update the displayed value of the parameter in the interface and will then call the scilab routine ("ScilabEval" command) to update the graph.

Remarks on the tcl/tk script style

Because Scilab manages the tcl/tk events, it creates the root window ".", this window should not be destroyed nor directly used by your tcl/tk scripts. You should thus always create your own toplevel windows. Moreover, since this module was written at a time when namespaces didn't exist, some variables defined by scilab tcl/tk scripts could collide your code. Running your scripts in a slave interpreter may help in such a case.

Examples

```
TCL_EvalFile(SCI+"/modules/tclsci/demos/tk/puzzle");
scipad();
TCL_EvalFile(SCI+"/modules/tclsci/demos/tk/puzzle","scipad");
```

See Also

ScilabEval , TCL_EvalStr , TCL_GetVar , TCL_SetVar , TCL_ExistVar , TCL_UnsetVar , TCL_UpVar

Authors

Allan CORNET

Name

TCL_EvalStr — Evaluate a string whithin the Tcl/Tk interpreter

```
TCL_EvalStr(str [,interp])
res = TCL_EvalStr(str [,interp])
```

Parameters

str
: string or matrix of strings, contains a Tcl/Tk script in each element.

interp
: optional character string parameter. Name of the slave Tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main Tcl interpreter created by Scilab.

res
: result of the evaluation, if it is successful. This is a character string matrix giving the evaluation result for each element of the input argument str

Description

This routine allows to evaluate Tcl/Tk instructions with the Tcl/Tk interpreter launched with Scilab (when the `interp` parameter is not given), or in a slave interpreter.

When Tcl/Tk support is enabled in Scilab, you can evaluate Tcl/Tk expression from Scilab interpreter. In fact, Scilab launches a main Tcl/Tk interpreter. The Scilab instruction `TCL_EvalStr` can be used to evaluate expressions without having to write Tcl/Tk instructions in a separated file (this capability is provided by `TCL_EvalFile`).

Examples

```
//with one call
TCL_EvalStr(["toplevel .foo1"
   "label .foo1.l -text ""TK married Scilab !!!"""
   "pack .foo1.l"
   "button .foo1.b -text close -command {destroy .foo1}"
   "pack .foo1.b"])

//step by step (debugging)
TCL_EvalStr("toplevel .foo2");
// creates a toplevel TK window.
TCL_EvalStr("label .foo2.l -text ""TK married Scilab !!!""");
// create a static label
TCL_EvalStr("pack .foo2.l");
// pack the label widget. It appears on the screen.
text="button .foo2.b -text close -command {destroy .foo2}";
TCL_EvalStr(text);
TCL_EvalStr("pack .foo2.b");

//kill the windows by program
TCL_EvalStr("destroy .foo1");
TCL_EvalStr("destroy .foo2");

//with one call, and in a slave interpreter
TCL_CreateSlave('TCLSlave');
```

```
TCL_EvalStr('set test ""in Slave TCL Interp""','TCLSlave');
TCL_GetVar('test','TCLSlave')

TCL_DeleteInterp('TCLSlave')

// return a result
res = TCL_EvalStr("expr 1+1")
res = TCL_EvalStr("tk_messageBox -message Hello -type okcancel")
res = TCL_EvalStr(["expr 4+5" "lsearch -all {a b c a b c} c" ; "list [list a b
```

See Also

ScilabEval , TCL_EvalFile , TCL_GetVar , TCL_SetVar , TCL_ExistVar , TCL_UnsetVar , TCL_UpVar

Authors

Allan CORNET

Name

TCL_ExistArray — Return %T if a tcl array exists

```
OK=TCL_ExistArray(arrayname [,interp])
```

Parameters

arrayname
: character string. Contains the name of the tcl/tk array.

interp
: optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

ok
: boolean. %T if arrayname exists.

Description

This routine allows to test if a tcl array exists.

Examples

```
 TCL_ExistVar("A")
a=["A","B","C";"D","E","F"];
TCL_SetVar("A",a)
TCL_ExistVar("A")
TCL_ExistArray("A")
```

See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_SetVar , TCL_UnsetVar , TCL_UpVar , TCL_CreateSlave

Authors

Allan CORNET

Name

TCL_ExistInterp — Return %T if a tcl slave interperter exists

```
OK=TCL_ExistInterp(interp)
```

Parameters

interp
: character string parameter. Name of the slave tcl interpreter.

ok
: boolean. %T if TCL interpreter exists.

Description

This routine allows to test if TCL interpreter exists.

Examples

```
TCL_ExistInterp('SlaveInterp')
TCL_CreateSlave('SlaveInterp')
TCL_ExistInterp('SlaveInterp')
TCL_DeleteInterp('SlaveInterp')
```

See Also

TCL_CreateSlave , TCL_DeleteInterp

Authors

Allan CORNET

Name

TCL_ExistVar — Return %T if a tcl variable exists

```
OK=TCL_ExistVar(varname [,interp])
```

Parameters

varname
: character string. Contains the name of the tcl/tk variable.

interp
: optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

ok
: boolean. %T if varname exists.

Description

This routine allows to test if a tcl variable exists.

Examples

```
TCL_SetVar("Scilab","OK")
TCL_GetVar("Scilab")
TCL_UnsetVar("Scilab")
TCL_ExistVar("Scilab")

TCL_SetVar("aa",1)
TCL_CreateSlave('SlaveInterp');
TCL_SetVar("aa",2,'SlaveInterp')
TCL_ExistVar("aa")
TCL_GetVar("aa")
TCL_UnsetVar("aa")
TCL_GetVar("aa",'SlaveInterp')
TCL_UnsetVar("aa",'SlaveInterp')
TCL_ExistVar("aa",'SlaveInterp')
TCL_DeleteInterp('SlaveInterp')
```

See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_SetVar , TCL_UnsetVar , TCL_UpVar , TCL_CreateSlave

Authors

Allan CORNET

Name

TCL_GetVar — Get a tcl/tk variable value

```
value=TCL_GetVar(Varname [,interp])
```

Parameters

varname
: character string. Contains the name of the tcl/tk variable.

interp
: optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

value
: may be a character string or a strings matrix. Contains the value of the tcl/tk variable `varname` in the interpreter `interp`.

Description

When tcl/tk support is enabled in Scilab, this routine can be used to retreive the value of a tcl/tk variable.

Examples

```
//-----------------------------------------------------
TCL_EvalStr("toplevel .tst1");
// creates a toplevel TK window.
TCL_EvalStr("entry .tst1.e -textvariable tvar");
// create an editable entry
TCL_EvalStr("set tvar foobar");
// set the entry value
TCL_EvalStr("pack .tst1.e");
// pack the entry widget. It appears on the screen.
text=TCL_GetVar("tvar")
// retrieve the variable value
// change the entry text and repeat the last command ...
//delete the toplevel TK window.
TCL_EvalStr("destroy .tst1")
//-----------------------------------------------------
a=["A","B","C";"D","E","F"];
TCL_SetVar("A",a)
AfromTCL=TCL_GetVar("A")
//-----------------------------------------------------
b=[6,4,1;2,3,5];
TCL_SetVar("B",b)
BfromTCL=TCL_GetVar("B")
//-----------------------------------------------------
TCL_SetVar("StringTCL","string")
StringFromTCL=TCL_GetVar("StringTCL")
//-----------------------------------------------------
TCL_SetVar("ScalarTCL",1.22)
ScalarFromTCL=TCL_GetVar("ScalarTCL")
//-----------------------------------------------------
// Examples with a slave interpreter
//-----------------------------------------------------
```

```
a=['AA','BB','CC';'DD','EE','FF'];
TCL_CreateSlave('SlaveInterp')
TCL_SetVar("A_slave",a,'SlaveInterp')
AfromTCL_slave=TCL_GetVar('A_slave','SlaveInterp')
TCL_DeleteInterp('SlaveInterp')
//-----------------------------------------------------
b=[66,44,11;22,33,55];
TCL_CreateSlave('SlaveInterp1')
TCL_SetVar("B_slave",b,'SlaveInterp1')
BfromTCL_slave=TCL_GetVar('B_slave','SlaveInterp1')
TCL_DeleteInterp('SlaveInterp1')
//-----------------------------------------------------
TCL_CreateSlave('SlaveInterp2')
TCL_SetVar("StringTCL_slave","string in slave interpreter",'SlaveInterp2')
StringFromTCL_slave=TCL_GetVar("StringTCL_slave",'SlaveInterp2')
TCL_DeleteInterp('SlaveInterp2')
//-----------------------------------------------------
TCL_CreateSlave('SlaveInterp3')
TCL_SetVar("ScalarTCL_slave",1.22,'SlaveInterp3')
ScalarFromTCL_slave=TCL_GetVar("ScalarTCL_slave",'SlaveInterp3')
TCL_DeleteInterp('SlaveInterp3')
//-----------------------------------------------------
```

