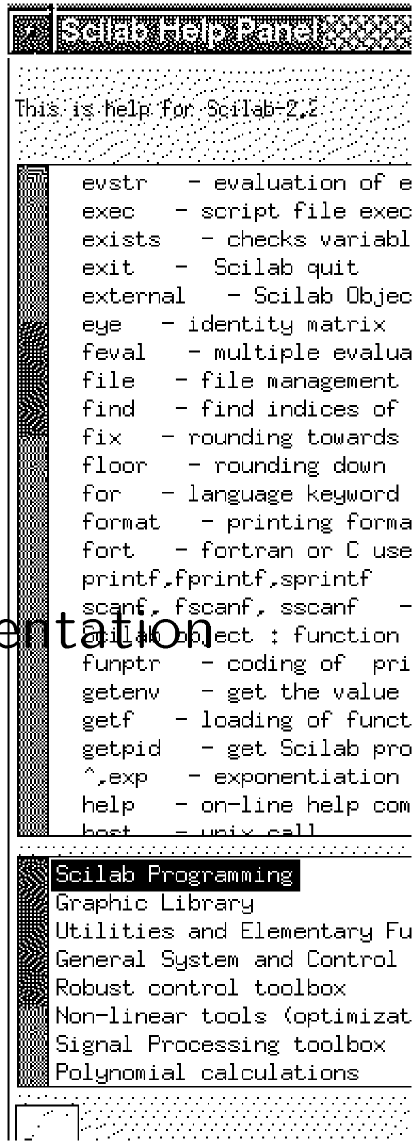
Scilab

Reference

Manual



On-line Documentation

Scilab Group

SCILAB REFERENCE MANUAL

Scilab Group
INRIA Meta2 Project/ENPC Cergrene

INRIA - Unité de recherche de Rocquencourt - Projet Meta2
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Email: Scilab@inria.fr

Chapter 1

Programming

1.0.1 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` 85, `pause` 79, `break` 9, `abort` 6, `quit` 85

1.0.2 addinter _____ new functions interface incremental linking at run time

CALLING SEQUENCE :

```
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 linking of a compiled C or Fortran new Scilab interface routine (see `intersci` documentation) and define corresponding scilab functions.
For machines using dlopen functionality one can unlink an interface with `ulink` (use the command `link('show')` to get the number of the shared library). 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.

SEE ALSO : `link` 54, `intersci` 49, `newfun` 73, `clearfun` 12

1.0.3 and _____ - logical and

CALLING SEQUENCE :

```
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` 74, `or` 76

1.0.4 ans _____ answer

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

1.0.5 apropos _____ searches keywords in Scilab help

CALLING SEQUENCE :

```
apropos word
apropos 'string'
```

DESCRIPTION :
Looks for keywords in `man/*/whatis` files.

EXAMPLE :

```
apropos '+'
apropos ode
apropos 'list of'
```

SEE ALSO : `help` 41

1.0.6 argn _____ number of arguments in a function call

CALLING SEQUENCE :

```
[lhs [,rhs] ]=argn(0)
```

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

SEE ALSO : `function` 36

1.0.7 backslash _____ - left matrix division.

CALLING SEQUENCE :

```
x=A\\b
```

DESCRIPTION :
Backslash denotes left matrix division. `x=A\b` is a solution to `A*x=b`.
If `A` is nonsingular `x=A\b` (uniquely defined) is equivalent to `x=inv(A)*b`.
If `A` is singular, `x` is a least square solution. i.e. `norm(A*x-b)` is minimal. 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 /.

EXAMPLE :


```
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` 97, `inv` 439, `pinv` 449, `percent` 80, `ieee` 44

1.0.8 bool2s _____ convert boolean matrix to a zero one matrix.

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
bool2s([%t %t %f %t])
bool2s([2.3 0 10 -1])
```

SEE ALSO : `boolean` 8, `find` 29

1.0.9 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.

EXAMPLE :

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

SEE ALSO : `matrices` 58, `or` 76, `and` 6, `not` 74

1.0.10 brackets - left and right brackets

CALLING SEQUENCE :

```
[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", semicolumn 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` 15, `semicolumn` 95

1.0.11 break keyword to interrupt loops

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

EXAMPLE :

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

SEE ALSO : `while` 109, `if` 45, `for` 30, `abort` 6, `return` 90

1.0.12 call Fortran or C user routines call

CALLING SEQUENCE :

```
// 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`).
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=call("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=call("fooc",a,2,"d",b,3,"d",m,4,"i",n,5,"i","out",[m,n],1,"d");
```

SEE ALSO : `link` 54, `c_link` 193, `intersci` 49, `addinter` 6

1.0.13 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` 94, `while` 109, `end` 19, `for` 30

1.0.14 ceil ______ rounding up

CALLING SEQUENCE :

```
[y]=ceil(x)
```

PARAMETERS :

`x` : a real matrix
`y` : integer matrix

DESCRIPTION :
`ceil(x)` returns an integer matrix made of rounded up elements

EXAMPLE :

```
ceil([1.9 -2.5])-[2,-2]
ceil(-%inf)
x=rand()*10^20;ceil(x)-x
```

SEE ALSO : `round` 90, `floor` 30, `int` 49

1.0.15 chdir ______ changes Scilab current directory

CALLING SEQUENCE :

```
ierr=chdir('path-name')
```

PARAMETERS :

`ierr` : an integer, 1 if `chdir` failed to change directory and `0` elsewhere.

DESCRIPTION :
Change the current Scilab directory to `'path-name'`

EXAMPLE :

```
chdir(TMPDIR);
if MSDOS then
  unix_w("dir");
else
  unix_w("ls");
end
```

SEE ALSO : `getcwd` 37

1.0.16 clearfun ______ remove primitive.

CALLING SEQUENCE :

```
clearfun('name')
```

DESCRIPTION :
`clearfun('name')` removes the primitive `'name'` from the set of primitives (built-in functions) . This function allows to rename a primitive : a Scilab primitive can be replaced by a user-defined function. For experts...

SEE ALSO : `newfun` 73, `funptr` 36

1.0.17 clearglobal ______ kills global variables

CALLING SEQUENCE :

```
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 variables pointing on these global variables are not destroyed.

SEE ALSO : `global` 40, `who` 109

EXAMPLE :

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

1.0.18 clear ______ kills variables

CALLING SEQUENCE :

```
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` 82, `who` 109

1.0.19 code2str ______ returns character string associated with Scilab integer codes.

CALLING SEQUENCE :

```
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))

EXAMPLE :

```
code2str([-28 12 18 21 10 11])
```

SEE ALSO : `str2code` 99

1.0.20 coeff ______ coefficients of matrix polynomial

CALLING SEQUENCE :

```
[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` 81, `degree` 16, `inv_coeff` 51

1.0.21 colon ______ - colon operator

DESCRIPTION :

`:` : Colon. Used in subscripts and loops.

`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.

`A(:)` is the vector of all the elements of `A` regarded as a single column.
`A(:,j)` is 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` 59

1.0.22 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 : `semi` 95, `brackets` 9

1.0.23 comments comments

DESCRIPTION :
Command lines which begin by `//` are not interpreted by Scilab.
Comments must not begin with `//end` !

1.0.24 comp scilab function compilation

CALLING SEQUENCE :

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

PARAMETERS :

`function` : an not compiled scilab function (type 11)
`opt` : integer flag with value 0 (default) or 1.

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 command: `getf('filename')` loads the functions in file `'filename'` and compiles them. So comp has to be used in very particular cases.
The opt==1 option is specific to code analysis purpose (see macr2lst)

REMARKS :
commands `who, help, what` cannot be compiled.

SEE ALSO : `deff` 16, `getf` 39, `whereis` 108, `macr2lst` 215, `lib` 53

1.0.25 date Current date as date string

CALLING SEQUENCE :

```
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` 37, `timer` 243

1.0.26 deff on-line definition of function

CALLING SEQUENCE :

```
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)
`'n'` : function is not "compiled"

DESCRIPTION :
On-line definition of function (user defined function): the name of the created function is `newfunction`. `text` is a sequence of instructions usually set as a vector of character strings.
This command can be used inside a function and the new function can be an input or output of any other function.
Usually, functions are defined in a file and loaded into Scilab by `getf`
Some time, in particular when you want to use define strings within `deff text` is rather difficult to write. A more tractable way may be to define your function in a file as usual, to load it into Scilab by `getf` (without `'c'` option) and use sci2exp to get 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` 39, `comp` 15, `exec` 22, `function` 36

1.0.27 degree degree of polynomial matrix

CALLING SEQUENCE :

```
[D]=degree(M)
```

PARAMETERS :

`M` : polynomial matrix
`D` : integer matrix

DESCRIPTION :
returns the matrix of highest degrees of `M`.

SEE ALSO : `poly` 81, `coeff` 14, `clean` 408

1.0.28 delbpt ______ delete breakpoint

CALLING SEQUENCE :

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

DESCRIPTION :
deletes the breakpoint at line `linenumb` in the function `macroname`. If `linenumb` is omitted all the breakpoints in the function are deleted.

EXAMPLE :

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

SEE ALSO : `setbpt` 95, `dispbpt` 17, `pause` 79, `resume` 89

1.0.29 diary ______ diary of session

CALLING SEQUENCE :

```
diary('file-name')
```

DESCRIPTION :
`diary` creates a file which contains a copy of the current Scilab session. `diary(0)` interrupts the diary.

SEE ALSO : `exec` 22, `unix` 104

1.0.30 dispbpt ______ display breakpoints

CALLING SEQUENCE :

```
dispbpt()
```

DESCRIPTION :
`dispbpt()` displays all active breakpoints actually inserted in functions.

SEE ALSO : `setbpt` 95, `delbpt` 17, `pause` 79, `resume` 89

1.0.31 disp ______ displays variables

CALLING SEQUENCE :

```
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.

SEE ALSO : `write` 111, `read` 88, `print` 85, `string` 100, `tlist` 102

EXAMPLES :

```
disp([1 2],3)
deff('[]=%t_p(l)','disp(l(3),l(2))')
disp(tlist('t',1,2))
```


1.0.32 dot ______ - symbol

CALLING SEQUENCE :

```
123.33
a.*b
[123,..
 456]
.SH DESCRIPTION
.TP 6
.
Dot is used to mark decimal point for numbers : 3.25 and 0.001
.TP
.<op>
used in cunjunction 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
.TP
..
Continuation.  Two or more decimal points at the end of
a line causes the following line to be a continuation.
.SH EXAMPLE
.nf
1.345
x=[1 2 3];x.^2 .*x // a space is required between 2 and dot
[123,..
 456]
```

SEE ALSO : `star` 99, `hat` 41, `slash` 97, `backslash` 7

1.0.33 double ______ conversion from integer to double precision representation

CALLING SEQUENCE :

```
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` 48, `inttype` 50, `type` 103

1.0.34 elseif _______________ keyword in if-then-else

DESCRIPTION :
See `if,then,else` .

1.0.35 else _______________ keyword in if-then-else

DESCRIPTION :
Used with `if`.

SEE ALSO : `if` 45

1.0.36 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.

EXAMPLE :

```
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` 58, `poly` 81, `string` 100, `boolean` 8, `rational` 86, `syslin` 241

1.0.37 end _______________ end keyword

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

SEE ALSO : `for` 30, `while` 109, `if` 45, `select` 94

1.0.38 equal _______________ - affectation, comparison equal sign

DESCRIPTION :
Equal sign is used to denote a value affectation to a variable.
== denote equality comparison between two expressions and returns a boolean matrix.

EXAMPLES :

```
a=sin(3.2)
[u,s]=schur(rand(3,3))
[1:10]==4
1~=2
```

SEE ALSO : `less` 53, `boolean` 8

1.0.39 errcatch _______________ error trapping

CALLING SEQUENCE :

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

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 followin 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.
`"continue"` : next instruction in the function or exec files is executed, current instruction is ignored. This option is useful for error recovery.
`"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.

SEE ALSO : `errclear` 20, `iserror` 51

1.0.40 errclear _______________ error clearing

CALLING SEQUENCE :

```
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` 20

1.0.41 error ________ error messages

CALLING SEQUENCE :

```
error('string' [,n])
error(m)
```

DESCRIPTION :

prints the character string `'string'` in an error message and stops the current instruction.

If `n` is given, it is associated to the number of the error. `n` should be larger than `10000` (default value).

`error(m)` prints the message associated with the error number `m`.

SEE ALSO : `warning` 108

1.0.42 evstr ________ evaluation of expressions

CALLING SEQUENCE :

```
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 evaluation of the matrix of character strings. Each element of the `M` matrix must be a character string defining a scilab expression.

If evaluation of M expression leads to an error `H=evstr(M)` produces an error which is handled as usual. `[H,ierr]=evstr(M)` produces an error message and return the error number in `ierr`.

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

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

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` 23

1.0.43 exec ________ script file execution

CALLING SEQUENCE :

```
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
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 an error message, aborts execution of the instructions and resume with `ierr` equal to the error number,if 'errcatch' flag is not present, standard error handling works.

REMARK :

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

EXAMPLES :

```
// create a script file
write(TMPDIR+'/myscript','a=1;b=2')
// execute it
exec(TMPDIR+'/myscript')
who

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

x=1 //create x to make it known by the script foo
```

```
exec(foo)
// a and y are created in the current environment
who
```

SEE ALSO : getf 39, execstr 23, evstr 21, comp 15, mode 67, chdir 12, getcwd 37

1.0.44 execstr _ scilab instructions execution by evaluation of strings

CALLING SEQUENCE :

```
execstr(instr)
ierr=execstr(instr,'errcatch')
```

PARAMETERS :

`instr` : vector of character strings, scilab instruction to be executed.
`ierr` : integer, 0 or error number

DESCRIPTION :
executes the Scilab instructions given in argument `instr` If an error is encountered while executing instructions defined in `instr`, if 'errcatch' flag is present `execstr` issues an error message, abort execution of the `instr` instructions and resume with `ierr` equal to the error number,if 'errcatch' flag is not present, standard error handling works.
EXAMPLE :

```
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')
```

SEE ALSO : evstr 21

1.0.45 exists _ checks variable existence

CALLING SEQUENCE :

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

PARAMETERS :

`name` : a character string
`where` : an optional character string with default value `'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 local environment of the current function and `0` otherwise.
EXAMPLE :


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

SEE ALSO : isdef 211, whereis 108, type 103, typeof 246, macrovar 216

1.0.46 exit _ Ends the current Scilab session

DESCRIPTION :
Ends the current Scilab session.
SEE ALSO : quit 85, abort 6, break 9, return 90, resume 89

1.0.47 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 which is not useful here).
If other values are needed by the external function these variables can be defined in the 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 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 routines can also be dynamically linked (see `link`)
SEE ALSO : ode 350, optim 355, impl 338, dassl 333, intg 342, schur 456, gschur 436

1.0.48 extraction _ matrix and list entry extraction

CALLING SEQUENCE :

```
x(i,j)
x(i)
[...]=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` : indices
`k1,...kn` : indices

DESCRIPTION :

`MATRIX CASE` `i` and `j`, can be:
- real scalars or vectors or matrices with positive elements.
* `r=x(i,j)` designs 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 designs 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 designs 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 designs the column vector `r` such as `r(l)` (`l` from 1 to `size(i,'*')`) designs 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,:)` designs 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)` designs 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(:)` designs the column vector `r` formed by the column concatenations of `x` columns. It is equivalent to `matrix(x,size(x,'*'),1)`.
- 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.