See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_SetVar , TCL_ExistVar , TCL_UnsetVar , TCL_UpVar , TCL_CreateSlave , TCL_DeleteInterp

Authors

Allan CORNET

Name

TCL_GetVersion — get the version of the TCL/TK library at runtime.

```
TCL_GetVersion()
ret=TCL_GetVersion('numbers')
```

Description

get the version of the TCL/TK library at runtime.

ret=TCL_GetVersion('numbers') returns a matrix with the version of the TCL/TK library at runtime.

Examples

```
TCL_GetVersion()
TCL_GetVersion("numbers")
```

Authors

Allan CORNET

Name

TCL_SetVar — Set a tcl/tk variable value

```
TCL_SetVar(varname, value [,interp])
```

Parameters

varname
: character string. Contains the name of the tcl/tk variable to set.

value
: may be a character string, a scalar, a real or string matrix (m x n). Contains the value to give to the tcl/tk variable.

interp
: optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

Description

This routine allows to set a variable within a tcl/tk interpreter. When tcl/tk support is enabled in scilab, this routine can be used to set up the value of a tcl/tk variable. This can be useful to change some value in the tcl/tk interpreter without having to build a tcl/tk instruction (and use `TCL_EvalStr`).

Examples

```
//-----------------------------------------------------
TCL_EvalStr("toplevel .tst1");
// creates a toplevel TK window.
TCL_EvalStr("entry .tst1.e -textvariable tvar");
// create an editable entry
TCL_EvalStr("set tvar foobar");
// set the entry value
TCL_EvalStr("pack .tst1.e");
// pack the entry widget. It appears on the screen.
text=TCL_GetVar("tvar")
// retrieve the variable value
// change the entry text and repeat the last command ...
//delete the toplevel TK window.
TCL_EvalStr("destroy .tst1")
//-----------------------------------------------------
a=["A","B","C";"D","E","F"];
TCL_SetVar("A",a)
AfromTCL=TCL_GetVar("A")
//-----------------------------------------------------
b=[6,4,1;2,3,5];
TCL_SetVar("B",b)
BfromTCL=TCL_GetVar("B")
//-----------------------------------------------------
TCL_SetVar("StringTCL","string")
StringFromTCL=TCL_GetVar("StringTCL")
//-----------------------------------------------------
TCL_SetVar("ScalarTCL",1.22)
ScalarFromTCL=TCL_GetVar("ScalarTCL")
```

```
//-----------------------------------------------------
// Examples with a slave interpreter
//-----------------------------------------------------
TCL_CreateSlave('TCLSlave')
a=['AA','BB','CC';'DD','EE','FF'];
TCL_SetVar("A_slave",a,'TCLSlave')
AfromTCL_slave=TCL_GetVar('A_slave','TCLSlave')
TCL_DeleteInterp('TCLSlave')
//-----------------------------------------------------
TCL_CreateSlave('TCLSlave')
b=[66,44,11;22,33,55];
TCL_SetVar("B_slave",b,'TCLSlave')
BfromTCL_slave=TCL_GetVar('B_slave','TCLSlave')
TCL_DeleteInterp('TCLSlave')
//-----------------------------------------------------
TCL_CreateSlave('TCLSlave')
TCL_SetVar("StringTCL_slave","string in slave interpreter",'TCLSlave')
StringFromTCL_slave=TCL_GetVar("StringTCL_slave",'TCLSlave')
TCL_DeleteInterp('TCLSlave')
//-----------------------------------------------------
TCL_CreateSlave('TCLSlave')
TCL_SetVar("ScalarTCL_slave",1.22,'TCLSlave')
ScalarFromTCL_slave=TCL_GetVar("ScalarTCL_slave",'TCLSlave')
TCL_DeleteInterp('TCLSlave')
//-----------------------------------------------------
```

See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_ExistVar , TCL_UnsetVar , TCL_UpVar , TCL_CreateSlave , TCL_DeleteInterp

Authors

Allan CORNET

Name

TCL_UnsetVar — Remove a tcl variable

```
OK=TCL_UnsetVar(varname [,interp])
```

Parameters

varname
: character string. Contains the name of the tcl/tk variable to unset.

interp
: optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

ok
: boolean. %T if varname was deleted.

Description

This routine allows to unset a tcl variable.

Examples

```
TCL_SetVar("Scilab","OK")
TCL_GetVar("Scilab")
TCL_UnsetVar("Scilab")
TCL_ExistVar("Scilab")

TCL_CreateSlave('InterpSlave');
TCL_SetVar("Scilab","Good",'InterpSlave')
TCL_GetVar("Scilab",'InterpSlave')
TCL_UnsetVar("Scilab",'InterpSlave')
TCL_ExistVar("Scilab",'InterpSlave')
TCL_DeleteInterp('InterpSlave')
```

See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_SetVar , TCL_ExistVar , TCL_UpVar , TCL_CreateSlave , TCL_DeleteInterp

Authors

Allan CORNET

Name

TCL_UpVar — Make a link from a tcl source variable to a tcl destination variable

```
OK=TCL_UpVar(varname1,varname2,[interp])
```

Parameters

varname1
: character string. Contains the name of the tcl source variable.

varname2
: character string. Contains the name of the tcl destination variable.

interp
: optional character string parameter. Name of the slave tcl interpreter in which the operation has to be performed. If not provided, it defaults to the main tcl interpreter created by Scilab.

ok
: boolean. %T if it is ok.

Description

Make a link from a tcl source variable to a tcl destination variable.

Examples

```
TCL_SetVar("Scilab","OK")
TCL_UpVar("Scilab","ScilabBis")
TCL_GetVar("ScilabBis")
TCL_SetVar("Scilab","NOK")
TCL_GetVar("ScilabBis")
TCL_SetVar("ScilabBis","modified")
TCL_GetVar("ScilabBis")
TCL_GetVar("Scilab")

TCL_CreateSlave('InterpBis')
TCL_SetVar("Scilab","Good",'InterpBis')
TCL_UpVar("Scilab","ScilabBis",'InterpBis')
TCL_GetVar("ScilabBis",'InterpBis')
TCL_SetVar("Scilab","Not good",'InterpBis')
TCL_GetVar("ScilabBis",'InterpBis')
TCL_SetVar("ScilabBis","modified again",'InterpBis')
TCL_GetVar("ScilabBis",'InterpBis')
TCL_GetVar("Scilab",'InterpBis')
TCL_DeleteInterp('InterpBis')
```

See Also

ScilabEval , TCL_EvalFile , TCL_EvalStr , TCL_GetVar , TCL_SetVar , TCL_ExistVar , TCL_UnsetVar , TCL_CreateSlave , TCL_DeleteInterp

Authors

Allan CORNET

Name

browsevar — Scilab variable browser

```
browsevar()
```

Description

`browsevar` is an embedded Scilab variable browser written in TCL/TK.