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

- 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` variable 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.

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.

EXAMPLE :

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

```
// 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` 29, `horner` 414, `parents` 78

1.0.49 eye ______ identity matrix

CALLING SEQUENCE :

```
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 mtrix 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` 75, `zeros` 252

1.0.50 feval ______ multiple evaluation

CALLING SEQUENCE :

```
[z]=feval(x,y,f)
[z]=feval(x,f)
```

PARAMETERS :

`x,y` : two vectors
`f` : function or character string (for Fortran 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, z(i,j)=f(x(i),y(j))`

`f` is an external (function or routine) depending 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`)

EXAMPLE :


```
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` 21, `horner` 414, `execstr` 23, `external` 24, `link` 54

1.0.51 file ______ file management

CALLING SEQUENCE :

```
[unit [,err]]=file('open', file-name [,status] [,access [,recl]] [,format])
file(action,unit)
[units [,typ [,nams]]] =file()
```

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.
`typs` : Character string vector, type (C or Fortran) of opened files.
`nams` : Character string vector, pathnames of opened 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.

EXAMPLE :

```
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)
//
file('close',file() ) //closes all opened files (C or Fortran type).
//
[units,typs,nams]=file()
```

SEE ALSO : `save` 91, `load` 57, `write` 111, `read` 88, `writb` 110, `readb` 88, `xgetfile` 249

1.0.52 find ___ find indices of boolean vector or matrix true elements

CALLING SEQUENCE :

```
[ii]=find(x)
[ir,ic]=find(x)
```

PARAMETERS :

`x` : a boolean vector or a boolean matrix or a "standard" matrix
`ii, ir, ic` : 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.
`[ir,ic]=find(x)` returns two vectors of indices `ir` (for rows) and `ic` (for columns) such that `x(il(n),ic(n))` is "true". If no true element found find returns empty matrices in `ir` and `ic`.
if `x` is standard matrix `find(x)` is interpreted as `find(x<>0)`
`find([])` returns `[]`

EXAMPLE :

```
A=rand(1,20);
w=find(A<0.5);
A(w)
w=find(A>100);
```

SEE ALSO : `boolean` 8, `extraction` 24, `insertion` 46

1.0.53 fix ___ rounding towards zero

CALLING SEQUENCE :

```
[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` 90, `floor` 30, `ceil` 12

1.0.54 floor ___ rounding down

CALLING SEQUENCE :

```
[y]=floor(x)
```

PARAMETERS :

`x` : a real matrix
`y` : integer matrix

DESCRIPTION :
`floor(x)` returns an integer matrix made of nearest rounded down integers.

EXAMPLE :

```
floor([1.9 -2.5])-[1,-3]
floor(-%inf)
x=rand()*10^20;floor(x)-x
```

SEE ALSO : `round` 90, `fix` 29, `ceil` 12

1.0.55 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.
Useful example : `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 conditionnal instruction (if while for or select/case) must be limited to 16k.

EXAMPLE :

```
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  e=eye(3,3),e,end
for v=a, write(6,v),end
for j=1:n,v=a(:,j), write(6,v),end
for l=list(1,2,'example'); l,end
```

1.0.56 format _____ number printing and display format

CALLING SEQUENCE :

```
format([type],[long])
format()
```

PARAMETERS :

`type` : character string
`long` : integer (max number of digits (default 10))

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 (0 if `v` ; 1 if `e`); second component is the number of digits.
In "variable format" mode, vectors entries which are less than %eps times the maximum absolute value of the entries are displayed as "0".

EXAMPLE :

```
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()
```

SEE ALSO : `write` 111, `disp` 17, `print` 85

1.0.57 fort _____ Fortran or C user routines call

CALLING SEQUENCE :

```
// 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 (# 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` 54, `c_link` 193, `intersci` 49, `addinter` 6

1.0.58 fprintf ________ Emulator of C language fprintf function

CALLING SEQUENCE :

```
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`)

DESCRIPTION :
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 `values` 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
```

SEE ALSO : `string` 100, `print` 85, `write` 111, `format` 31, `disp` 17, `file` 28, `printf` 84, `sprintf` 97

1.0.59 fprintfMat ________ print a matrix in a file.

CALLING SEQUENCE :

```
fprintfMat(fil,M,format)
```

PARAMETERS :

`fil` : a string, path of the file
`format` : a character string, a C like format.
`M` : A matrix of real numbers.

DESCRIPTION :

The fprintfMat function prints a matrix in a formated file. Each row of the matrix give a line in the file.
EXAMPLE :

```
n=50;
a=rand(n,n,'u');
fprintfMat(TMPDIR+'/Mat',a,'%5.2f');
a1=fscanfMat(TMPDIR+'/Mat');
```

SEE ALSO : `mclose` 61, `meof` 61, `mfprintf` 68, `mfscanf` 70, `fscanfMat` 35, `mget` 64, ?? mgetstr, mopen, mprintf, mput, mputstr, mscanf, mseek, mtell

1.0.60 fscanf ________ Converts formatted input read on a file

CALLING SEQUENCE :

```
 [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` 84, `read` 88, `scanf` 92, `sscanf` 98

1.0.61 fscanfMat ____ Reads a Matrix from a text file.

CALLING SEQUENCE :

```
M=fscanfMat(filename);
```

PARAMETERS :

`filename` : a character string giving the name of the file to be scanned.
`M` : Output variable. A matrix of real numbers.

DESCRIPTION :
The `fscanfMat` function is used to read a scalar matrix from a text file. The first non-numeric lines of the file are ignored 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`.

EXAMPLE :

```
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` 61, `meof` 61, `mfprintf` 68, `fprintfMat` 34, `mfscanf` 70, `fscanfMat` 35, `mget` 64, `mgetstr` 65, `mopen` 68, `mprintf` 68, `mput` 69, `mputstr` 70, `mscanf` 70, `mseek` 71, `mtell` 72

1.0.62 funcprot ____ switch scilab functions protection mode

CALLING SEQUENCE :

```
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

EXAMPLE :

```
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
```

1.0.63 function ____ Scilab procedure and Scilab object

DESCRIPTION :
Functions 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 (see `lib`).
They can also be defined on-line (see `deff`).
In a file a function must begin by a "syntax definition" line as follows:

```
function [y1,...,yn]=foo(x1,...,xm)
```

followed by 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, the following `function` keyword or the `EOF` mark.
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.
For example if the file myfct.sci contains:

```
function [x,y]=myfct(a,b)
x=a+b
y=a-b
```

you can load and use it in the following way:

```
getf('pathname/myfct.sci','c')
[a,b]=myfct(1,2)
```

SEE ALSO : `deff` 16, `getf` 39, `comp` 15, `lib` 53, `function` 36, `overloading` 76

1.0.64 funptr ____ coding of primitives (wizard stuff)

DESCRIPTION :
Utility function (for experts only) `funptr('name')` returns `100*fun + fin` where `(fun,fin)` is the internal coding of the primitive `'name'` . `fun` is the interface number and `fin` the routine number

SEE ALSO : `clearfun` 12, `newfun` 73

1.0.65 genlib _ building lib with all functions defined in a directory

CALLING SEQUENCE :

```
getd(lib-name [,path])
```

PARAMETERS :

`lib-name` : Scilab string. The variable name of the library to (re)create
`path` : Scilab string. The directory pathname. Default value is the current directory

DESCRIPTION :
For each `xxxxx.sci` files (containing Scilab functions) defined in the `path` directory, this function executes a getf of `xxxxx.sci` if `xxxxx.bin` does not exist or is older and saves the functions defined in the file in a `xxxxx.bin` file in the given directory

When all `xxxxx.sci` files have been proceed `genlib` creates a library variable named `lib-name` in the scilab environment and save it in the file named `lib` in the given directory

EXAMPLE :

```
genlib('auto1','SCI/macros/auto')
disp(auto1)
```

SEE ALSO : `getf` 39, `lib` 53

1.0.66 getcwd _ get Scilab current directory

CALLING SEQUENCE :

```
path=getcwd()
pwd
```

PARAMETERS :

`path` : a string

DESCRIPTION :
return in `path` the Scilab current directory.

EXAMPLE :

```
p=getcwd();
pwd
```

SEE ALSO : `chdir` 12, `unix` 104

1.0.67 getdate _ get date and time information

CALLING SEQUENCE :

```
dt=getdate()
```

PARAMETERS :

`dt` : an integer vector with 9 entries

DESCRIPTION :
dt=getdate() returns

`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 week number of the year (Monday as the first day of the week). a decimal number between 1 and 53. If the week containing January 1 has four or more days in the new year, then it is considered week 1; otherwise, it is week 53 or the previous year, and the next week is week 1.
`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 Monday.
`dt(6)` : The day of the month as a number between 01 and 31.
`dat(7)` : The hour of the day as a number between 00 and 23.
`dt(8)` : The minute as a number between 00 and 59.
`dt(9)` : The second is output as a number between 00 and 61.

EXAMPLES :

```
w=getdate()
mprintf("Year:%d,Month:%d,Day%d",w(1),w(2),w(7));
```

SEE ALSO : `date` 15, `timer` 243

1.0.68 getd _ getting all functions defined in a directory

CALLING SEQUENCE :

```
getd(path)
```

PARAMETERS :

`path` : Scilab string. The directory pathname

DESCRIPTION :
loads all `.sci` files (containing Scilab functions) defined in the `path` directory.

EXAMPLE :

```
getd('SCI/macros/auto')
```

SEE ALSO : `getf` 39, `lib` 53, `getcwd` 37, `pwd` 85, `chdir` 12

1.0.69 getenv _ get the value of an environment variable

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
getenv('SCI')
getenv('FOO','foo')
```

1.0.70 getfield ____ list field extraction

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
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` 24

1.0.71 getf ____ loading function

CALLING SEQUENCE :

```
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"

DESCRIPTION :
loads one or several 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)
```

followed by 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, the following `function` keyword or the `EOF` mark.

REMARK :
Last line of the file must be terminated by a newline to be taken into account.

EXAMPLE :

```
getf('SCI/macros/xdess/plot.sci')
```

SEE ALSO : `genlib` 37, `getd` 38, `exec` 22, `edit` 201, `comp` 15

1.0.72 getpid ____ get Scilab process identificator

CALLING SEQUENCE :

```
id=getpid()
```

DESCRIPTION :
Return an the scilab process identificator integer

EXAMPLE :

```
d='SD_'+string(getpid())+'_'
```

1.0.73 global ____ Define global variable

CALLING SEQUENCE :

```
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` 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.

EXAMPLE :

```
//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` 109, `isglobal` 51, `clearglobal` 13, `gstacksize` 41, `resume` 89

1.0.74 gstacksize ________ set/get scilab global stack size

CALLING SEQUENCE :

```
gstacksize(n)
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.

SEE ALSO : `who` 109, `stacksize` 99

1.0.75 hat ________ - exponentiation

CALLING SEQUENCE :

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

EXAMPLES :

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

SEE ALSO : `exp` 432, `inv` 439

1.0.76 help ________ on-line help command

CALLING SEQUENCE :

```
help word
```


DESCRIPTION :
To each documented word corresponds a `word.cat` ascii file. these files are organised within directories (chapters). Each chapter must contain `*.cat` files and a `whatis` file with one line for each documented word in the chapter. Each line must have the following format :

```
word - quick description
```

List of chapter directories is given in a file (whose path is given in the `$MANCHAPTERS` environment variable) with the following format for each chapter:

```
chapter_path    chapter_title
```

`MANCHAPTERS` default value is SCI/man/Chapters. If you want to add new help chapters you have to do a copy of the SCIDIR/man/Chapters file where you want, to add descriptions lines for each new chapter and to define the new value of MANCHAPTERS environment variable.
See also Scilab's manual

SEE ALSO : `apropos` 7

1.0.77 %helps ________ Variable defining the path of help directories

DESCRIPTION :
The 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 a 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.star"`.
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 which appears in the bottom part of the help window, raised by clicking on the help button.
A valid help directory must contain:

1- A set of `.cat` files (e.g. `item1.cat, item2.cat` etc). The `.cat` files do do require special format. Usually, they are built as Unix man pages.

2- A `whatis` file, which must have a special format. Each row of the `whatis` must be as follows:

```
item - what is item @item
```

`item` is the item of the help, i.e. the command `help item` returns the contents of the file `item.cat`.
`what is item` is a brief description of the item.

The`whatis` file appears in the top window of the help window, once a chapter has been selected in the bottom window.

Clicking on one item of the top window opens the manual page.

The command `apropos keyword` returns the row(s) of all the `whatis` file(s) in which the keyword appears.
On Unix-Linux platforms Scilab provides a Makefile for transforming `.man` pages into `.cat` pages (see SCIDIR/examples/man-examples).

SEE ALSO : `help` 41, `apropos` 7

1.0.78 host ______ shell (sh) command execution

CALLING SEQUENCE :

```
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 sh shell. 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 sh return code.

EXAMPLE :

```
host("ls $SCI/demos");
host("emacs $SCI/demos/wheel2/Makefile");
deff('wd=pwd()','if MSDOS then host(''cd>''+TMPDIR+''\path'');..
                 else host(''pwd>''+TMPDIR+''/path'');end..
      wd=read(TMPDIR+''/path'',1,1,''(a)'')')
wd=pwd()
```

See Also : edit 201, manedit 216, unix_g 104, unix_s 105, unix_w 105, unix_x 106

1.0.79 hypermat ______ initialize an N dimensional matrices

CALLING SEQUENCE :

```
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,..`

EXAMPLES :

```
M=hypermat([2 3 2 2],1:24)
```

1.0.80 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)

EXAMPLES :


```
a(1,1,1,1:2)=[1 2]
a=[1 2;3 4];a(:,:,2)=rand(2,2)
a(1,1,:)
[a a]
```

See Also : hypermat 43

1.0.81 iconvert ___ conversion to 1 or 4 byte integer representation

CALLING SEQUENCE :

```
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]

EXAMPLE :

```
b=int32([1 -120 127 312])
y=iconvert(b,8)
```

See Also : double 18, inttype 50

1.0.82 ieee ______ set floating point exception mode

CALLING SEQUENCE :

```
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 produce an error
`1` : floating point exception produce a warning
`2` : floating point exception procudes Inf or Nan

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

The initial mode value is 0.