`browsevar` can show all variables and function (like who). `browsevar` can be costumized to show all or some type of variable. It's also posibble exclude variable names.

Examples

```
browsevar();
```

Authors

Jaime Urzua

Name

config — Scilab general configuration.

```
config()
```

Description

`config()` allows configure scilab parameters like lines to display, stacksize, %ODEOPTIONS.

Authors

Jaime Urzua

Name

editvar — Scilab variable editor

```
editvar varname
```

Parameters

varname
: variable name. The variable must exist in scilab.

Description

`editvar` is an embedded Scilab variable editor written in TCL/TK.

`editvar` can edit the following variable type: real or complex constant matrix (type 1), boolean matrix (type 4) an matrix of character strings (type 10).

Examples

```
a=rand(10,10);
editvar a;
b=['hello';'good bye'];
editvar b;
```

Authors

Jaime Urzua

Name

tk_getdir — dialog to get a directory path

```
path=tk_getdir([Title="string"])
path=tk_getdir(startdir,[Title="string"])
path=tk_getdir(startdir,windowtitle)
```

Parameters

startdir
: a character string which gives the initial directory used for directory search. By default tk_getdir uses the previously selected directory.

path
: is the user selected file path if user answers "Ok" or the " " string if user answers "Cancel"

Title="string"
: Optional argument which gives the title for the tk_getdir window. `Warning`: Use the new variable `Title` instead of the old variable `title`.

Description

Creates a dialog window for file selection.

Examples

```
tk_getdir()
tk_getdir("SCI/modules/")
tk_getdir(Title="Choose a directory name")
```

See Also

uigetfile , file , fileinfo

Name

tk_getfile — dialog to get one or more file paths (obsolete)

```
path=tk_getfile([Title="string"])
path=tk_getfile([multip="1"])
path=tk_getfile(file_mask[,Title="string"][,multip="1"])
path=tk_getfile(file_mask,dir[,Title="string"])
path=tk_getfile(file_mask,dir[,Title="string"][,multip="1"])
path=tk_getfile(file_mask,dir,"string"[,"multip"])
```

Parameters

file_mask
: a character string which gives the file mask to use for file selection. `file_mask` is written with Unix convention. The default value is '*'.

dir
: a character string which gives the initial directory used for file search. By default tk_getfile uses the previously selected directory.

path
: is the user selected file path(s) if user answers "Ok" or the " " string if user answers "Cancel".

Title="string"
: Optional argument which gives the title for the tk_getfile window. `Warning`: Use the new variable `Title` instead of the old variable `title`.

multip
: Optional argument which allows to select more than one file at once in the tk_getfile window. If given, it must be the string "1". Otherwise, or if not given, this argument defaults to "0" i.e. only one file can be selected in the dialog.

Description

Creates a dialog window for file selection.

This function is obsolete and will be removed in Scilab 5.2, please use uigetfile instead.

Examples

```
tk_getfile()
tk_getfile("*.sci","SCI/modules/graphics/macros")
tk_getfile(Title="Choose a file name")
tk_getfile(Title="Choose many file names at once",multip="1")
tk_getfile(multip="1")
```

See Also

uigetfile , tk_getdir , file , fileinfo

Name

tk_savefile — dialog to get a file path for writing

```
path=tk_savefile([Title='string'])
path=tk_savefile(file_mask,[Title='string'])
path=tk_savefile(file_mask,dir,[Title='string'])
path=tk_savefile(file_mask,dir,'string')
```

Parameters

file_mask
: a character string which gives the file mask to use for file selection. `file_mask` is written with Unix convention. the default value is '*'.

dir
: a character string which gives the initial directory used for file search. by default tk_savefile uses the previously selected directory.

path
: is the user selected file path if user answers "Ok" or the " " string if user answers "Cancel"

Title='string'
: :Optional argument which gives the title for the tk_savefile window. `Warning`: Use the new variable `Title` instead of the old variable `title`.

Description

Creates a dialog window for output file selection

Examples

```
tk_savefile()
tk_savefile('*.sci','SCI/modules/graphics/macros')
tk_savefile(Title='Choose a file name ')
```

See Also

uigetfile , tk_getdir , file , fileinfo

Name

winclose — close windows created by sciGUI

```
winclose(winIds)
```

Parameters

winIds
: matrix of integer greater than 0, window identificator.

Description

`winclose(winIds)` close windows created by sciGUI.

Examples

```
//CREATE SOME WINDOWS
win1=waitbar('This is an example');
win2=waitbar('HELLO!');
winclose([win1,win2]);
```

Authors

Jaime Urzua

Name

winlist — Return the winId of current window created by sciGUI

```
winIds=winlist()
```

Parameters

winIds
: matrix of integer greater than 0, window identificator.

Description

`winlist()` Return the winId of current window created by sciGUI.

Authors

Jaime Urzua

Texmacs

Name

pol2tex — convert polynomial to TeX format

```
[y]=pol2tex(x)
```

Parameters

x
: polynomial

y
: list

Description

Latex source code for the polynomial `x`. (For use with `texprint`)

Examples

```
s=poly(0,'s');
p=s^3+2*s-5;
pol2tex(p)
```

See Also

texprint

Name

texprint — TeX output of Scilab object

```
[text]= texprint(a)
```

Parameters

a
: Scilab object

text
: list

Description

returns the Tex source code of the Scilab variable `a`. `a` is a matrix (constant, polynomial, rational) or a linear system (`syslin` list).

Examples

```
s=poly(0,'s');
texprint([1/s,s^2])
```

See Also

pol2tex , pol2str

Time and Date

Name

calendar — Calendar

```
c=calendar()
c = calendar(y,m)
```

Description

c = calendar returns a list containing a calendar for the current month. The calendar runs Sunday to Saturday.

c = calendar(y,m), where y and m are integers, returns a calendar for the specified month of the specified year.

Examples

```
calendar()
calendar(1973,8)
```

See Also

datevec , datenum

Authors

Allan CORNET

Name

clock — Return current time as date vector

```
c = clock
```

Description

c = clock returns a 6-element date vector containing the current date and time in decimal form:

c = [year month day hour minute seconds]

the first five elements are integers. The seconds element is accurate to several digits beyond the decimal point.

Examples

```
clock
```

See Also

datenum , datevec , timer , etime , tic , toc

Authors

P.M

Name

date — Current date as date string

```
dt=date()
```

Parameters

dt
: a string

Description

dt=date() returns a string containing the date in dd-mmm-yyyy format.

Examples

```
date()
```

See Also

getdate , toc , tic , timer , etime

Name

datenum — Convert to serial date number

```
N = datenum()
N = datenum(DT)
N = datenum(Y, M, D)
N = datenum(Y, M, D, H, MI, S)
```

Description

The datenum function converts date vectors (defined by datevec) into serial date numbers. Date numbers are serial days elapsed from some reference date. By default, the serial day 1 corresponds to 1-Jan-0000.

N = datenum() returns the serial date numbers corresponding to current date.

N = datenum(DT) converts one or more date vectors to serial date number N. DT can be an m-by-6 or m-by-3 matrix containing m full or partial date vector respectively.

N = datenum(Y, M, D) returns the serial date numbers for corresponding elements of the Y, M, and D (year, month, day) arrays. Y, M and D must be arrays of the same size (or any can be a scalar).

N = datenum(Y, M, D, H, MI, S) returns the serial date numbers for corresponding elements of the Y, M, D, H, MI, and S (year, month, day, hour, minute, and second) array values.Y, M, D, H, MI, and S must be arrays of the same size (or any can be a scalar).

Examples

```
// N = datenum()
datenum()

// N = datenum(DT)
A = [ 0 1 1 0 0 0 ; 2005 2 8 21 37 30 ]
datenum(A)

// N = datenum(Y, M, D)

Years = [0; 1973; 2006]
Months = [1; 8; 2]
Days = [1; 4; 8]
datenum(Years,Months,Days)

Years = [0 0 0 ; 0 0 0]
Months = [1 1 1 ; 1 1 1]
Days = [1 2 3 ; 4 5 6]
datenum(Years,Months,Days)

// N = datenum(Y, M, D, H, MI, S)

Years = grand(5,10,'uin',0,2006)
Months = grand(5,10,'uin',1,12)
Days = grand(5,10,'uin',1,28)
Hours = grand(5,10,'uin',0,23)
Minutes = grand(5,10,'uin',0,59)
Seconds = grand(5,10,'uin',0,59)
datenum(Years,Months,Days,Hours,Minutes,Seconds)
```

See Also

datevec , calendar

Authors

A.C

Name

datevec — Date components

```
V=datevec(DT)
[Y,M,D,H,MI,S]=datevec(DT)
```

Description

V = datevec(DT) converts a serial date number (defined by datenum) to a date vector V having elements [year, month, day, hour, minute, second]. The first five vector elements are integers. DT can be an array.

[Y, M, D, H, MI, S] = datevec(DT) returns the components of the date vector as individual variables. DT can be an array.

Examples

```
// First example
datevec(720840)

// Second example
datevec(datenum())

// Third example (With integers values)
A = grand(10,12,'uin',1,1000000)
datevec(A)

// Fourth example (With real values)
A = grand(10,12,'unf',1,1000000)
datevec(A)
```

See Also

datenum , calendar

Authors

A.C

Name

eomday — Return last day of month

```
E = eomday(Y, M)
```

Description

E = eomday(Y, M) returns the last day of the year and month given by corresponding elements of arrays Y and M.

Examples

```
eomday(2006,3);
```

See Also

datenum , datevec , weekday

Authors

P.M

Name

etime — Elapsed time

```
e = etime(t2,t1)
```

Parameters

t2
: a vector with 6 or 10 values.

t1
: a vector with 6 or 10 values.

e
: number of seconds between `t2` and `t1`.

Description

t1 and t2 with 10 values
: `t2` and `t1` must have format returned by `getdate`. In this case, their third, fourth and fifth values are ignored.

t1 and t2 with 6 values
: `t2` and `t1` must have format: T = [Year Month Day Hour Minute Second] with Second a number of seconds with milliseconds (e.g: 12.345).

`t2` and `t1` must have the same size.

`t2` et `t1` can be matrices with each line containing a format described above (all lines having same format).

Examples

```
t1=[2004 06 10 17 00 12.345]
t2=[2004 06 10 17 01 13.965]
E1=etime(t2,t1)
t1=[2004 06 24 162 5 10 17 00 12 345]
t2=[2004 06 24 162 5 10 17 01 13 965]
E2=etime(t2,t1)
```

See Also

tic , toc , getdate , datenum , datevec

Authors

V.C.

Name

getdate — get date and time information

```
dt=getdate()
x=getdate("s")
dt=getdate(x)
```

Parameters

dt
: an integer vector with 10 entries (see below)

x
: an integer containing a date coded in second from 1 Jan 1970

Description

dt=getdate()
: returns the current date in format given below:

  dt(1)
  : The year as a number (with the century) between 0000 and 9999.

  dt(2)
  : The month of the year as a number between 01 and 12.

  dt(3)
  : The ISO 8601 week number as a number between 01 and 53.

  dt(4)
  : The Julian day of the year as a number between 001 and 366.

  dt(5)
  : Specifies the weekday as a decimal number [1,7], with 1 representing Sunday.

  dt(6)
  : The day of the month as a number between 01 and 31.

  dt(7)
  : The hour of the day is output as a number between 00 and 23.

  dt(8)
  : The minute is output as a number between 00 and 59.

  dt(9)
  : The second is output as a number between 00 and 59.

  dt(10)
  : The millisecond is output as a number between 000 and 999.

x=getdate("s")
: returns a scalar with the number of seconds since Jan 1, 1970, 00:00 UTC (Unix Time Convention)

dt=getdate(x)
: formats the date given by x (number of seconds since Jan 1, 1970, 00:00 UTC) in format given above. In this case dt(10) is always equal to 0.

Examples

```
w=getdate()
mprintf("Year:%d,Month:%d,Day:%d",w(1),w(2),w(6));

x=getdate("s")
getdate(x)
```

See Also

date , timer

Authors

V.C.

Name

now — Return current date and time

```
t = now()
```

Description

t = now() returns date and time as a serial date number. (See datenum)

Examples

```
realtimeinit(1);
realtime(0);
t1 = now()
datevec(t1)
realtime(10);
t1 = now()
datevec(t1)
```

See Also

clock , datenum , datevec

Authors

P.M

Name

realtimeinit — set time unit
realtime — set dates origin or waits until date

```
realtimeinit(time_unit)
realtime(t)
```

Parameters

time_unit
: a real number. The number of seconds associated to the `realtime` argument

t
: a real number. A date

Description

These two functions can be used to handle real time into Scilab.

`realtimeinit(time_unit)` defines the time unit associated to the `realtime` argument `t`

first call to `realtime(t0)` sets current date to (`t0`). subsequent calls to `realtime(t)` wait till date `t` is reached.

Examples

```
 realtimeinit(1/2);//sets time unit to half a second
 realtime(0);//sets current date to 0
 for k=1:10,realtime(k);mprintf('current time is '+string(k/2)+'sec .\r\n');en

//next instruction outputs a dot each 2 seconds
 realtimeinit(2); realtime(0);for k=1:10,realtime(k);mprintf('.\r\n');end

 realtimeinit(1);realtime(0);
 dt=getdate('s'); realtime(10);   getdate('s')-dt
```

See Also

getdate

Name

sleep — suspend Scilab

```
sleep(milliseconds)
```

Description

`sleep` : Sleep process for specified number of miliseconds specified by the argument. The actual suspension time may be longer because of other activities in the system, or because of the time spent in processing the call.

Examples

```
tic;sleep(6000);toc
```

See Also

xpause

Authors

Allan CORNET

Name

tic — start a stopwatch timer

```
tic()
```

Description

The sequence of commands `tic(); operation; toc();` prints the number of seconds required for the operation.

Examples

```
tic();
realtimeinit(1);
realtime(0);
realtime(10);
toc();
```

See Also

toc , timer , etime

Authors

V.C.
A.C.

Name

timer — cpu time

```
timer()
```

Description

Returns the CPU time since the preceding call to `timer()`.

timer has a time precision of 100 nanoseconds.

NOTE: CPU time is the number of processor cycles used for a computation. This is not at all equivalent to real-world time.

CPU time can be used to compare CPU usage between different programs or functions , irrespective of background processes that might slow down the computer.

Examples

```
timer();A=rand(100,100);timer()
```

See Also

getdate, toc, tic, etime

Name

toc — Read the stopwatch timer

```
toc()
t = toc()
```

Parameters

t
: number of seconds since last call to `tic()` (Precision in order of millisecond).

Description

The sequence of commands `tic(); operation; toc();` prints the number of seconds required for the operation.

Examples

```
tic();
realtimeinit(1);
realtime(0);
realtime(10);
toc();
```

See Also

tic , timer , etime

Authors

V.C.
A.C.

Name

weekday — Return day of week

```
[N,S] = weekday(D)
[N,S] = weekday(D, form)
```

Description

[N,S] = weekday(D) returns the day of the week in numeric(N) and string(S) form for a given serial date number or date string D. Input argument D can represent more than one date in an array of serial date number.