REMARKS :
Floating point exeception arizing inside some library algorithms are not yet handled by `ieee` modes.

EXAMPLE :

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

SEE ALSO : `errcatch` 20

1.0.83 if _______________ else - conditional execution

SYNTAX :

```
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.
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 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.

EXAMPLE :

```
   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 : `while` 109, `select` 94, `boolean` 8, `end` 19, `then` 101, `else` 19

1.0.84 imag _______________ imaginary part

CALLING SEQUENCE :

```
[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` 89

1.0.85 insertion _______________ matrix and list insertion or modification

CALLING SEQUENCE :

```
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` `i` and `j`, may be:
- real scalars or vectors or matrices with positive elements.
* 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 `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 which 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,'*')

- 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.

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 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 ith 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.

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.

EXAMPLE :

- a character string associated with a sub-list entry name. // 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 29, horner 414, parents 78, extraction 24

1.0.86 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

CALLING SEQUENCE :

```
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]

EXAMPLE :

```
int8([1 -120 127 312])
uint8([1 -120 127 312])

x=int32(-200:100:400)
int8(x)
```

SEE ALSO : `double` 18, `inttype` 50, `iconvert` 44

1.0.87 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` 31, `external` 24, `addinter` 6

1.0.88 int _ integer part

CALLING SEQUENCE :

```
[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` 90, `floor` 30, `ceil` 12

1.0.89 intppty _ set interface argument passing properties

CALLING SEQUENCE :

```
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` 36, `addinter` 6

1.0.90 inttype _ type integers used in integer data types

CALLING SEQUENCE :

```
[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

EXAMPLE :

```
x=uint16(1:10);
inttype(x)
```

SEE ALSO : `int8` 48

1.0.91 inv_coeff ______ build a polynomial matrix from its coefficients

CALLING SEQUENCE :

```
[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 it returns a polynomial matrix of degree k. `C=[C0,C1,...,Ck]` and `P= C0 + C1*x +... +Ck*x^k`.

EXAMPLE :

```
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` 81, `degree` 16, `coeff` 14

1.0.92 iserror ______ error test

CALLING SEQUENCE :

```
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` 21, `errcatch` 20

1.0.93 isglobal ______ check if a variable is global

CALLING SEQUENCE :

```
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.

EXAMPLE :

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

SEE ALSO : `global` 40, `clearglobal` 13, `who` 109

1.0.94 keyboard ______ keyboard commands

DESCRIPTION :
Let `C-` stands for the control key. The following keyboard commands are available:

`C-l` clears the Scilab window
`C-d` deletes the current character
`C-p` calls back the preceding command
`C-n` go to next command line
`C-a` moves the cursor to the beginning of command line.
`C-b` backspace, moves the cursor one character to the left
`C-f` forwards, moves the cursor one character to the right
`C-k` kills command line from cursor to the end.
`C-y` yank, retrieves killed line.
`!beg` looks for last command line which begins by `beg`.
`C-c` interrupts Scilab and pause after carriage return. (Only functions can be interrupted). Clicking on the stop button enters a `C-C`.

SEE ALSO : `pause` 79, `read` 88, `input` 207

1.0.95 left ______ - left bracket

CALLING SEQUENCE :

```
[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", semicolumn 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']

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

1.0.96 length ______ length of object

CALLING SEQUENCE :

```
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.

SEE ALSO : `size` 96

1.0.97 less - lower 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` 45

1.0.98 lib library definition

CALLING SEQUENCE :

```
xlib=lib('lib_path')
```

PARAMETERS :

`lib_path` : character string

DESCRIPTION :
`lib_path` is a character string defining the path of a directory containing functions . This directory must contain the binary files `names(i).bin` and an additional file `names` (which contains the names of the functions). After the command `lib` all the functions of `lib_path` can be called interactively by Scilab.

Such a binary file, for example `foo.bin` can be created by Scilab using the command `save(lib_path+'/foo.bin'`

Standard Scilab libraries are defined using `lib` on SCIDIR/macros/* subdirectories

EXAMPLE :

```
deff('[z]=myplus(x,y)','z=x+y')
deff('[z]=yourplus(x,y)','x=x-y')
```


are two functions and `lib_path` is : `lib_path='/usr/mymachine/mydirectory'`
This directory contains the file `names` including `myplus` (first line of the file) and `yourplus` (second line of the file).
`myplus` and `yourplus` are compiled functions respectively saved in the files
`'/usr/mymachine/mydirectory/myplus.bin'`
`'/usr/mymachine/mydirectory/yourplus.bin'`
by the command:

```
save(lib_path+'/myplus.bin',myplus)
save(lib_path+'/yourplus.bin',yourplus)
```

A library can now be created with the command:

```
xlib=lib(lib_path+'/')
```

`xlib` is then a Scilab variable which can be saved and loaded in a future session of Scilab or defined on-line or put in the startup (`.scilab`) file. The functions in this library are known by Scilab which automatically loads them when necessary.

SEE ALSO : `genlib` 37, `save` 91, `deff` 16, `getf` 39, `whereis` 108

1.0.99 lines rows and columns used for display

CALLING SEQUENCE :

```
[nl,nc]=lines([n [,nc]])
```

DESCRIPTION :
`lines` handles Scilab display paging.

`lines()` returns the vector [# columns, # rows] currently used by Scilab for displaying the results.
`lines(n)` sets the number of displayed lines (before user is asked for more) to `n`.
`lines(0)` disables vertical paging
`lines(n,nc)` changes also the size of the output to `nc` columns.

1.0.100 link dynamic link

CALLING SEQUENCE :

```
link(files, sub-name)
link(files, sub-name, flag)
lst=link('show')
// Link extensions for machines using ``dlopen''
// (sun-solaris/linux-elf/alpha/hppa)
x=link(files [, sub-names,flag]);
link(x , sub-names [, flag]);
ulink(x)
```

PARAMETERS :

`files` : a character string or a vector of character strings. `ld` files used to define the new entry point (compiled routines, user libraries, system libraries,..)
`sub-name` : a character string. Name of the entry point in `files` to be linked.
`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.
`names` : a vector of character string. Names of dynamically linked entry points.

DESCRIPTION :

`link` is a 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 `fort`. Linked routines can also be used as "external" for e.g. non linear problem solvers (`ode`, `optim`, `intg`, `dassl`...). Here are some examples:
The command `link('foo.o','foo','f')` links the Fortran object file `foo.o` with the entry point `foo`.
The command `link('foo.o','foo','c')` links the C object file `foo.o` with the entry point `foo`.
The command `link('SCIDIR/libs/calelm.a','dcopy')` links the Fortran routine `dcopy` in the library `calelm.a`.
A routine can be linked several times and can be unlinked with `ulink`. Note that, on some architectures (the ones on which `ulink` exists) when a routine is linked several times, all the version are kept inside Scilab.
Used with no arguments, `link()` returns the current linked routines.
If Scilab is compiled with static link (this is the default for SystemV machines) you may have to include the system libraries in the "link" command.
For example, if `foo.o` defines the object code of a routine named `foo`, you will use `link` in one the following way:

```
link('foo.o','foo').
link('foo.o -lm -lc','foo','c').
link('foo.o -lfor -lm -lc','foo').
link('foo.o -lftn -lm -lc','foo').
link('foo.o -L/opt/SUNWspro/SC3.0/lib/lib77 -lm -lc','foo')
```

If Scilab compiled with the "shared" option, the first example can be used even if a warning for unresolved references is issued.
(Experienced) users may also `link` a new Scilab interface routine to add a set of new functions. See `Intersci` documentation for interface generation and `addinter` function.

REMARKS :

`IBM`: For IBM-RS6000 only one program can be dynamically linked.

`Demo`: When running a demo, you may have some trouble with the link due to slight differences between systems. In this case, you modify the demo by adding the needed libraries in the link command.

`dlopen`: For machines using dlopen functionality extended command can be used. 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.

for example to link functions `f` and `g` form binary file `test.o` the two following command can be used :

```
link('test.o',['f','g'])
```

or

```
x=link('test.o','f');
link(x,'g');
```

But


```
link('test.o','f');
link('test.o','g');
```

will also work but `f` and `g` will be loaded from two different shared libraries and won't be able to share data.

`show`: The command `lst=link('show')` will report information about linked shared libraries and linked functions. The return value of the function `lst` is `1` or `0`. If the return value is `1` then the extended calling sequence described as `Link extensions for machines using ''dlopen''` are accepted.

`unlink` : (dlopen version) If the function `f` is changed and one wants to link the new version, it is necessary to use unlink to get rid of previous loaded versions of the function `f`

```
x=link('test.o','f');
// if I need to reload a new definition of f a call to unlink
// is necessary.
ulink(x);
link('test.o','f');
```

`scilab symbols`: In order to load a symbol from the Scilab code on can use

```
link("Scilab",['Scilab-entry-point'])
```

This does not work on all architectures. On some machines, on can link a Scilab internal function after a first call to link (with a default binary file)

```
link("test.o",['Scilab-entry-point'])
```

Note that with `dld` (Linux machine aout) you can use an empty string

```
link(" ",['Scilab-entry-point'])
```

SEE ALSO : `fort` 31, `c_link` 193, `addinter` 6

1.0.101 list ________ Scilab object and list function definition

CALLING SEQUENCE :

```
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()` is 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 : `l(i)=a`
deletion : `l(i)=null()` removes the i-th element of the `list l`.

EXAMPLE :

```
x=list(1,2,3);
x(4)=10;
x(10)='a'
```

SEE ALSO : `null` 74, `tlist` 102, `insertion` 46, `extraction` 24, `size` 96, `length` 52

1.0.102 load ______ load saved variable

CALLING SEQUENCE :

```
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.
`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`.
COMPATIBLITY :
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` 91, `getf` 39, `mopen` 68

1.0.103 lsslist ______ Scilab linear state space function definition

CALLING SEQUENCE :

```
lsslist()
lsslist(a1,....an)
```

DESCRIPTION :
`lsslist(a1,....an)` is a shortcut to 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` 102, `syslin` 241

1.0.104 lstcat ______ list concatenation

CALLING SEQUENCE :

```
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 `li` are other type of variables they are simply added to the resulting list.
EXAMPLE :

```
lstcat(list(1,2,3),33,list('foo',%s))
lstcat(1,2,3)
```

SEE ALSO : `list` 56

1.0.105 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` 16, `getf` 39, `comp` 15, `lib` 53

1.0.106 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.
`syslin` lists in state-space form or transfer matrices can also be defined as above.
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` 81, `string` 100, `boolean` 8, `rational` 86, `syslin` 241, `empty` 19, `hypermatrices` 43

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

CALLING SEQUENCE :

```
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 matrix or hypermatrix

DESCRIPTION :
For a vector or a matrix with `n x m` entries `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)`,`y=matrix(v,sizes)` (or equivalently `y=matrix(v,n1,n2,...nm)`) transforms `v` into an matrix or hypermatrix by stacking columnwise the entries of `v`. `y=matrix(v,sizes)` results in a regular matrix if sizes is a scalar or a 2-vector.

SEE ALSO : `matrices` 58, `hypermatrices` 43, `ones` 75, `zeros` 252, `rand` 86, `poly` 81, `empty` 19

EXAMPLES :

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

1.0.108 maxi maximum

CALLING SEQUENCE :

```
[m [,k]]=maxi(A)
[m [,k]]=maxi(A,'c') or [m [,k]]=maxi(A,'r')
[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=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)= max( 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)`

EXAMPLE :

```
[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` 228, `find` 29, `mini` 65

1.0.109 max maximum

CALLING SEQUENCE :

```
[m [,k]]=max(A)
[m [,k]]=max(A,'c') or [m [,k]]=max(A,'r')
[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=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)`

EXAMPLE :

```
[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` 228, `find` 29, `mini` 65

1.0.110 mclearerr reset binary file access errors

CALLING SEQUENCE :

```
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 : `mclose` 61, `mopen` 68, `mput` 69, `mget` 64, `mgetstr` 65, `mputstr` 70, `meof` 61, `mseek` 71, `mtell` 72, **??** file, read, write, save, load

1.0.111 mclose ______ close an opened file

CALLING SEQUENCE :

```
err=mclose([fd])
```

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.
To close files opened using `file('open',...)` use function `file('close',...)`.

SEE ALSO : `mclose` 61, `meof` 61, `mfprintf` 68, `fprintfMat` 34, `mfscanf` 70, `fscanfMat` 35, `mget` 64, `mgetstr` 65, `mopen` 68, `mprintf` 68, `mput` 69, `mputstr` 70, `mscanf` 70, `mseek` 71, `mtell` 72

1.0.112 meof ______ check if end of file has been reached

CALLING SEQUENCE :

```
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` 61, `meof` 61, `mfprintf` 68, `fprintfMat` 34, `mfscanf` 70, `fscanfMat` 35, `mget` 64, **??** mgetstr, mopen, mprintf, mput, mputstr, mscanf, mseek, mtell

1.0.113 mprintf ______ interface to the C printf function

mfprintf - interface to the C fprintf function
msprintf - interface to the C sprintf function

CALLING SEQUENCE :

```
mfprintf(fd,format,value_1,..,value_n)
mprintf(format,value_1,..,value_n)
str=msprintf(format,value_1,..,value_n)
```

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.
`value_i` : Specifies the data to be converted according to the format parameter.
`str` : a Scilab string.
`fd:` The fd parameter returned by the function `mopen` is used as a file descriptor (it is a positive integer). When specifying the fd parameter, the value -1 refers to the default file (i.e the last opened file).

DESCRIPTION :

The `mfprintf` function writes formatted operands to the stream `fd`. The argument operands are formatted under control of the format operand.
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 `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 are treated as a normal sequence of characters.

EXAMPLES :

```
mprintf('For iteration %i,\\tResult is:\\nalpha=%f",10,0.535)

t=msprintf('For iteration %i, Result is: alpha=%f",10,0.535)

mprintf('The hexadecimal value of %i is %x',123456,123456)

x=1.23456789;
mprintf('!%f!%15f!%.1f!%#.0f!%.13f!',x,x,x,x,x);
x=-341.234567;
mprintf('!%g!%15g!%.1g!%#.0g!%.13g!',x,x,x,x,x);

x=-0.0000023456;
mprintf('!%e!%15e!%.1e!%#.0e!%.13e!',x,x,x,x,x);

u=mopen(TMPDIR+'/foo','w')
t=0:0.1:2*%pi;
for tk=t
 mfprintf(u,'time = %6.3f value = %6.3f\n',tk,sin(tk)) // write a line
end
mclose(u) //close the result file
unix_g('cat '+TMPDIR+'/foo')
```

SEE ALSO : `mclose` 61, `meof` 61, `mfprintf` 68, `fprintfMat` 34, `mfscanf` 70, `fscanfMat` 35, `mget` 64, `mgetstr` 65, `mopen` 68, `mprintf` 68, `mput` 69, `mputstr` 70, `mscanf` 70, `mseek` 71, `mtell` 72

1.0.114 mscanf ______________ interface to the C scanf function

mfscanf - interface to the C fscanf function
msscanf - interface to the C sscanf function

CALLING SEQUENCE :

```
[n,v_1,...v_n]=mfscanf(fd,format)
L=mfscanf(fd,format)

[n,v_1,...v_n]=mscanf(format)
L=mscanf(format)

[n,v_1,...v_m]=msscanf(format,str)
L=msscanf(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.
`fd` :The fd parameter returned by the function `mopen` is used as a file descriptor (it's a positive integer). When specifying the fd parameter, the value -1 refers to the default file (i.e the last opened file).
`str` : a Scilab string.
`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` : a list of the `v_1,...,v_n` variables or -1 if EOL has been encountered before any datum has been read.

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`.

EXAMPLES :

```
s='1 1.3'
[n,a,b]=msscanf(s,"%i %e")
msscanf(s,"%i %e")


msscanf(' 12\n','%c%c%c%c') //scan characters

msscanf('0xabc','%x') //scan with hexadecimal format


msscanf('012345abczoo','%[0-9abc]%s')  //[] notation

//create a file with data
u=mopen(TMPDIR+'/foo','w');
t=0.5;mfprintf(u,'%6.3f %6.3f\n',t,sin(t))
t=0.6;mfprintf(u,'%6.3f %6.3f\n',t,sin(t))
mclose(u);
//read the file
u=mopen(TMPDIR+'/foo','r');
```



```
[n,a,b]=mfscanf(u,'%e %e')
l=mfscanf(u,'%e %e')
mclose(u);
```

SEE ALSO : mclose 61, meof 61, mfprintf 68, fprintfMat 34, mfscanf 70, fscanfMat 35, mget 64, mgetstr 65, mopen 68, mprintf 68, mput 69, mputstr 70, mscanf 70, mseek 71, mtell 72

1.0.115 mget ________ reads byte or word in a given binary format

CALLING SEQUENCE :

```
x=mget([n,type,fd])
```

PARAMETERS :

`n` :a positive scalar: The number of items to be read.
`fd` : 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 numbers

DESCRIPTION :
The `mget` function reads data in the input specified by the stream parameter `fd`. Data is read at the position at which the file pointer is currently pointing and advances the indicator appropriately. The `tye` 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 reading 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 read are automatically swapped if necessary (by checking little-endian status). This default swapping mode can be suppressed by adding a flag in the `mopen` function.
`"..l" or "..b"` : It is also possible to read 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 read a double in big-endian mode.

EXAMPLE :

```
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 61, meof 61, mfprintf 68, fprintfMat 34, mfscanf 70, fscanfMat 35, mget 64, mgetstr 65, mopen 68, mprintf 68, mput 69, mputstr 70, mscanf 70, mseek 71, mtell 72

1.0.116 mgetstr — read a character string

CALLING SEQUENCE :

```
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` 61, `meof` 61, `mfprintf` 68, `fprintfMat` 34, `mfscanf` 70, `fscanfMat` 35, `mget` 64, `mgetstr` 65, `mopen` 68, `mprintf` 68, `mput` 69, `mputstr` 70, `mscanf` 70, `mseek` 71, `mtell` 72

1.0.117 mini — minimum

CALLING SEQUENCE :

```
[m [,k]]=mini(A)
[m [,k]]=mini(A,'c') or [m [,k]]=mini(A,'r')
[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 smallest 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=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)`

EXAMPLE :

```
[m,n]=mini([1,3,1])
[m,n]=mini([3,1,1],[1,3,1],[1,1,3])
[m,n]=mini(list([3,1,1],[1,3,1],[1,1,3]))
[m,n]=mini(list(1,3,1))
```

SEE ALSO : `sort` 228, `find` 29, `maxi` 59, `min` 66

1.0.118 min — minimum

CALLING SEQUENCE :

```
[m [,k]]=min(A)
[m [,k]]=min(A,'c') or [m [,k]]=min(A,'r')
[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 smallest 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=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)`

EXAMPLE :

```
[m,n]=min([1,3,1])
[m,n]=min([3,1,1],[1,3,1],[1,1,3])
[m,n]=min(list([3,1,1],[1,3,1],[1,1,3]))
[m,n]=min(list(1,3,1))
```

SEE ALSO : `sort` 228, `find` 29, `max` 60

1.0.119 minus — - substraction operator, sign changes

CALLING SEQUENCE :

```
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.
EXAMPLE :

```
[1,2]-1
[]-2

%s-2
1/%s-2
"cat"+"enate"
```

SEE ALSO : addf 181, mtlb_mode 72

1.0.120 mlist ____ Scilab object, matrix oriented typed list definition.

CALLING SEQUENCE :

```
mlist(typ,a1,....an )
```

PARAMETERSP :

`typ` : vector of character strings
`ai` : any Scilab object (`matrix, list,string...`).

DESCRIPTION :
mlist object are very similar to tlist objects. But if `M` is an mlist, for any index `i` which is not a field name, `M(i)` is interpreted as the `i` th entry of `M` seen has a matrix.
mlist fields must then be designed by there names.

EXAMPLE :

```
M=mlist(['V','name','value'],['a','b','c'],[1 2 3]);
//define display
deff('%V_p(M)','disp([M(''name'');string(M(''value''))])')
//define extraction operation
deff('r=%V_e(i,M)','r=mlist([''V'',''name'',''value''],M(''name'')(i),M(''value'')(i))')
//define M as a tlist
M=tlist(['V','name','value'],['a','b','c'],[1 2 3]);
M(2)
M('name')
```

SEE ALSO : tlist 102, list 56

1.0.121 mode ____ select a mode in exec file

CALLING SEQUENCE :

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

DESCRIPTION :
Used 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 semi 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. 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.

line display is disable for compiled scilab function (see getf or comp)

SEE ALSO : exec 22, getf 39, semi 95, comma 15

1.0.122 mopen ____ open a file

CALLING SEQUENCE :

```
[fd,err]=mopen(file [, mode, swap ])
```

PARAMETERS :

`file` : a character string. The pathname of the file to open.
`mode` : a character string. The `mode` parameter can be `"rb"` for read binary or `"wb"` for write binary or `"ab"` for append. The default value is `"rb"`.
`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.
To open files in a way compatible with Fortran like functions use function `file`.

SEE ALSO : mclose 61, meof 61, mfprintf 68, fprintfMat 34, mfscanf 70, fscanfMat 35, mget 64, mgetstr 65, mopen 68, mprintf 68, mput 69, mputstr 70, mscanf 70, mseek 71, mtell 72

1.0.123 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

CALLING SEQUENCE :

```
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). The value `-1` refers to the default file (i.e the last opened file).
`format` : a character string, a C like format. See cformat for details
`str` : a character string, string to be scanned.
`a1,...,an` : variables to print.

DESCRIPTION :

The mprintf, mfprintf, and msprintf functions are interface for C-coded version of `fprintf`, `sprintf` and `printf` functions.
The mprintf function converts, formats the `ai` parameters and display the result in the main Scilab window
mfprintf function converts, formats the `ai` parameters and store the result in the file given by the file descriptor `fd`.
msprintf function converts, formats the `ai` parameters and store the result in the return variable `str`.

EXAMPLE :

```
mprintf('At iteration %i, Result is:\nalpha=%f',33,0.535)

msprintf('%f %f',123,0.732)
msprintf('%f\n%f',123,0.732)
```

SEE ALSO : mclose 61, meof 61, mfprintf 68, fprintfMat 34, mfscanf 70, fscanfMat 35, mget 64, mgetstr 65, mopen 68, mprintf 68, mput 69, mputstr 70, mscanf 70, mseek 71, mtell 72

1.0.124 mput _ writes byte or word in a given binary format

CALLING SEQUENCE :

```
err=mput(x [,type,fd])
```

PARAMETERS :

x : a vector of numbers
err : a scalar. Error indicator
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.
err : a scalar. Error indicator

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 tye 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.

EXAMPLE :

```
  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 61, meof 61, mfprintf 68, fprintfMat 34, mfscanf 70, fscanfMat 35, mget 64, mgetstr 65, mopen 68, mprintf 68, mput 69, mputstr 70, mscanf 70, mseek 71, mtell 72

1.0.125 mputstr _ write a character string in a file

CALLING SEQUENCE :

```
err=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
err : scalar. Error indicator

DESCRIPTION :

mputstr function allows to write a character string in a binary file.

SEE ALSO : mclose 61, meof 61, mfprintf 68, fprintfMat 34, mfscanf 70, fscanfMat 35, mget 64, mgetstr 65, mopen 68, mprintf 68, mput 69, mputstr 70, mscanf 70, mseek 71, mtell 72

1.0.126 mfscanf _ scan data from file

mscanf - scan data from input
msscanf - scan data from string

CALLING SEQUENCE :

```
[n,a1,...,am]=mfscanf(fd,format);
data=mfscanf(fd,format);

[n,a1,...,am]=mscanf(format);
data=mscanf(format);

[n,a1,...,am]=msscanf(str,format);
data=msscanf(str,format);
```

PARAMETERS :

`fd` : scalar, file descriptor given by `mopen` (it's a positive integer). The value `-1` refers to the default file (i.e the last opened file).
`format` : a character string, a C like format.
`str` : a character string, string to be scanned.
`n` : a scalar integer, the number or data really read.
`a1,...,am` : Output variables. if `m>n` the `n+1:n` last `ai'` are set to `[]`.
`data` : a list formed by the data really read.

DESCRIPTION :
The mscanf(), mfscanf(), and msscanf() functions are interface for C-coded version of `fscanf`, `sscanf` and `scanf` functions.
The mscanf(), mfscanf(), and msscanf() functions read character data, interpret it according to a format (see cformat), and store the converted results into variables. The format parameter contains conversion specifications used to interpret the input.
These functions read their input from the following sources:

`mscanf()` : Reads from the Scilab input.
`mfscanf()` : Reads from the file given by the file descriptor `fd`.
`msscanf()` : Reads from the character string specified by the `str` parameter.

EXAMPLE :

```
[n,a1,a2]=msscanf('123 456','%i %s')
[n,a1,a2,a3]=msscanf('123 456','%i %s')
data=msscanf('123 456','%i %s')

fd=mopen(SCI+'/scilab.star','r')
mfscanf(fd,'%s %s %s')
mclose(fd)
```

SEE ALSO : mclose 61, meof 61, mfprintf 68, fprintfMat 34, mfscanf 70, fscanfMat 35, mget 64, mgetstr 65, mopen 68, mprintf 68, mput 69, mputstr 70, mscanf 70, mseek 71, mtell 72

1.0.127 mseek ______ set current position in binary file.

CALLING SEQUENCE :

```
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.
EXAMPLE :


```
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 61, meof 61, mfprintf 68, fprintfMat 34, mfscanf 70, fscanfMat 35, mget 64, mgetstr 65, mopen 68, mprintf 68, mput 69, mputstr 70, mscanf 70, mseek 71, mtell 72

1.0.128 mtell ______ binary file management

CALLING SEQUENCE :

```
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 61, meof 61, mfprintf 68, fprintfMat 34, mfscanf 70, fscanfMat 35, mget 64, mgetstr 65, mopen 68, mprintf 68, mput 69, mputstr 70, mscanf 70, mseek 71, mtell 72

1.0.129 mtlb_mode ______ switch Matlab like operations

CALLING SEQUENCE :

```
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` 19

1.0.130 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
```

1.0.131 newfun ________ add a name in the table of functions

CALLING SEQUENCE :

```
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` 12

1.0.132 not ________ - logical not

CALLING SEQUENCE :

```
~A
```

DESCRIPTION :
`~A` gives the element-wise negation of the elements of the boolean matrix `A`.

EXAMPLES :

```
~[%t %t %f]
```

SEE ALSO : `and` 6, `or` 76, `find` 29

1.0.133 null ________ delete an element in a list

CALLING SEQUENCE :

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

DESCRIPTION :
Deletion of objects inside a list

EXAMPLE :

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

SEE ALSO : `list` 56, `clear` 13

1.0.134 oldload ____ load saved variable in 2.4.1 and previous formats

CALLING SEQUENCE :

```
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` 91, `getf` 39

1.0.135 oldsave ________ saving variables in 2.4.1 and previous format

CALLING SEQUENCE :

```
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');
// sequential save into a file

fd=file('open','TMPDIR/foo','unformatted','unknown')
for k=1:4, x=k^2;oldsave(fd,x,k),end
file('close',fd)
fd=file('open','TMPDIR/foo','old','unknown')
for i=1:4, oldload(fd,'x','k');x,k,end
file('close',fd)
```

See Also : `load` 57, `file` 28

1.0.136 ones ________ matrix made of ones

CALLING SEQUENCE :

```
y=ones(m,n)
y=ones(x)
y=ones()
```

DESCRIPTION :

Returns a matrix made of ones.

`ones(m,n)` returns a `(m,n)` 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).

See Also : `eye` 27, `zeros` 252

1.0.137 or ________ - logical or

CALLING SEQUENCE :

```
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` 6, `not` 74, `find` 29

1.0.138 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.

`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:

string	c
polynomial	p
function	m
constant	s
list	l
tlist	¡tlist_type¿
boolean	b
sparse	sp
boolean sparse	spb

<op_code> is a single character associated with each operator as described in the following table:

'	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

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_a>_i_<type_of_b>(i1,...,in,a,b)`.

`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.

SEE ALSO : `tlist` 102, `disp` 17, `symbols` 101

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')
```

1.0.139 parents ______) - left and right parenthesis

CALLING SEQUENCE :

```
(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 x1=e1, x2=e2, ...

EXAMPLE :

```
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 14, comma 15, brackets 9, list 56, extraction 24, insertion 46

1.0.140 part ______ extraction of strings

CALLING SEQUENCE :

```
[c]=part(mp,v)
```

PARAMETERS :

`mp,c` : string matrices
`v` : integer vector.