[N,S] = weekday(D, form) returns the week in numeric(N) and string(S) form, where the content of S depends on the form argument. If form is 'long', then S countains the full name of the weekday (e.g, Thuesday). If form is 'short', then S contains an abbreviated name (e.g., Tue) from this table.

Examples

```
today = datenum();
[N,S] = weekday(today)
[N,S] = weekday(today,'short')
[N,S] = weekday(today,'long')
```

See Also

datenum , datevec , weekday

Authors

P.M

UMFPACK Interface

Name

PlotSparse — plot the pattern of non nul elements of a sparse matrix

```
PlotSparse(A [,style])
```

Parameters

A
: a sparse matrix

style
: (optional) a string given the color and/or the marker type of the form "[color][mark]" where color may be a number referring the color you want to use (in the current colormap). If you use the std colormap then color may be one of the following letters :

```
k  for black       b  for blue
r  for red         g  for green
c  for cyan        m  for magenta
y  for yellow      t  for turquoise
G  a dark green
```

mark must be one of the following :

```
.  point           +  plus
x  cross           *  circled plus
D  filled diamond  d  diamond
^  upper triangle  v  down triangle
o  circle
```

by default you have "b." (in fact the 2d color) and this is also forced in case of error.

Description

plot the pattern of non nul elements of a sparse matrix : each non nul element is drawn with a marker. For "big" matrix use essentially the point . as marker

Examples

```
[A,description,ref,mtype] = ReadHBSparse(SCI+"/modules/umfpack/examples/arc130.
set figure_style old
PlotSparse(A,"y+")
xtitle(ref + "." + mtype + " : " + description)
```

See Also

ReadHBSparse

Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

Name

ReadHBSparse — read a Harwell-Boeing sparse format file

```
[A, description, ref, mtype] = ReadHBSparse([filename])
```

Parameters

filename
: (optional) a string given the filename (eventually preceeding by the path), if filename is not given then the function use uigetfile to get filename interactively

A
: the sparse matrix

description
: a string given some information about the matrix

ref
: a string given the reference of the matrix

mtype
: a string given the type of the matrix

Description

An utility to read the Harwell-Boeing sparse matrix format. Currently don't work for unassembled matrix. Also the eventual rhs vectors of the file are not returned. Generally the file name is of the form ref.mtype where mtype is a 3 letters word abc given some information (already inside the file) on the matrix :

```
a = R|C|P   for real|complex|pattern (no values given)
b = S|H|Z|U for symetric|hermitian|skew symetric|unsymetric
c = A|E     for assembled|unassembled matrix
            (case E is not treated by this func)
```

References

Users' Guide for the Harwell-Boeing Sparse Matrix Collection Iain S. Duff, Roger G. Grimes, John G. Lewis. You may found this guide and numerous sparse matrices (in the Harwell-Boeing format) at the University of Florida Sparse Matrix Collection

web site : http://www.cise.ufl.edu/research/sparse/matrices/

maintained by Tim Davis (http://www.cise.ufl.edu/~davis/)

Examples

```
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/arc130.rua");
```

See Also

PlotSparse

Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

Name

cond2sp — computes an approximation of the 2-norm condition number of a s.p.d. sparse matrix

```
[K2, lm, vm, lM, vM] = cond2sp(A, C_ptr [, rtol, itermax, verb])
```

Parameters

A
: a real symetric positive definite sparse matrix

C_ptr
: a pointer to a Cholesky factorization (got with taucs_chfact)

rtol
: (optional) relative tolerance (default 1.e-3) (see details in DESCRIPTION)

itermax
: (optional) maximum number of iterations in the underlying algorithms (default 30)

verb
: (optional) boolean, must be %t for displaying the intermediary results, and %f (default) if you don't want.

K2
: estimated 2-norm condition number `K2 = ||A||_2 ||A^(-1)||_2 = lM/lm`

lm
: (real positive scalar) minimum eigenvalue

vm
: associated eigenvector

lM
: (real positive scalar) maximum eigenvalue

vM
: associated eigenvector

Description

This quick and dirty function computes `(lM,vM)` using the iterative power method and `(lm,vm)` with the inverse iterative power method, then `K2 = lM/lm`. For each method the iterations are stopped until the following condition is met :

```
    | (l_new - l_old)/l_new | < rtol
```

but 4 iterations are nevertheless required and also the iterations are stopped if itermax is reached (and a warning message is issued). As the matrix is symetric this is the rayleigh quotient which gives the estimated eigenvalue at each step (`lambda = v'*A*v`). You may called this function with named parameter, for instance if you want to see the intermediary result without setting yourself the rtol and itermax parameters you may called this function with the syntax :

```
   [K2, lm, vm, lM, vM] = cond2sp(A , C_ptr, verb=%t )
```

Caution

Currently there is no verification for the input parameters !

Remark

This function is intended to get an approximation of the 2-norm condition number (K2) and with the methods used, the precision on the obtained eigenvectors (vM and vm) are generally not very good. If you look for a smaller residual `||Av - l*v||`, you may apply some inverse power iterations from v0 with the matrix :

```
B = A - l0*speye(A)
```

For instance, applied 5 such iterations for `(lm,vm)` is done with :

```
l0 = lm ; v0 = vm;  // or l0 = lM ; v0 = vM;  // to polish (lM,vM)
B = A - l0*speye(A);
LUp = umf_lufact(B);
vr = v0; nstep = 5;
for i=1:nstep, vr = umf_lusolve(LUp, vr, "Ax=b", B); vr = vr/norm(vr) ; end
umf_ludel(LUp); // if you don't use anymore this factorization
lr = vr'*A*vr;
norm_r0 = norm(A*v0 - l0*v0);
norm_rr = norm(A*vr - lr*vr);
// Bauer-Fike error bound...
mprintf(" first estimated eigenvalue : l0 = %e \n\t", l0)
mprintf(" |l-l0| <= ||Av0-l0v0|| = %e , |l-l0|/l0 <= %e \n\r", norm_r0, norm_r0
mprintf(" raffined estimated eigenvalue : lr = %e \n\t", lr)
mprintf(" |l-lr| <= ||Avr-lrvr|| = %e , |l-lr|/lr <= %e \n\r", norm_rr, norm_rr
```

Examples

```
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa");
C_ptr = taucs_chfact(A);
[K2, lm, vm, lM, vM] = cond2sp(A , C_ptr, 1.e-5, 50, %t );
taucs_chdel(C_ptr)
```

See Also

condestsp , taucs_chfact , rcond

Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

Name

condestsp — estimate the condition number of a sparse matrix

```
[K1] = condestsp(A, LUp, t)
[K1] = condestsp(A, LUp)
[K1] = condestsp(A, t)
[K1] = condestsp(A)
```

Parameters

A
: a real or complex square sparse matrix

LUp
: (optional) a pointer to (umf) LU factors of A obtained by a call to umf_lufact ; if you have already computed the LU (= PAQ) factors it is recommanded to give this optional parameter (as the factorization may be time consuming)

t
: (optional) a positive integer (default value 2) by increasing this one you may hope to get a better (even exact) estimate

K1
: estimated 1-norm condition number of A

Description

Give an estimate of the 1-norm condition number of the sparse matrix A by Algorithm 2.4 appearing in :

```
 "A block algorithm for matrix 1-norm estimation
  with an application to 1-norm pseudospectra"
  Nicholas J. Higham and Francoise Tisseur
  Siam J. Matrix Anal. Appl., vol 21, No 4, pp 1185-1201
```

Noting the exact condition number `K1e = ||A||_1 ||A^(-1)||_1`, we have allways `K1 <= K1e` and this estimate gives in most case something superior to `1/2 K1e`

Examples

```
A = sparse( [ 2  3  0  0  0;
              3  0  4  0  6;
              0 -1 -3  2  0;
              0  0  1  0  0;
              0  4  2  0  1] );
K1 = condestsp(A)
// verif by direct computation
K1e = norm(A,1)*norm(inv(full(A)),1)

// another example
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/arc130.rua");
K1 = condestsp(A)
// this example is not so big so that we can do the verif
```

```
K1e = norm(A,1)*norm(inv(full(A)),1)

// if you have already the lu factors condestsp(A,Lup) is faster
// because lu factors are then not computed inside condestsp
Lup = umf_lufact(A);
K1 = condestsp(A,Lup)
umf_ludel(Lup)         // clear memory
```

See Also

umf_lufact , rcond

Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

Name

rafiter — (obsolete) iterative refinement for a s.p.d. linear system

```
[xn, rn] = rafiter(A, C_ptr, b, x0, [, nb_iter, verb])
```

Parameters

A
: a real symetric positive definite sparse matrix

C_ptr
: a pointer to a Cholesky factorization (got with taucs_chfact)

b
: column vector (r.h.s of the linear system) but "matrix" (multiple r.h.s.) are allowed.

x0
: first solution obtained with taucs_chsolve(C_ptr, b)

nb_iter
: (optional) number of raffinement iterations (default 2)

verb
: (optional) boolean, must be %t for displaying the intermediary results, and %f (default) if you don't want.

xn
: new refined solution

rn
: residual (`A*xn - b`)

Description

This function is somewhat obsolete, use `x = taucs_chsolve(C_ptr,b,A)` (see taucs_chsolve) which do one iterative refinement step.

To use if you want to improve a little the solution got with taucs_chsolve. Note that with verb=%t the displayed internal steps are essentially meaningful in the case where b is a column vector.

Caution

Currently there is no verification for the input parameters !

Examples

```
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa");
C_ptr = taucs_chfact(A);
b = rand(size(A,1),1);
x0 = taucs_chsolve(C_ptr, b);
norm(A*x0 - b)
[xn, rn] = rafiter(A, C_ptr, b, x0, verb=%t);
norm(A*xn - b)
taucs_chdel(C_ptr)
```

See Also

taucs_chsolve , taucs_chfact

Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

Name

res_with_prec — computes the residual r = Ax-b with precision

```
[r,norm2_r] = res_with_prec(A, x, b)
```

Parameters

A
: real or complex sparse matrix (m x n)

x
: column vector (n x 1) or matrix (n x p)

b
: column vector (m x 1) or matrix (m x p)

r
: column vector (m x 1) or matrix (m x p)

norm2_r
: scalar or vector (1 x p) when b is a m x p matrix

Description

This function computes the residual of a linear system `r = Ax - b` (together with its 2-norm) with the additionnal precision provided on "Intel like" FPU (80 bits in place of 64) if the compiler translate "long double" to use it. Else one must get the same than using `A*x - b` at the scilab level. In both cases using `[r, nr] = res_with_prec(A,x,b)` is faster than `r = A*x - b` (and faster than `r = A*x - b; nr = norm(r)`).

When `p > 1`, `norm2_r(i)` is the 2-norm of the vector `r(:,i)`.

Examples

```
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa");
C_ptr = taucs_chfact(A);
b = rand(size(A,1),1);
x0 = taucs_chsolve(C_ptr, b);
norm(A*x0 - b)
norm(res_with_prec(A, x0, b))
```

See Also

rafiter

Authors

Bruno Pincon <Bruno.Pincon@iecn.u-nancy.fr>

Name

taucs_chdel — utility function used with taucs_chfact

```
taucs_chdel(C_ptr) or taucs_chdel()
```

Parameters

C_ptr
: a pointer to a Cholesky factorization

Description

This function is used in conjunction with taucs_chfact and taucs_chsolve. It clears the internal memory space used to store the Cholesky factorization (got with taucs_chfact). Use without argument it frees the memory for all the current scilab (taucs) Cholesky factorizations.

Examples

see the example section of taucs_chfact

See Also

taucs_chfact , taucs_chsolve , taucs_chinfo , taucs_chget

Authors

taucs by Sivan Toledo (see taucs_license)
scilab interface by Bruno Pincon

Name

taucs_chfact — cholesky factorisation of a sparse s.p.d. matrix

```
C_ptr = taucs_chfact(A)
```

Parameters

A
: a sparse real symetric positive definite (s.p.d.) matrix

C_ptr
: a pointer to the Cholesky factors (C,p : A(p,p)=CC')

Description

This function computes a Cholesky factorization of the sparse symetric positive definite (s.p.d.) matrix A and retrieves at the scilab level, a pointer (C_ptr) to an handle of the Cholesky factors (C,p) (the memory used for them is "outside" scilab space).

If your matrix is s.p.d. this function must be used in place of umf_lufact or in place of the scilab function chfact for a gain in speed (also as chfact uses the scilab memory for the factors the user must set the stacksize with a large value because of the fill-in occuring in computing the factor C which then may take more memory than the initial matrix A).

When such a factorisation have been computed, a linear system must be solved with taucs_chsolve. **To free the memory used by the Cholesky factors, use taucs_chdel(C_ptr)**; to retrieve the Cholesky factors at the scilab level (for example to display their sparse patterns), use taucs_chget; to get some information (number of non zeros in C), use taucs_chinfo. To compute an approximation of the condition number in norm 2 use cond2sp.

Remarks

- taucs_chfact works only with the upper triangle of A, and the matrix A must be provided either in its complete form (that is with the lower triangle also) or only with its upper triangle;
- currently taucs_chfact uses the genmmd (generalized minimum degree) algorithm of Liu to find in a first step the permutation p (so as to minimize the fill-in in the factorization); future versions will let the user choose his/her own reordering by providing a supplementary argument p.

Examples

```
// Example #1 : a small linear test system
// whom solution must be [1;2;3;4;5]
A = sparse( [ 2 -1  0  0  0;
             -1  2 -1  0  0;
              0 -1  2 -1  0;
              0  0 -1  2 -1;
              0  0  0 -1  2] );
b = [0 ; 0; 0; 0; 6];
Cp = taucs_chfact(A);
x = taucs_chsolve(Cp,b)
// don't forget to clear memory with
taucs_chdel(Cp)

// Example #2 a real example
```

```
// first load a sparse matrix
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa");
// compute the factorisation
Cp = taucs_chfact(A);
b = rand(size(A,1),1); // a random rhs
// use taucs_chsolve for solving Ax=b
x = taucs_chsolve(Cp,b);
norm(A*x - b)
// the same with one iterative refinement step
x = taucs_chsolve(Cp,b,A);
norm(A*x - b)
// don't forget to clear memory
taucs_chdel(Cp)
```

See Also

taucs_chsolve , taucs_chdel , taucs_chinfo , taucs_chget , cond2sp

Authors

taucs by Sivan Toledo (see taucs_license)
scilab interface by Bruno Pincon

Name

taucs_chget — retrieve the Cholesky factorization at the scilab level

```
[Ct,p] = taucs_chget(C_ptr)
```

Parameters

C_ptr
: a pointer to the Cholesky factorization (C,p : A(p,p)=CC')

Ct
: a scilab sparse matrix (you get the upper triangle i.e. Ct is equal to C')

p
: column vector storing the permutation

Description

This function may be used if you want to plot the sparse pattern of the Cholesky factorization (or if you code something which use the factors). Traditionnaly, the factorization is written :

```
   P A P' = C C'
```

with P' the permutation matrix associated to the permutation p. As we get the upper triangle Ct (= C'), in scilab syntax we can write :

```
   A(p,p) = Ct' * Ct
```