DESCRIPTION :
Let `s[k]` stands for the `k` character of string `s` (or the empty character if `k` >`length(s)`).
`part` returns `c`, a matrix of character strings, such that `c(i,j)` is the string `"s[v(1)]...s[v(n)]"` (`s=mp(i,j)`).

EXAMPLE :

```
c=part(['a','abc','abcd'],[1,1,2])
```

SEE ALSO : string 100, length 52

1.0.141 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 session in which all the lower-level variables (and in particular all the variable of the function) are available. To return to lower session enter `"return"`
In this mode, `[...]=return(...)` returns the variables of the argument `(...)` to lower session 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 : return 90, abort 6, quit 85, whereami 108, where 109

1.0.142 percent ______ - special character

DESCRIPTION :
Some predefined variables begin with `%`, such as `%i` (for `sqrt(-1)`), `%inf` (for `Infinity`), `%pi` (for `3.14...`), `%T` (for the boolean variable `"true"`),...
In addition, functions whose names begin with `%` are special : they are used for coding (extensions of usual) operations .
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`.

EXAMPLE :

```
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
```

1.0.143 plus ______ - addition operator

CALLING SEQUENCE :

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

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
`str1` : a character string, a vector or a matrix of character strings
`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.
For character strings `+` means concatenation.
Addition may also be defined for other data types through "soft-coded" operations.

EXAMPLE :

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

SEE ALSO : addf 181, mtlb_mode 72

1.0.144 poly _______ polynomial definition

CALLING SEQUENCE :

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

PARAMETERS :

`a` : matrix or real number
`x` : symbolic variable
`"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).
`poly(v,"x","coeff")` creates the polynomial with symbol `"x"` and with coefficients the entries of `v`. (Here `poly` and `coeff` are inverse functions).
`s=poly(0,"s")` is the seed for defining polynomials with symbol `"s"`.

EXAMPLE :

```
s=poly(0,"s");p=1+s+2*s^2;
A=rand(2,2);poly(A,"x")
```

SEE ALSO : `coeff` 14, `matrices` 58, `rational` 86

1.0.145 power _______ power operation (^,.^)

CALLING SEQUENCE :

```
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 scalar 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) with compatible dimensions `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.

EXAMPLE :

```
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` 432

1.0.146 predef _______ variable protection

CALLING SEQUENCE :

```
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 fVwho('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 `scilab.star`.
User may in particular set its own predefined variables in user's startup file `home/.scilab`

SEE ALSO : `clear` 13, `save` 91

1.0.147 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 -).
`"space"` : 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.
`0` : 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 %.
`d,i` :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.
`u` :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.
`o` :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.
`x, X` :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.
`f` : 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.

`e, E` :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.

`g, G` : 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.
`c` :Accepts and displays an integer value converted to a character.
`s` :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` 84, `fprintf` 33, `sprintf` 97

1.0.148 printf Emulator of C language printf function

CALLING SEQUENCE :

```
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` 100, `print` 85, `write` 111, `format` 31, `disp` 17, `file` 28, `fprintf` 33, `sprintf` 97

1.0.149 print ______ prints variables in a file

CALLING SEQUENCE :

```
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` 111, `read` 88, `format` 31, `printf` 84, `disp` 17

1.0.150 pwd ______ print Scilab current directory

CALLING SEQUENCE :

```
pwd
```

DESCRIPTION :

return in `ans` the Scilab current directory.

EXAMPLE :

```
pwd
```

SEE ALSO : `getcwd` 37, `chdir` 12, `unix` 104

1.0.151 quit ______ decrease the pause level or exit

DESCRIPTION :

`quit` terminates Scilab or decreases the `pause` level.

SEE ALSO : `pause` 79, `break` 9, `abort` 6, `exit` 24

1.0.152 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'''
```


1.0.153 rand ______ random number generator

CALLING SEQUENCE :

```
rand(m,n [,rtype])
rand(x [, rtype])
rand('key'[,n])
rand()
```

DESCRIPTION :

random matrix generator.

`rand(m,n)` is a random matrix of dimension mxn.
`rand(a)` is a random matrix of same size as a. `rand(a)` is complex if `a` is a complex matrix
`rand('uniform')` The default random generator is set to a uniform random number generator.
`rand('normal')` The default random generator is set to a Gaussian random number generator.
`rand('seed')` returns the current value of the seed.
`rand('seed',n)` puts the seed to `n`. (`n=0` at first call).
`str=rand('info')` return the type of the default random generator ('uniform' or 'normal')

`rand()` : with no arguments gives a scalar whose value changes each time it is referenced. By default, random numbers are uniformly distributed in the interval (0,1). `rand('normal')` switches to a normal distribution with mean 0 and variance 1. `rand('uniform')` switches back to the uniform distribution.

The type of the random generator can also be locally changed by the use of the extra parameter `rtype` (which can be `'uniform'` or `'normal'`

EXAMPLE :

```
x=rand(10,10,'uniform')
rand('normal')
rand('info')
y=rand(x,'normal');
```

SEE ALSO : `ssrand` 235

1.0.154 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` 81, `syslin` 241, `simp` 421

1.0.155 rat _______ Floating point rational approximation

CALLING SEQUENCE :

```
[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`

SEE ALSO : `int` 49, `round` 90

EXAMPLES :

```
[n,d]=rat(%pi)
[n,d]=rat(%pi,1.d-12)
n/d-%pi
```

1.0.156 read4b _______ fortran file binary read

CALLING SEQUENCE :

```
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` 28, `write` 111, `writb` 110, `mget` 64, `write4b` 110

1.0.157 readb _______ fortran file binary read

CALLING SEQUENCE :

```
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` 28, `write` 111, `writb` 110, `mget` 64, `read4b` 87

1.0.158 read _______ matrices read

CALLING SEQUENCE :

```
[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.
Examples for `format`:

```
(1x,e10.3,5x,3(f3.0))
(10x,a20)
.LP
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.

.LP
A direct access file can be used if using the parameter \fVk\fR which is
is the vector of record numbers to be read (one record per row),
thus \fVm\fR must be \fVm=prod(size(k))\fR.
.LP
To read on the keyboard use \fVread(%io(1),...)\fR.
.SH REMARK
Last line of data files must be terminated by a newline to be taken
into account.
.SH EXAMPLE
.nf
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` 28, `readb` 88, `write` 111, `x_dialog` 249, `scanf` 92

1.0.159 real ______ real part

CALLING SEQUENCE :

```
[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` 45

1.0.160 resume ______ return or resume execution and copy some local variables

CALLING SEQUENCE :

```
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` 6, `break` 9

1.0.161 return ______ return or resume execution and copy some local variables

CALLING SEQUENCE :

```
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 lower level `[..]=return(..)` returns to lower 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` 6, `break` 9

1.0.162 rlist ______ Scilab rational fraction function definition

CALLING SEQUENCE :

```
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` 102, `syslin` 241

1.0.163 round ______ rounding

CALLING SEQUENCE :

```
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.
EXAMPLE :

```
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 49, floor 30, ceil 12

1.0.164 save ______ saving variables in binary files

CALLING SEQUENCE :

```
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. 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)
```

SEE ALSO : load 57, mopen 68

1.0.165 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.

SEE ALSO : scanf 92, scanf 92, fscanf 34

1.0.166 scanf ______ Converts formatted input on standard input

CALLING SEQUENCE :

```
[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 84, read 88, fscanf 34, sscanf 98

1.0.167 sciargs _ scilab command line arguments

CALLING SEQUENCE :

```
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 correspond to the `getarg` function in C langage

SEE ALSO : `getenv` 38

1.0.168 scilab _ Major unix script to execute Scilab and miscellaneous tools

CALLING SEQUENCE :

```
scilab [-ns -nw -display display -f file -args arguments]
scilab -help [ <key> ]
scilab -k <key>
scilab -xk <key>
scilab -link <objects>
```

DESCRIPTION :

`scilab [-ns -nw -display display -f path -args arguments]` : run scilab.
`-ns` :if this option is present the startup file `SCI/scilab.star` is not executed.
`-nw` :if this option is present then scilab is not run in an X window.
`-f file` : if this option is present then Scilab script `file` is executed first into Scilab.
`-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'`
`scilab -help` <key> : write on-line documentation about <key> (usually automatically called by scilab command "help <key>"). Example:

```
scilab -help plot3d
```

`scilab -k` <key> : gives the list of Scilab commands containing the keyword <key> in their description (same as UNIX command `man -k`)
`scilab -xk` <key> : gives the list of Scilab commands containing the keyword <key> in their description in a X window.
`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`>.

If, in the list of object files, some names are known from `SCIDIR/routines/default`, then the `scilex` default files are omitted and replaced with the given ones .

This command also produces an `xscilab` 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`.

1.0.169 scilink _ Unix script to relink Scilab

CALLING SEQUENCE :

```
scilink <object-files>
```

DESCRIPTION :
This script is used to produce a local `scilex` (executable code of Scilab) linked with the additional files given by the user in <`object-files`>.
If in the list of object files some names are known `scilex` names (from `SCIDIR/routines/default`) then the `scilex` default files are omitted and replaced with the given ones .
This script also produces an `xscilab` script, which when called will ran the new generated scilex file.
For example the command

```
scilink 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`.

SEE ALSO : `link` 54, `addinter` 6

1.0.170 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.

EXAMPLE :

```
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` 45, `while` 109, `for` 30

1.0.171 semicolumn - ending expression and row separator

DESCRIPTION :
In a file, the line separator ";" suppresses the display of the line.
Within brackets ; denotes row separator in matrix definition.

EXAMPLES :
sin(%pi) sin(%pi); a=[1,2;3 4]

1.0.172 semi - instruction and row separator

DESCRIPTION :
semicolumns are used to separate rows in a matrix definition.
semicolumns may also be used at the end of an instruction. In this case it means that the result(s) 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` 15, `brackets` 9

1.0.173 setbpt setting breakpoints

CALLING SEQUENCE :

```
setbpt(macro-name [,line-num])
```

PARAMETERS :

`macro-name` : string
`line-num` : integer

DESCRIPTION :
`setbpt` interactively inserts a breakpoint in the line number `line-num` (default value is 1) of the function `macro-name`
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 20 and the maximum number of breakpoints is set to 100.

SEE ALSO : `delbpt` 17, `dispbpt` 17, `pause` 79, `resume` 89

1.0.174 setfield list field insertion

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
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 filed value to 1:8
```

SEE ALSO : `insertion` 46

1.0.175 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.

EXAMPLE :

```
sign(rand(2,3))
sign(1+%i)
```

SEE ALSO : `abs` 180

1.0.176 signm matrix sign function

DESCRIPTION :

For square and Hermitian matrices `X=sign(A)` is matrix sign function.

EXAMPLE :

```
A=rand(4,4);B=A+A';X=sign(B);spec(X)
```

SEE ALSO : `sign` 96

1.0.177 size size of objects

CALLING SEQUENCE :

```
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 :
Size of a (constant, polynomial, string, boolean, rational) matrix `x`, `y = 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

Size (length) of ordinary `list` (number of elements). In this case the syntax must be `y=size(x)`
Caution: if `x` is a `syslin` list representing 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 `syslin` list representing 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`.

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` 52, `syslin` 241

1.0.178 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 operator, not of the number.
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` 439, `percent` 80, `backslash` 7, `ieee` 44

1.0.179 sprintf ______ Emulator of C language sprintf function

CALLING SEQUENCE :

```
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
`file` : a Scilab string specifying a file name or a logical unit number (see `file`)

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 `values` remain after the entire `format` has been processed, they are ignored.
Note that sprintf is a scilab emulation of C language function build in Scilab. Consequently it is quite slow. Use string whenever it is possible.

EXAMPLES :

```
fahr=120
sprintf('%3d Fahrenheit = %6.1f Celsius',fahr,(5/9)*(fahr-32))
```

SEE ALSO : `string` 100, `print` 85, `write` 111, `format` 31, `disp` 17, `file` 28, `printf` 84, `fprintf` 33

1.0.180 sscanf ______ Converts formatted input given by a string

CALLING SEQUENCE :

```
[v_1,...v_n]=sscanf (string,format)
```

PARAMETERS :

`format` :Specifies the format conversion. :Specifies the input file name or file number.
`string` :Specifies input to be read.

DESCRIPTION :
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 : `printf` 84, `read` 88, `scanf` 92, `fscanf` 34

1.0.181 stacksize — set scilab stack size

CALLING SEQUENCE :

```
stacksize(n)
sz=stacksize()
```

PARAMETERS :

`n` : integer, the required stack size given in number of double precision words
`sz` : 2-vector [total used]

DESCRIPTION :
Scilab stores all variables in a unique stack `stk`.
`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.
This function with the `n` argument may only be called at the main prompt; it cannot be called within a scilab function.
`sz=stacksize()` returns a 2-vector which contains the current total and used stack size. It can be used everywhere.

SEE ALSO : `who` 109

1.0.182 star — - multiplication operator

DESCRIPTION :
Multiplication. Usual meaning. Valid for constant, boolean, polynomial and rational matrices.
Element-wise multiplication is denoted `x.*y`. If `x` or `y` is scalar (1x1 matrix) `.*` is the same as `*`.
Kronecker product is `x.*.y`

SEE ALSO : `mulf` 219

1.0.183 startup — startup file

DESCRIPTION :
The startup files `.scilab` (in your home directory) and `.scilab` in your working directory are automatically executed (if present) when Scilab is invoked, in addition with the file `scilab.star` in the Scilab directory.

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

1.0.184 str2code — return scilab integer codes associated with a character string

CALLING SEQUENCE :

```
c=str2code(str)
```

PARAMETERS :

`str` : a character string

`c` : vector of character integer codes

DESCRIPTION :
Return c such that c(i) is the scilab integer code of part(str,i))

EXAMPLE :

```
str2code('Scilab')
```

SEE ALSO : `code2str` 13

1.0.185 string — conversion to string

CALLING SEQUENCE :

```
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` 79, `length` 52, `quote` 85, `evstr` 21, `execstr` 23, `strsubst` 238, `strcat` 237, `strindex` 237, `sci2exp` 224

1.0.186 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`.

EXAMPLE :

```
['this','is'; 'a 2x2','matrix']
"matrix"=="mat"+"rix"
```

SEE ALSO : `part` 79, `length` 52, `strcat` 237

1.0.187 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
:	column
,	comma
;	semi
^	hat
.^	power
|	or
&	and
.*.,./., .\.	kron

SEE ALSO : `overloading` 76

1.0.188 testmatrix ______ generate some particular matrices

CALLING SEQUENCE :

```
[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)`).

1.0.189 then ______ keyword in if-then-else

DESCRIPTION :
Used with `if`.

SEE ALSO : `if` 45

1.0.190 tilda ______ - logical not

CALLING SEQUENCE :

```
~m
```

PARAMETERS :

`m` : boolean matrix

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

1.0.191 tlist ______ Scilab object and typed list definition.

CALLING SEQUENCE :

```
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) or Sys('A') is the state-matrix and Sys('td') 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) or H('num') is the transfer matrix numerator

SEE ALSO : `null` 74, `percent` 80, `syslin` 241, `list` 56

1.0.192 type ______ variable type

CALLING SEQUENCE :

```
[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.
8 : matrix of integers stored on 1 2 or 4 bytes
10 : matrix of character strings.
11 : un-compiled function.
13 : compiled function.
14 : function library.
15 : list.
16 : typed list (tlist)
128 : pointer

SEE ALSO : `typeof` 246

1.0.193 typename ______ associates a name to variable type

CALLING SEQUENCE :

```
[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 informations 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` 103, `typeof` 246, `overloading` 76, `tlist` 102, `mlist` 67

1.0.194 ulink ______ unlink a dynamically linked shared object

CALLING SEQUENCE :

```
ulink(x)
```

DESCRIPTION :
see `link`

SEE ALSO : `link` 54

1.0.195 unix_g shell (sh) command execution, output redirected to a variable

CALLING SEQUENCE :

```
rep=unix_g(cmd)
```

PARAMETERS :

`cmd` : a character string
`rep` : a column vector of character strings

DESCRIPTION :
Sends a string `cmd` to Unix for execution by the sh shell. The standard output is redirected to scilab variable `rep`. 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.

EXAMPLE :

```
if MSDOS then unix_g('dir '+WSCI+'\demos');
else unix_g("ls $SCI/demos"); end
deff('wd=pwd()','if MSDOS then wd=unix_g(''cd'');..
                 else wd=unix_g(''pwd''); end')
wd=pwd()
```

SEE ALSO : `edit` 201, `manedit` 216, `unix_s` 105, `unix_w` 105, `unix_x` 106, `unix` 104

1.0.196 unix ______ shell (sh) command execution

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
unix("ls $SCI/demos");
deff('wd=dir()','if MSDOS then unix(''cd>''+TMPDIR+''\path'');..
                else unix(''pwd>''+TMPDIR+''/path'');end..
      wd=read(TMPDIR+''/path'',1,1,''(a)'')')
wd=dir()
```

SEE ALSO : edit 201, manedit 216, unix_g 104, unix_s 105, unix_w 105, unix_x 106, host 43

1.0.197 unix_s ______ shell (sh) command execution, no output

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
if MSDOS then unix_s("del foo");
else unix_s("rm foo"); end
```

SEE ALSO : edit 201, manedit 216, unix_g 104, unix_w 105, unix_x 106, unix 104

1.0.198 unix_w _ shell (sh) command execution, output redirected to scilab window

CALLING SEQUENCE :

```
rep=unix_w(cmd)
```

PARAMETERS :

cmd : a character string
rep : a column vector of character strings

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.

EXAMPLE :

```
if MSDOS then unix_w("dir "+WSCI+"\demos");
else unix_w("ls $SCI/demos"); end
```

SEE ALSO : edit 201, manedit 216, unix_g 104, unix_s 105, unix_x 106, unix 104

1.0.199 unix_x . shell (sh) command execution, output redirected to a window

CALLING SEQUENCE :

```
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 xless 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.

EXAMPLE :

```
if MSDOS then unix_x("dir "+WSCI+"\demos");
else unix_x("ls $SCI/demos"); end
```

SEE ALSO : edit 201, manedit 216, unix_g 104, unix_s 105, unix_w 105, unix 104

1.0.200 user ______ interfacing a fortran routine

CALLING SEQUENCE :

```
[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 31, link 54

1.0.201 varargin ______ variable numbers of arguments in an input argument list

SYNTAX :
varargin must be the rightmost argument of the function definition input list.

DESCRIPTION :
A function whose input argument list contains varargin must 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)

EXAMPLE :

```
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` 36, `varargout` 107, `list` 56

1.0.202 varargout ______ variable numbers of arguments in an output argument list

SYNTAX :
`varargout` must be the rightmost argument of the function definition output 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

`varagout = function ex()` may be called with any number of output arguments. Within function `ex` output arguments may be stored in in `varargout(i)`.

EXAMPLE :

```
deff('varargout=exampl()','varargout=list(1,2,3,4)')

x=exampl()
[x,y]=exampl()
[x,y,z]=exampl()
```

SEE ALSO : `function` 36, `varargin` 106, `list` 56

1.0.203 varn ______ symbolic variable of a polynomial

CALLING SEQUENCE :

```
[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).

EXAMPLE :

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

`varn(p)` is the string `'s'` and `varn(p,'x')` is the polynomial matrix `[x^2+1,x]`

SEE ALSO : `horner` 414, `poly` 81

1.0.204 warning ______ warning messages

CALLING SEQUENCE :

```
warning('string')
```

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

SEE ALSO : `error` 21

1.0.205 what ______ list the Scilab primitives

DESCRIPTION :
List of low level primitives and commands.

1.0.206 whereami ______ display current instruction calling tree

CALLING SEQUENCE :

```
whereami()
```

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

EXAMPLE :

```
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` 109, `pause` 79

1.0.207 whereis ______ name of library containing a function

CALLING SEQUENCE :

```
[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` 53

1.0.208 where ______ get current instruction calling tree

CALLING SEQUENCE :

```
[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` 108, `pause` 79

1.0.209 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 "do" keyword must be on the same line line as "while" keyword.
- The "keyword "then" or "do" can be replaced by a carriage return or a comma.
- 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.

EXAMPLE :

```
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` 30, `select` 94, `break` 9, `return` 90, `pause` 79

1.0.210 who ______ listing of variables

CALLING SEQUENCE :

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


DESCRIPTION :
`who` displays current variable names.
`who('local')` or `who('get')` Returns current variable names and memory used in double precision worlds.
`who('global')` returns global variable names and memory used in double precision worlds.

SEE ALSO : `whos` 110

1.0.211 whos ______ listing of variables in long form

CALLING SEQUENCE :

```
whos()
```

DESCRIPTION :
`who` displays current variable names, types and memory used

SEE ALSO : `who` 109

1.0.212 writb ______ fortran file binary write

CALLING SEQUENCE :

```
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` 28, `readb` 88, `write` 111, `mput` 69, `write4b` 110

1.0.213 write4b ______ fortran file binary write

CALLING SEQUENCE :

```
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` 28, `readb` 88, `write` 111, `mput` 69, `read4b` 87

1.0.214 write — write in a formatted file

DESCRIPTION :

```
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.

EXAMPLE :

```
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` 28, `writb` 110, `read` 88, `print` 85, `string` 100, `fprintf` 33, `printf` 84, `sprintf` 97

Chapter 2

Graphic Library

2.0.215 addcolor _____ add new colors to the current colormap

CALLING SEQUENCE :

```
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` 118

2.0.216 black _____ Black's diagram (Nichols chart)

CALLING SEQUENCE :

```
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` if `sl` is discrete-time.

EXAMPLE :

```
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))
chart();
sstr='(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)';
black(h,0.01,100,sstr);
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
xbasc()
black([h1;h],0.01,100,['h1';'h'])
```

SEE ALSO : `bode` 115, `nyquist` 138, `chart` 117, `freq` 274, `repfreq` 291, `calfrq` 260, `phasemag` 289

2.0.217 bode ______ Bode plot

CALLING SEQUENCE :

```
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` if `sl` is discrete-time. Automatic discretization of frequencies is made by `calfrq`.

EXAMPLE :

```
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))
title='(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)';
bode(h,0.01,100,title);
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
xbasc()
bode([h1;h],0.01,100,['h1';'h'])
```

SEE ALSO : `black` 114, `nyquist` 138, `gainplot` 127, `repfreq` 291, `g_margin` 276, `p_margin` 289, `calfrq` 260, `phasemag` 289

2.0.218 champ1 ______ 2D vector field plot with colored arrows

CALLING SEQUENCE :

```
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.
`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.

EXAMPLE :

```
xset("use color",1)
champ1(-5:5,-5:5,rand(11,11),rand(11,11),2,[-10,-10,10,10],"021")
```

SEE ALSO : `champ` 116, `fchamp` 123, `plot2d` 142

AUTHOR : J.Ph.C.

2.0.219 champ ______ 2D vector field plot

CALLING SEQUENCE :

```
champ(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.
`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.

EXAMPLE :

```
// using rect as plot boundaries
champ(-5:5,-5:5,rand(11,11),rand(11,11),1,[-10,-10,10,10],"011")
// using (x,y) to get boundaries
xbasc()
champ(-5:5,-5:5,rand(11,11),rand(11,11),2,[-10,-10,10,10],"021")
```

See Also : `champ1` 116, `fchamp` 123, `plot2d` 142

Author : J.Ph.C.

2.0.220 chart ____ Nichols chart

CALLING SEQUENCE :

```
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 number (see plot2d) for gain curves
`cphi` : color number (see plot2d) for phase curves

DESCRIPTION :
plot the Nichols'chart.
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)]`

EXAMPLE :

```
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,'(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)')
chart(list(1,0,2,3));
```

Another example :

```
xbasc()
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'])
chart([-8 -6 -4],[80 120],list(1,0));
```


2.0.221 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 by using `xset("colormap",cmap)`.

Each color in the colormap has an id you have to use to specify color in most plot functions. To see the ids, use `xset()` or `xgetcolor()`.

You can come back to default colormap with `xset("default")`.

EXAMPLE :

```
m=228;
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)
plot3d1()
xset("default")
```

See Also : `addcolor` 114, `getcolor` 129, `xset` 174

2.0.222 contour2d ____ level curves of a surface on a 2D plot

CALLING SEQUENCE :

```
contour2d(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.
- 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 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.

Use `contour` to draw levels curves on a 3D surface.

Enter the command `contour2d()` to see a demo.

EXAMPLE :

```
contour2d(1:10,1:10,rand(10,10),5,1:5,"011"," ",[0,0,11,11])
// changing the format of the printing of the levels
xset("fpf","%.2f")
xbasc()
contour2d(1:10,1:10,rand(10,10),5,1:5,"011"," ",[0,0,11,11])
```

SEE ALSO : contour 119, fcontour 124, fcontour2d 124, plot2d 142, xset 174

AUTHOR : J.Ph.C.

2.0.223 contour — level curves on a 3D surface

CALLING SEQUENCE :

```
contour(x,y,z,nz,[theta,alpha,leg,flag,ebox,zlev])
```

PARAMETERS :

`x,y` : two real row vectors of size n1 and n2.
`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.
`-` If `nz` is a vector, `nz(i)` gives the value of the ith level curve.
`theta,alpha,leg,flag,ebox` : see `plot3d`.
`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.

Usually we use `contour2d` to draw levels curves on a 2D plot.

Enter the command `contour()` to see a demo.

EXAMPLE :

```
t=%pi*[-10:10]/10;
deff("[z]=surf(x,y)","z=sin(x)*cos(y)"); z=feval(t,t,surf);
rect=[-%pi,%pi,-%pi,%pi,-1,1];
contour(t,t,z,10,35,45," ",[0,1,0],rect)
// changing the format of the printing of the levels
xset("fpf","%.2f")
xbasc()
contour(t,t,z,10,35,45," ",[0,1,0],rect)
```

SEE ALSO : contour2d 118, fcontour 124, fcontour2d 124, plot3d 144, xset 174

AUTHOR : J.Ph.C.

2.0.224 driver — select a graphics driver

CALLING SEQUENCE :

```
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` or `xbasimp`. 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 with recording of all the graphics commands. This is the default driver.
`"Fig"` : output into XFig format. Clipping of objects is not provided in XFig.
`"GIF"` : output into Gif format (beta test driver written by Tom Leitner (http://wiis.tu-graz.ac.at/people/tom.html) Note that line thickness is not handled yet.

SEE ALSO : xtape 178, xbasc 156, xbasimp 157

AUTHOR : J.Ph.C.

2.0.225 edit_curv — interactive graphic curve editor

CALLING SEQUENCE :

```
[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

AUTHOR : Serge Steer

2.0.226 errbar ________ add vertical error bars on a 2D plot

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
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` 142

AUTHOR : J.Ph.C.

2.0.227 eval3d ________ values of a function on a grid

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
  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` 27

AUTHOR : Steer S.

2.0.228 eval3dp ________ compute facets of a 3D surface

CALLING SEQUENCE :

```
[x,y,z]=eval3dp(fun,p1,p2)
```

PARAMETERS :

`x,y,z` : matrices of size (4,n-1*m-1). `x(:,i)` ,`y(:,i)` and `z(:,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 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.

EXAMPLE :

```
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)"])
[x,y,z]=eval3dp(scp,p1,p2);
plot3d(x,y,z)
```

SEE ALSO : `genfac3d` 128, `plot3d` 144

2.0.229 evans ________ Evans root locus

CALLING SEQUENCE :

```
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`.

EXAMPLE :

```
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
xbasc();s=poly(0,'s');n=1+s;
d=real(poly([-1 -2 -%i %i],'s'));
evans(n,d,100);
//
xbasc();n=real(poly([0.1-%i 0.1+%i,-10],'s'));
evans(n,d,80);
```

SEE ALSO : kpure 278, krac2 278, locate 136

2.0.230 fac3d _ 3D plot of a surface (obsolete)

CALLING SEQUENCE :

```
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 144, plot3d1 143

2.0.231 fchamp _ direction field of a 2D first order ODE

CALLING SEQUENCE :

```
fchamp(f,t,xr,yr,[arfact,rect,strf])
```

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,x,[u])`. `f` returns a column vector of size 2, `y`, which gives the value of the direction field `f` at point `x` and at time `t`.
- It can also be an object of type list, `list(f,u1)` where `f` is a function of type `y=f(t,x,u)` and `u1` gives the value of the parameter `u`.