Examples

```
// Example #1 : a small linear test system
A = sparse( [ 2 -1  0  0  0;
             -1  2 -1  0  0;
              0 -1  2 -1  0;
              0  0 -1  2 -1;
              0  0  0 -1  2] );
Cp = taucs_chfact(A);
[Ct, p] = taucs_chget(Cp);
full(A(p,p) - Ct'*Ct)  // this must be near the null matrix
taucs_chdel(Cp)

// Example #2 a real example
stacksize(3000000)  // the last PlotSparse need memory
// first load a sparse matrix
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/bcsstk24.rsa");
// compute the factorisation
Cptr = taucs_chfact(A);
// retrieve the factor at scilab level
[Ct, p] = taucs_chget(Cptr);
// plot the initial matrix
xset("window",0) ; xbasc()
```

```
PlotSparse(A) ; xtitle("Initial matrix A (bcsstk24.rsa)")
// plot the permuted matrix
B = A(p,p);
xset("window",1) ; xbasc()
PlotSparse(B) ; xtitle("Permuted matrix B = A(p,p)")
// plot the upper triangle Ct
xset("window",2) ; xbasc()
PlotSparse(Ct) ; xtitle("The pattern of Ct (A(p,p) = C*Ct)")
// retrieve cnz
[OK, n, cnz] = taucs_chinfo(Cptr)
// cnz is superior to the realnumber of non zeros elements of C :
cnz_exact = nnz(Ct)
// don't forget to clear memory
taucs_chdel(Cptr)
```

See Also

taucs_chfact , taucs_chsolve , taucs_chdel , taucs_chinfo , taucs_chget , cond2sp

Authors

taucs by Sivan Toledo (see taucs_license)
scilab interface by Bruno Pincon

Name

taucs_chinfo — get information on Cholesky factors

```
[OK, n, cnz] = taucs_chinfo(C_ptr)
```

Parameters

C_ptr
: a pointer to a Cholesky factorization

OK
: a scalar boolean

n
: a scalar integer

cnz
: a scalar integer

Description

This function may be used to know basic information about the Cholesky factor created with taucs_chfact :

- first `OK` is `%t` if `C_ptr` is a valid pointer to an Cholesky factorization (and `%f` else)
- if `OK` is `%t` then `n` and `cnz` are respectively the matrix order and the number of non zeros elements in the supernodal structure storing `C` ; if `OK` is `%f`, `n` and `cnz` are set to the void matrix [].

Details

Due to the supernodal structure used for `C`, `cnz` is larger than the exact number of non-zeros elements in `C` (and so this `cnz` is a mesure of the memory used internally). To get the exact `cnz` you may retrieve the Cholesky factor with taucs_chget then apply the nnz scilab function (see the 2d example in taucs_chget).

See Also

taucs_chfact , taucs_chsolve , taucs_chdel , taucs_chget

Authors

taucs by Sivan Toledo (see taucs_license)
scilab interface by Bruno Pincon

Name

taucs_chsolve — solve a linear sparse (s.p.d.) system given the Cholesky factors

```
[x] = taucs_chsolve(C_ptr, b [, A])
```

Parameters

C_ptr
: a pointer to a handle of the Cholesky factors (C,p with A(p,p)=CC')

b
: a real column vector or a matrix (multiple rhs)

x
: a real column vector or a matrix in case of multiple rhs (x(:,i) is solution of A x(:,i) = b(:,i))

A
: (optional) the real s.p.d. matrix A (to use for iterative refinement step)

Description

This function must be used in conjonction with taucs_chfact which computes the Cholesky factorization of a sparse real s.p.d. matrix. When the matrix A is provided, one iterative refinement step is done (the refined solution is accepted if it improves the 2-norm of the residual Ax-b).

Like in taucs_chfact the matrix A may be provided either in its complete form (that is with the lower triangle also) or only with its upper triangle.

Examples

see the example section of taucs_chfact

See Also

taucs_chfact , taucs_chdel , taucs_chinfo , taucs_chget , cond2sp

Authors

taucs by Sivan Toledo (see taucs_license)
scilab interface by Bruno Pincon

Name

taucs_license — display the taucs license

Copyright

TAUCS License

Availability

http://www.tau.ac.il/~stoledo/taucs/

Name

umf_license — display the umfpack license

Copyright

UMFPACK License

Availability

http://www.cise.ufl.edu/research/sparse

Name

umf_ludel — utility function used with umf_lufact

```
umf_ludel(LU_ptr) or umf_ludel()
```

Parameters

LU_ptr
: a pointer to an handle of umf lu factors (L,U,p,q,R)

Description

This function must be used in conjunction with umf_lufact and umf_lusolve. It clears the internal memory space used to store the LU factors (got with umf_lufact). Use without argument it frees the memory for all the current scilab umfpack LU factors.

Examples

see the example section of umf_lufact

See Also

umfpack , umf_lufact , umf_lusolve , umf_luget , umf_luinfo

Authors

umfpack by Timothy A. Davis (see umf_license)
scilab interface by Bruno Pincon

Name

umf_lufact — lu factorisation of a sparse matrix

```
LU_ptr = umf_lufact(A)
```

Parameters

A
: a sparse, real or complex, square or rectangular, matrix

LU_ptr
: a pointer to umf lu factors (L,U,p,q,R)

Description

This function computes a LU factorisation of the sparse matrix A () and return at the scilab level, a pointer (LU_ptr) to an handle of the LU factors (L,U,p,q,R) (the memory used for them is "outside" scilab stack).

This function must be used in place of umfpack if you have multiple linear systems with the same matrix to solve when the rhs are not known at the same time (for instance A x1 = b1 and A x2 = b2 but b2 depends on x1, etc...).

When such a factorisation have been computed, a linear system must be solved with umf_lusolve (in general x = umf_lusolve(LU_ptr, b) but others options are possible, see umf_lusolve. **To free the memory used by the LU factors, use umf_ludel(LU_ptr) (umf_ludel);** to retrieve the LU factors at the scilab level (for example to display their sparse patterns), use umf_luget; to get some information (number of non zeros in L and U), use umf_luinfo. To compute an approximation of the condition number use condestsp

Examples

```
// this is the small linear test system from UMFPACK
// whom solution must be [1;2;3;4;5]
A = sparse( [ 2  3  0  0  0;
              3  0  4  0  6;
              0 -1 -3  2  0;
              0  0  1  0  0;
              0  4  2  0  1] );
b = [8 ; 45; -3; 3; 19];
Lup = umf_lufact(A);
x = umf_lusolve(Lup,b)

// solve now A'x=b
x = umf_lusolve(Lup,b,"A''x=b")
norm(A'*x - b)

// don't forget to clear memory with
umf_ludel(Lup)

// a real (but small)  example
// first load a sparse matrix
[A] = ReadHBSparse(SCI+"/modules/umfpack/examples/arc130.rua");
// compute the factorisation
Lup = umf_lufact(A);
```

```
b = rand(size(A,1),1); // a random rhs
// use umf_lusolve for solving Ax=b
x = umf_lusolve(Lup,b);
norm(A*x - b)

// now the same thing with iterative refiment
x = umf_lusolve(Lup,b,"Ax=b",A);
norm(A*x - b)

// solve now the system A'x=b
x = umf_lusolve(Lup,b,"A''x=b");  // without refinement
norm(A'*x - b)
x = umf_lusolve(Lup,b,"A''x=b",A);  // with refinement
norm(A'*x - b)

// don't forget to clear memory
umf_ludel(Lup)
```

See Also

umfpack , umf_luget , umf_lusolve , umf_ludel , umf_luinfo , condestsp

Authors

umfpack by Timothy A. Davis (see umf_license)
scilab interface by Bruno Pincon with contributions from Antonio Frasson

Name

umf_luget — retrieve lu factors at the scilab level

```
[L,U,p,q,Rd] = umf_luget(LU_ptr)
```

Parameters

LU_ptr
: a pointer to umf lu factors (L,U,p,q,R)

L,U
: scilab sparse matrix

p,q
: column vectors storing the permutations

Rd
: vector storing the (row) scaling factors

Description

This function may be used if you want to plot the sparse pattern of the lu factors (or if you code something which use the lu factors). The factorization provided by umfpack is of the form:

P R^(-1) A Q = LU

where P and Q are permutation matrices, R is a diagonal matrix (row scaling), L a lower triangular matrix with a diagonal of 1, and U an upper triangular matrix. The function provides the matrices L and U as Sparse scilab matrices but P and Q are given as permutation vectors p and q (in fact p is the permutation associated to P') and Rd is the vector corresponding to the diagonal of R.