`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.
`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.

EXAMPLE :


```
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)
xbasc()
fchamp(derpol,0,xf,yf,1,[-2,-2,2,2],"011")
```

SEE ALSO : champ 116, champ1 116

AUTHOR : J.Ph.C.

2.0.232 fcontour2d _ level curves of a surface defined by a function on a 2D plot

CALLING SEQUENCE :

```
fcontour2d(xr,yr,f,nz,[style,strf,leg,rect,nax])
```

PARAMETERS :

`xr,yr` : two real row vectors of size n1 and n2.
`f` : is an external which defines the surface `z=f(x,y)`. It is first computed on the grid specified by `xr,yr`. Then, control is passed to the routine `contour2d`.
`nz,[style,strf,leg,rect,nax]` : see `contour2d`.

DESCRIPTION :

Draws level curves of a surface `z=f(x,y)`. The level curves are drawn on a 2D plot. The surface is given by the external function `f`. See `contour2d`.

Enter the command `fcontour2d()` to see a demo.

SEE ALSO : contour 119, contour2d 118, fcontour 124

AUTHOR : J.Ph.C.

2.0.233 fcontour _ level curves on a 3D surface defined by a function

CALLING SEQUENCE :

```
fcontour(xr,yr,f,nz,[theta,alpha,leg,flag,ebox,zlev])
```

PARAMETERS :

`xr,yr` : two real row vectors of size n1 and n2.
`f` : is an external which defines the surface `z=f(x,y)`. It is first computed on the grid specified by `xr,yr`. Then, control is passed to the routine `contour`.
`nz,[theta,alpha,leg,flag,ebox,zlev]` : see `contour`.

DESCRIPTION :

Draws level curves of a surface `z=f(x,y)`. The level curves are drawn on a 3D surface. The surface is given by the external function `f`. See `contour`.

Enter the command `fcontour()` to see a demo.

SEE ALSO : contour 119, contour2d 118, fcontour2d 124

AUTHOR : J.Ph.C.

2.0.234 fec _ contour level of a function defined on a triangular mesh

CALLING SEQUENCE :

```
fec(x,y,triangles,func,no,Ntr,strflag,legend,brect,aaint)
```

PARAMETERS :

`x,y` : two vectors of size `no`, `(x(i),y(i))` gives the coordinates of node `i`
`func` : a vector of size `no` : `func(i)` gives the value of the function for which we want the level curves.
`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
`rest` : for the remaining arguments `strflag,legend,brect,aint`, see plot2d

DESCRIPTION :
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.

SEE ALSO : `Sfgrayplot` 151, `Sgrayplot` 151

2.0.235 fgrayplot ___ 2D plot of a surface defined by a function using colors

CALLING SEQUENCE :

```
fgrayplot(x,y,f,[strf,rect,nax])
```

PARAMETERS :

`x,y` : real row vectors.
`f` : external of type `y=f(x,y)`.
`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.

EXAMPLE :

```
t=-1:0.1:1;
deff("[z]=surf(x,y)","z=x**2+y**2")
fgrayplot(t,t,surf,"111",[-2,-2,2,2])
```

SEE ALSO : `grayplot` 133, `plot2d` 142, `Sgrayplot` 151, `Sfgrayplot` 151

AUTHOR : J.Ph.C.

2.0.236 fplot2d ____________ 2D plot of a curve defined by a function

CALLING SEQUENCE :

```
fplot2d(xr,f,[style,strf,leg,rect,nax])
```

PARAMETERS :

`xr` : vector.
`f` : external of type `y=f(x)`.
`style,strf,leg,rect,nax` : 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))`.

Enter the command `fplot2d()` to see a demo.

EXAMPLE :

```
deff("[y]=f(x)","y=sin(x)+cos(x)")
x=[0:0.1:10]*%pi/10;
fplot2d(x,f)
```

SEE ALSO : `plot2d` 142

AUTHOR : J.Ph.C.

2.0.237 fplot3d1 . 3D gray or color level plot of a surface defined by a function

CALLING SEQUENCE :

```
fplot3d1(xr,yr,f,[theta,alpha,leg,flag,ebox])
```

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`.

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.

SEE ALSO : `plot3d1` 143

AUTHOR : J.Ph.C.

2.0.238 fplot3d — 3D plot of a surface defined by a function

CALLING SEQUENCE :

```
fplot3d(xr,yr,f,[theta,alpha,leg,flag,ebox])
```

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`.

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.

SEE ALSO : `plot3d` 144

AUTHOR : J.Ph.C.

2.0.239 gainplot — magnitude plot

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
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)')
xbasc()
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` 115, `black` 114, `nyquist` 138, `freq` 274, `repfreq` 291, `g_margin` 276, `p_margin` 289

2.0.240 genfac3d — compute facets of a 3D surface

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
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` 122, `plot3d` 144

2.0.241 geom3d — projection from 3D on 2D after a 3D plot

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
deff("[z]=surf(x,y)","z=sin(x)*cos(y)")
t=%pi*(-10:10)/10;
// 3D plot of the surface
fplot3d(t,t,surf,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(t,t,sin(t).*cos(t));
xpoly(x,y,"lines")
// adding a comment
[x,y]=geom3d([0,0],[0,0],[5,0]);
xsegs(x,y)
xstring(x(1),y(1),"point (0,0,0)")
```

AUTHOR : J.Ph.C.

2.0.242 getcolor ______ dialog to select colors in the current colormap

CALLING SEQUENCE :

```
c=getcolor(title,[cini])
c=getcolor()
```

PARAMETERS :

`title` : string, dialog title.
`cini` : vector of initial selected color ids. Default value is xget("pattern").
`c` : vector of selected color ids, or [] if the user has clicked on the "Cancel" button.

DESCRIPTION :
`getcolor` opens a dialog choice box with as many palettes as `cini` vector size. Palettes depend on the current colormap.

SEE ALSO : `xset` 174

2.0.243 getsymbol ______ dialog to select a symbol and its size

CALLING SEQUENCE :

```
c=getsymbol([title])
```

PARAMETERS :

`title` : string, dialog title.
`c` : vector of size 2 `[n,sz]`.

DESCRIPTION :
`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 : `xset` 174

2.0.244 graduate ______ pretty axis graduations

CALLING SEQUENCE :

```
[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`

EXAMPLE :


```
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` 173, `plot2d` 142

AUTHOR : S. Steer 1992

2.0.245 Graphics ______ graphics library overview

2D PLOTTING :

`plot` : simple plot of a curve
`plot2d` : plot a curve
`plot2d1` : plot a curve, allows logarithmic axes
`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
`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
`xbasimp` : send graphics to a Postscript printer or in a file
`xs2fig` : send graphics to a file in Xfig 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
`xclea` : erase a rectangle
`xclip` : set a clipping zone
`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

2.0.246 graycolormap _____ linear gray colormap

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
xset("colormap",graycolormap(32))
plot3d1()
```

SEE ALSO : `colormap` 118, `hotcolormap` 135, `xset` 174

2.0.247 grayplot ______ 2D plot of a surface using colors

CALLING SEQUENCE :

```
grayplot(x,y,z,[strf,rect,nax])
```

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)).
`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.

Enter the command `grayplot()` to see a demo.

EXAMPLE :

```
x=-10:10; y=-10:10;m =rand(21,21);
grayplot(x,y,m,"111",[-20,-20,20,20])
t=-%pi:0.1:%pi; m=sin(t)'*cos(t);
xbasc()
grayplot(t,t,m)
```

SEE ALSO : `fgrayplot` 125, `plot2d` 142, `Sgrayplot` 151, `Sfgrayplot` 151

AUTHOR : J.Ph.C.

2.0.248 gr_menu ______ simple interactive graphic editor

CALLING SEQUENCE :

```
[sd1]=gr_menu([sd,flag,no_frame])
```

PARAMETERS :

`sd` : list (output of `gr_menu`), or vector of length four `[xmin,ymin,xmax,ymax]` (boundaries of the plot).
`sd1` : list (graphical objects created under `gr_menu`
`flag,noframe` : integers with 0, 1 value. Use `flag=1` for non interactive mode (i.e to redraw saved `gr_menu` graphics) and `no_frame=1` to avoid a frame around `gr_menu` graphics.

DESCRIPTION :
`gr_menu` is a simple interactive graphic editor. When you execute `gr_menu()`, three new menus, `Objects`, `Settings` and `Edit` are added to the current graphics window. Use the item `Exit` of menu `Edit` to exit `gr_menu`.

The created graphics are saved as a list which can be given to `gr_menu` as an entry value.

`[sd]=gr_menu([xmin,ymin,xmax,ymax])` : enters `gr_menu` with a given frame
`[sd]=gr_menu();` : enters `gr_menu` with the frame [0 0 100 100].
`[sd]=gr_menu(sd)` : redraws the graphics stored in sd and enters interactive mode
`[sd]=gr_menu(sd,1)` : only draws the graphics stored in `sd`.
`[sd]=gr_menu(sd,1,1)` : only draws the graphics stored in `sd` and no frame is added.

AUTHOR : S.S. & J.Ph.C.

2.0.249 hist3d ______ 3D representation of a histogram

CALLING SEQUENCE :

```
hist3d(f,[theta,alpha,leg,flag,ebox])
hist3d(list(f,x,y),[theta,alpha,leg,flag,ebox])
```

PARAMETERS :

`f` : matrix of size (m,n) defining the histogram `f(i,j)=F(x(i),y(j))`, where `x` and `y` are taken as `0:m` and `0:n`.
`list(f,x,y)` : where f is a matrix of size (m,n)defining the histogram `f(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.

SEE ALSO : `histplot` 134, `plot3d` 144

AUTHOR : Steer S. & JPhilippe C.

2.0.250 histplot ______ plot a histogram

CALLING SEQUENCE :

```
histplot(npoint,data,[style,strf,leg,rect,nax])
```

PARAMETERS :

`npoint` : integer or a row vector of increasing values.
`data` : real vector.
`style,strf,leg,rect,nax` : see `plot2d`.

DESCRIPTION :

- If `npoint` is an integer, `histplot` plots a histogram of the values stored in `data` using `npoint` equally spaced classes.
- If `npoint` is a vector `histplot` plots a histogram of the values stored in `data` using the classes `]npoint(k),npoint(k+1)]`.

Enter the command `histplot()` to see a demo.

SEE ALSO : `hist3d` 134, `plot2d` 142

2.0.251 hotcolormap ______ red to yellow colormap

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
xset("colormap",hotcolormap(32))
plot3d1()
```

SEE ALSO : `colormap` 118, `graycolormap` 132, `xset` 174

2.0.252 isoview _ set scales for isometric plot (do not change the size of the window)

CALLING SEQUENCE :

```
isoview(xmin,xmax,ymin,ymax)
```

PARAMETERS :

`xmin,xmax,ymin,ymax` : four real values

DESCRIPTION :
`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`).

EXAMPLE :

```
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")
```

SEE ALSO : `square` 152, `xsetech` 173

AUTHOR : Steer S.

2.0.253 locate ______ mouse selection of a set of points

CALLING SEQUENCE :

```
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` 159, `xgetmouse` 164

AUTHOR : S.S. & J.Ph.C

2.0.254 Matplot ______ 2D plot of a matrix using colors

CALLING SEQUENCE :

```
Matplot(a,[strf,rect,nax])
```

PARAMETERS :

`a` : real matrix of size (n1,n2).
`strf,rect,nax` : optional arguments, see `plot2d`.

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 length 1 with center at location `(x=j,y=(n2-i+1))`.

Enter the command `Matplot()` to see a demo.

EXAMPLE :

```
Matplot([1 2 3;4 5 6])
// draw the current colormap
Matplot((1:xget("lastpattern")))
```

SEE ALSO : `colormap` 118, `plot2d` 142

AUTHOR : J.Ph.C.

2.0.255 m_circle — M-circle plot

CALLING SEQUENCE :

```
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` is used with `nyquist`.

EXAMPLE :

```
//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` 138, `chart` 117, `black` 114

AUTHOR : S.Steer.

2.0.256 milk_drop — milk drop 3D function

CALLING SEQUENCE :

```
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` adn `plot3d`.

EXAMPLE :

```
x=-2:0.1:2; y=x;
z=eval3d(milk_drop,x,y);
plot3d(x,y,z)
```

SEE ALSO : `eval3d` 121, `plot3d` 144

AUTHOR : Steer S.

2.0.257 nyquist — nyquist plot

CALLING SEQUENCE :

```
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` if `sl` is discrete-time.
Automatic discretization of frequencies is made by `calfrq`.

EXAMPLE :

```
xbasc();
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))
xbasc();
nyquist([h1;h],0.01,100,['h1';'h'])
xbasc();nyquist([h1;h])
```

SEE ALSO : `bode` 115, `black` 114, `calfrq` 260, `freq` 274, `repfreq` 291, `phasemag` 289

2.0.258 param3d1 — 3D plot of curves

CALLING SEQUENCE :

```
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))+1`; 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 mark and color ids.

`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=[type,box]` : `type` and `box` have the same meaning as in `plot3d`.
`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 :

`param3d1` is used to plot 3D curves defined by their coordinates `x`, `y` and `z`.

Enter the command `param3d1()` to see a demo.

EXAMPLE :

```
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])
```

SEE ALSO : `param3d` 139, `plot3d` 144, `xset` 174

AUTHOR : J.Ph.C.

2.0.259 param3d ____ 3D plot of a curve

CALLING SEQUENCE :

```
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.
`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`.
`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 :

`param3d` is used to plot a 3D curve defined by its coordinates `x`, `y` and `z`.

Use `param3d1` to do multiple plots.

Enter the command `param3d()` to see a demo.

EXAMPLE :

```
t=0:0.1:5*%pi;
param3d(sin(t),cos(t),t/10,35,45,"X@Y@Z",[2,3])
```

SEE ALSO : `param3d1` 138, `plot3d` 144

AUTHOR : J.Ph.C.