Examples

```
// this is the test matrix from UMFPACK
A = sparse( [ 2  3  0  0  0;
              3  0  4  0  6;
              0 -1 -3  2  0;
              0  0  1  0  0;
              0  4  2  0  1] );
Lup = umf_lufact(A);
[L,U,p,q,R] = umf_luget(Lup);
B = A;
for i=1:5, B(i,:) = B(i,:)/R(i); end // apply the row scaling
B(p,q) - L*U  // must be a (quasi) nul matrix

umf_ludel(Lup) // clear memory

// the same with a complex matrix
A = sparse( [ 2+%i  3+2*%i  0      0    0;
              3-%i  0       4+%i   0    6-3*%i;
              0    -1+%i   -3+6*%i 2-%i 0;
              0     0       1-5*%i 0    0;
              0     4       2-%i   0    1] );
Lup = umf_lufact(A);
[L,U,p,q,R] = umf_luget(Lup);
```

```
B = A;
for i=1:5, B(i,:) = B(i,:)/R(i); end // apply the row scaling
B(p,q) - L*U  // must be a (quasi) nul matrix

umf_ludel(Lup) // clear memory
```

See Also

umfpack , umf_lufact , umf_lusolve , umf_ludel , umf_luinfo

Authors

umfpack by Timothy A. Davis (see umf_license)
scilab interface by Bruno Pincon

Name

umf_luinfo — get information on LU factors

```
[OK, nrow, ncol, lnz, unz, udiag_nz, it] = umf_luinfo(LU_ptr)
```

Parameters

LU_ptr
: a pointer to umf lu factors (L,U,p,q, R)

OK
: a scalar boolean

nrow, ncol, lnz, unz, udiag_nz, it
: scalars (integers)

Description

This function may be used to know basic information about LU factors created with umf_lufact :

first `OK` is %t if `LU_ptr` is a valid pointer to an umfpack LU numeric handle (and %f else)

if `OK` is %t then:

nrow, ncol
: are the matrix size (L is `nrow x n` and U is `n x ncol` where `n = min(nrow,ncol)`

lnz, unz
: are the number of non zeros elements in L and in U;

udiag_nz
: are the number of non zeros elements on the diagonal of U; if the matrix is square (`nrow = ncol = n`) then it is not inversible if udiag_nz < n (more precisely it appears to be numericaly not inversible through the LU factorization).

it
: 0 if the factors are real and 1 if they are complex.

if OK is %f then all the others outputs are set to the empty matrix [].

Examples

```
// this is the test matrix from UMFPACK
A = sparse( [ 2  3  0  0  0;
              3  0  4  0  6;
              0 -1 -3  2  0;
              0  0  1  0  0;
              0  4  2  0  1] );
Lup = umf_lufact(A);
[OK, nrow, ncol, lnz, unz, udiag_nz, it] = umf_luinfo(Lup)  // OK must be %t, n
[L,U,p,q,R] = umf_luget(Lup);
nnz(L)  // must be equal to lnz
nnz(U)  // must be equal to unz
umf_ludel(Lup) // clear memory
```

See Also

umfpack , umf_lufact , umf_lusolve , umf_ludel , umf_luget

Authors

umfpack by Timothy A. Davis (see umf_license)
scilab interface by Bruno Pincon

Name

umf_lusolve — solve a linear sparse system given the LU factors

```
[x] = umf_lusolve(LU_ptr, b [, st, A])
```

Parameters

LU_ptr
: a pointer to umf lu factors (L,U,p,q,R)

b
: a real or complex column vector or a matrix (multiple rhs)

st
: (optional) a string "Ax=b" (default) or "Ax'=b" (to be written "Ax"=b" in scilab langage: a quote in a string must be doubled !)

A
: (optional) the sparse square matrix corresponding to the LU factors (LU_ptr must be got with LU_ptr = umf_lufact(A))

x
: a column vector or a matrix in case of multiple rhs (x(:,i) is solution of A x(:,i) = b(:,i) or A'x(:,i) = b(:,i))

Description

This function must be used in conjonction with umf_lufact which computes the LU factors of a sparse matrix. The optional `st` argument lets us choose between the solving of Ax=b (general case) or of A'x=b (sometimes useful). If you give the 4th argument then iterative refinement will be also proceceed (as in umfpack) to give a better numerical solution.

Examples

see the example section of umf_lufact

See Also

umfpack , umf_lufact , umf_luget , umf_ludel , umf_luinfo

Authors

umfpack by Timothy A. Davis (see umf_license)
scilab interface by Bruno Pincon with contributions from Antonio Frasson

Name

umfpack — solve sparse linear system

```
x = umfpack(A,"\",b)
x = umfpack(b,"/",A)
```

Parameters

A
: a sparse (real or complex) square matrix n x n

b
: in the first case, a column vector (n x 1) or a n x m matrix ; in the second case, a row vector (1 x n) or a m x n matrix

x
: in the first case , a column vector (n x 1) or a n x m matrix ; in the second case, a row vector (1 x n) or a m x n matrix

2d arg
: string specifier "\" or "/"

Description

This function is intended to work like the classic operators \ and / x = A\b and x = b/A) i.e. it solves a linear system Ax = b or xA = b with a sparse square (says n x n) real or complex matrix and with a compatible rhs b : n x m in the first case and m x n in the second.

Details

First an LU factorisation of the matrix is computed (`P R^(-1) A Q = LU` where P and Q are permutation matrices, R is a diagonal matrix (row scaling), L a lower triangular matrix with a diagonal of 1, and U an upper triangular matrix) then a first solution is computed with forward/backward subtitutions ; finaly the solution is improved by iterative refinement.

Examples

```
// this is the small linear test system from UMFPACK
// whom solution must be [1;2;3;4;5]
A = sparse( [ 2  3  0  0  0;
              3  0  4  0  6;
              0 -1 -3  2  0;
              0  0  1  0  0;
              0  4  2  0  1] );
b = [8 ; 45; -3; 3; 19];
x = umfpack(A,"\",b)

// test the other form x A = b
b = [8  20  13  6  17];
x = umfpack(b,"/",A)   // solution must be [1 2 3 4 5]

// test multiple rhs
b = rand(5,3);
x = umfpack(A,"\",b)
norm(A*x - b)
```

```
// test multiple rhs for x A = b
b = rand(3,5);
x = umfpack(b,"/",A)
norm(x*A - b)

// solve a complex system
A = sparse( [ 2+%i  3+2*%i  0      0    0;
              3-%i  0       4+%i   0    6-3*%i;
              0    -1+%i   -3+6*%i 2-%i 0;
              0     0       1-5*%i 0    0;
              0     4       2-%i   0    1] );
b = [ 3+13*%i ; 58+32*%i ; -19+13*%i ; 18-12*%i ; 22+16*%i ];
x = umfpack(A,"\",b)  // x must be [1+i; 2+2i; 3+3i; 4 + 4i; 5+5i]
```

See Also

umf_lufact , umf_lusolve , umf_ludel , umf_luinfo , umf_luget

Authors

umfpack by Timothy A. Davis (see umf_license)
scilab interface by Bruno Pincon with contributions from Antonio Frasson