2.0.260 plot2d1 ____ 2D plot (logarithmic axes)

CALLING SEQUENCE :

```
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`.

DESCRIPTION :

`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 `xbasc`.

Enter the command `plot2d1()` to see a demo.

EXAMPLE :

```
// 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
xbasc()
plot2d1("onn",x,[sin(x) sin(2*x) sin(3*x)])
// logarithmic plot
x=[0.1:0.1:3]'; xbasc()
plot2d1("oll",x,[exp(x) exp(x^2) exp(x^3)])
```

SEE ALSO : `plot2d` 142, `plot2d2` 140, `plot2d3` 141, `plot2d4` 141, `xbasc` 156

AUTHOR : J.Ph.C.

2.0.261 plot2d2 ____ 2D plot (step function)

CALLING SEQUENCE :

```
plot2d2(str,x,y,[style,strf,leg,rect,nax])
```

PARAMETERS :
[ ] : see `plot2d1` for a description of parameters.
DESCRIPTION :
`plot2d2` is the same as `plot2d1` 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 `xbasc`.

Enter the command `plot2d2()` to see a demo.
EXAMPLE :

```
// plots a step function of value i on the segment [i,i+1]
// the last segment is not drawn
plot2d2("gnn",[1:4]',[1:4]',1,"111","step function",[0,0,5,5])
// compare the following with plot2d1
x=[0:0.1:2*%pi]';
xbasc()
plot2d2("onn",x,[sin(x) sin(2*x) sin(3*x)])
```

SEE ALSO : `plot2d` 142, `plot2d1` 140, `plot2d3` 141, `plot2d4` 141, `xbasc` 156

AUTHOR : J.Ph.C.

2.0.262 plot2d3 — 2D plot (vertical bars)

CALLING SEQUENCE :

```
plot2d3(str,x,y,[style,strf,leg,rect,nax])
```

PARAMETERS :
[ ] : see `plot2d1` for a description of parameters.
DESCRIPTION :
`plot2d3` is the same as `plot2d1` but curves are plotted using vertical bars.

By default, successive plots are superposed. To clear the previous plot, use `xbasc`.

Enter the command `plot2d3()` to see a demo.
EXAMPLE :

```
// compare the following with plot2d1
x=[0:0.1:2*%pi]';
plot2d3("onn",x,[sin(x) sin(2*x) sin(3*x)])
```

SEE ALSO : `plot2d` 142, `plot2d1` 140, `plot2d2` 140, `plot2d4` 141, `xbasc` 156

AUTHOR : J.Ph.C.

2.0.263 plot2d4 — 2D plot (arrows style)

CALLING SEQUENCE :

```
plot2d4(str,x,y,[style,strf,leg,rect,nax])
```

PARAMETERS :
[ ] : see `plot2d1` for a description of parameters.
DESCRIPTION :
`plot2d4` is the same as `plot2d1` 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 `xbasc`.

Enter the command `plot2d4()` to see a demo.
EXAMPLE :


```
// compare the following with plot2d1
x=[0:0.1:2*%pi]';
plot2d4("onn",x,[sin(x) sin(2*x) sin(3*x)])
```

SEE ALSO : `fchamp` 123, `plot2d` 142, `plot2d1` 140, `plot2d2` 140, `plot2d3` 141, `xbasc` 156

AUTHOR : J.Ph.C.

2.0.264 plot2d — 2D plot

CALLING SEQUENCE :

```
plot2d(x,y,[style,strf,leg,rect,nax])
plot2d(y)
```

PARAMETERS :

`x,y` : two matrices of 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.

For instance, to plot one curve: `x=[1:10]'` and `y=sin(x)`.

For instance, to plot 2 curves: `x1=[1:10]'`, `x2=[11:20]'`, `x=[x1 x2]` and `y=[sin(x1) cos(x1)]`.

`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 `"xyz"`.
`The default is "021".`
`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.
`z` : controls the display of information on the frame around the plot.
`z=0` : nothing is drawn around the plot.
`z=1` : axes are drawn. The number of tics can be specified by the `nax` optional argument.
`z=2` : the plot is surrounded by a box without tics.

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.

DESCRIPTION :

`plot2d` plots a set of 2D curves: each column of `y` is plotted with respect to the corresponding column of `x`. Piecewise linear plotting is used.

By default, successive plots are superposed. To clear the previous plot, use `xbasc`.

Invoked with only one argument, `plot2d(y)` plots all the columns of `y` with respect to the vector 1:<number of rows of `y`>.

Enter the command `plot2d()` to see a demo.

Other high level plot2d function exist:

`plot2d1`: same as `plot2d` but enables logarithmic plots and more facilities to specify `x` vectors.
`plot2d2`: same as `plot2d1` but the curve is supposed to be piecewise constant.
`plot2d3`: same as `plot2d1` but the curve is plotted with vertical bars.
`plot2d4`: same as `plot2d1` but the curve is plotted with arrows.

EXAMPLE :

```
//simple plot
x=[0:0.1:2*%pi]';
plot2d(sin(x))
//multiple plot
xbasc()
plot2d([x x x],[sin(x) sin(2*x) sin(3*x)])
// multiple plot giving the dimensions of the frame
xbasc()
plot2d([x x x],[sin(x) sin(2*x) sin(3*x)],..
  1:3,"011"," ",[0,0,6,0.5])
//multiple plot with captions and given tics
xbasc()
plot2d([x x 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])
// isoview
xbasc()
plot2d(x,sin(x),1,"041")
// auto scale
xbasc()
plot2d(x,sin(x),1,"061")
```

SEE ALSO : plot2d1 140, plot2d2 140, plot2d3 141, plot2d4 141, xbasc 156, xset 174

AUTHOR : J.Ph.C.

2.0.265 plot3d1 — 3D gray or color level plot of a surface

DESCRIPTION :

```
plot3d1(x,y,z,[theta,alpha,leg,flag,ebox])
plot3d1(xf,yf,zf,[theta,alpha,leg,flag,ebox])
```

PARAMETERS :

See `plot3d` for a full description. There is just a slight difference, only the sign of the `flag(1)=mode` parameter is used: if it is negative the grid is not drawn.

DESCRIPTION :

`plot3d` plots a surface `z=f(x,y)` with colors depending on the z-level of the surface.

Enter the command `plot3d1()` to see a demo.

EXAMPLE :

```
// 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);
xbasc()
plot3d1(xx,yy,zz)
// multiple plots
xbasc()
plot3d1([xx xx],[yy yy],[zz 4+zz])
// simple plot with viewpoint and captions
xbasc()
plot3d1(1:10,1:20,10*rand(10,20),35,45,"X@Y@Z",[2,2,3])
// same plot without grid
xbasc()
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));
xbasc()
plot3d1(xx,yy,zz)
```

SEE ALSO : plot3d 144

AUTHOR : J.Ph.C.

2.0.266 plot3d — 3D plot of a surface

CALLING SEQUENCE :

```
plot3d(x,y,z,[theta,alpha,leg,flag,ebox])
plot3d(xf,yf,zf,[theta,alpha,leg,flag,ebox])
plot3d(xf,yf,list(zf,colors),[theta,alpha,leg,flag,ebox])
```

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)`.

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`. `colors(i)` is the color of facet `i`. If `colors(i)` is negative, color id `-colors(i)` is used and the boundary of the facet is drawn. Use `xset()` to see the ids of the colors.

`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 shadow of the surface is 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` the boundaries are specified by the value of the optional argument `ebox`.
`other values` the boundaries are computed using the given data.
`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 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 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.

EXAMPLE :

```
// 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)
```

SEE ALSO : `eval3dp` 122, `genfac3d` 128, `geom3d` 128, `param3d` 139, `plot3d1` 143, `xset` 174

AUTHOR : J.Ph.C.

2.0.267 plotframe ______ plot a frame with scaling and grids

CALLING SEQUENCE :

```
plotframe(rect,tics,[arg_opt1,arg_opt2,arg_opt3])
```

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.
`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`).

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.

EXAMPLE :

```
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,[%t,%f],["My plot with grids","x","y"],[0.5,0,0.5,0.5])
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,tics,[%f,%t],..
["My plot without grids but with automatic bounds","x","y"],..
```

```
[0.5,0.5,0.5,0.5])
plot2d(x,y,5,"000")
xset("default")
```

SEE ALSO : plot2d 142, graduate 129, xsetech 173

2.0.268 plot _____ simple plot

CALLING SEQUENCE :

```
plot(x,y,[xcap,ycap,caption])
plot(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, `plot(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`>.

For more complex plotting, use plot2d.

EXAMPLE :

```
x=0:0.1:2*%pi;
// simple plot
plot(sin(x))
// using captions
xbasc()
plot(x,sin(x),"sin","time","plot of sinus")
// plot 2 functions
xbasc()
plot([sin(x);cos(x)])
```

SEE ALSO : plot2d 142

AUTHOR : J.Ph.C.

2.0.269 plzr _____ pole-zero plot

CALLING SEQUENCE :

```
plzr(sl)
```

PARAMETERS :

`sl` : list (`syslin`)

DESCRIPTION :
produces a pole-zero plot of the linear system `sl` (`syslin` list)

EXAMPLE :


```
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 303, roots 420, syslin 241

2.0.270 printing _____ printing scilab graphics

CALLING SEQUENCE :

```
Blatexpr xscale yscale filename.ps
BEpsf filename.ps
Blpr "Title" filename1.ps filename2.ps ... filenamen.ps | lpr
```

DESCRIPTION :
The scilab graphics can be saved with the `xbasimp` command into unix files. The Scilab command :

```
xbasimp(xx,'des.ps',0)
```

will save the graphics recorded in the graphic window xx in the file `des.ps.xx`. This file can't be directly send to a Postscript printer and a set of programs (in the `bin` Scilab directory) are given with Scilab to print it :

`BEpsf` : The `BEpsf` command will create an Epsf file from your `des.ps.xx` under the name `des.epsf`, this Epsf file can be printed on a Postscript printer or inserted into an other Postscript document.

`Blatexpr` : The `Blatexpr` command will create an Epsf file from your `des.ps.xx`

```
mv des.ps.xx des.ps
Blatexpr 1.0 1.0 des.ps
```

under the name `des.epsf` and a LaTeX file `des.tex`. The file `des.tex` can be inserted in a LaTeX file in order to get the latex figure as follows (the postscript file is inserted with the `special` command of LaTeX)

```
\input des.tex
\dessin{caption}{label}
```

`Blpr` : The `Blpr` command is used to print a set of graphics on a same sheet of paper. For example to print two graphics on a unique page, one can use :

```
Blpr "Two graphics" file1.ps.0 file2.ps.1 | lpr
```

`Blatexprs` : The `Blatexprs` command is used to insert in a single LaTeX figure a set of Scilab Graphics

```
Blatexprs res file1.ps.0 file2.ps.1
```

will create two files `res.ps` and `res.tex`. The file `res.tex` is used as in the `Blatexpr` command in order to get the figure.

SEE ALSO : xbasimp 157

2.0.271 replot ________ redraw the current graphics window with new boundaries

CALLING SEQUENCE :

```
replot(rect)
```

PARAMETERS :

`rect` : row vector of size 4.

DESCRIPTION :

`replot` is used to redraw the content of the current graphics window with new boundaries defined by `rect=[xmin,ymin,xmax,ymax]`. It works only with the driver `"Rec"`.

EXAMPLE :

```
x=[0:0.1:2*%pi]';
plot2d(x,sin(x))
replot([-1,-1,10,2])
```

SEE ALSO : `xbasr` 157

AUTHOR : J.Ph.C.

2.0.272 rotate ________ rotation of a set of points

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
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
```


2.0.273 scaling ________ affine transformation of a set of points

CALLING SEQUENCE :

```
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`.

2.0.274 sd2sci ________ gr_menu structure to scilab instruction convertor

CALLING SEQUENCE :

```
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.

AUTHOR : Serge Steer INRIA 1988

SEE ALSO : `gr_menu` 133, `execstr` 23

2.0.275 secto3d ________ 3D surfaces conversion

CALLING SEQUENCE :

```
[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` 144

AUTHOR : Steer S.

2.0.276 Sfgrayplot smooth 2D plot of a surface defined by a function using colors

CALLING SEQUENCE :

```
Sfgrayplot(x,y,f,[strf,rect,nax])
```

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:

```
 ------
 | /| /|
 |/_|/_|
 | /| /|
 |/_|/_|
```

Enter the command `Sfgrayplot()` to see a demo.

EXAMPLE :

```
t=-1:0.1:1;
deff("[z]=surf(x,y)","z=x**2+y**2")
Sfgrayplot(t,t,surf,"111",[-2,-2,2,2])
```

SEE ALSO : `fec` 125, `fgrayplot` 125, `grayplot` 133, `Sgrayplot` 151

AUTHOR : J.Ph.C.

2.0.277 Sgrayplot smooth 2D plot of a surface using colors

CALLING SEQUENCE :

```
Sgrayplot(x,y,z,[strf,rect,nax])
```

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:


```
 ------
 | /| /|
 |/_|/_|
 | /| /|
 |/_|/_|
```

Enter the command `Sgrayplot()` to see a demo.

EXAMPLE :

```
x=-10:10; y=-10:10;m =rand(21,21);
Sgrayplot(x,y,m,"111",[-20,-20,20,20])
t=-%pi:0.1:%pi; m=sin(t)'*cos(t);
xbasc()
Sgrayplot(t,t,m)
```

SEE ALSO : `fec` 125, `fgrayplot` 125, `grayplot` 133, `Sfgrayplot` 151

AUTHOR : J.Ph.C.

2.0.278 sgrid s-plane grid lines.

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
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` 122

2.0.279 square set scales for isometric plot (change the size of the window)

CALLING SEQUENCE :

```
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`).

EXAMPLE :

```
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` 135, `xsetech` 173

AUTHOR : Steer S.

2.0.280 titlepage — add a title in the middle of a graphics window

CALLING SEQUENCE :

```
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` 178

AUTHOR : S. S.

2.0.281 winsid — return the list of graphics windows

CALLING SEQUENCE :

```
x=winsid()
```

PARAMETERS :

`x` : row vector.

DESCRIPTION :
`winsid` is used to get the list of graphics windows as a vector of windows numbers.

2.0.282 xarc — draw a part of an ellipse

CALLING SEQUENCE :

```
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 style and scale.

EXAMPLE :

```
// isoview scaling
plot2d(0,0,-1,"031"," ",[-2,-2,2,2])
xset("dashes",3)
xarc(-1,1,2,2,0,90*64)
xarc(-1.5,1.5,3,3,0,360*64)
```

SEE ALSO : `xarcs` 154, `xfarc` 161, `xfarcs` 161

AUTHOR : J.Ph.C.

2.0.283 xarcs — draw parts of a set of ellipses

CALLING SEQUENCE :

```
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`). `style(i)` gives the dash style used to draw ellipse number `i`.

EXAMPLE :

```
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` 154, `xfarc` 161, `xfarcs` 161

AUTHOR : J.Ph.C.

2.0.284 xarrows ________ draw a set of arrows

CALLING SEQUENCE :

```
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 dash style to use for all arrows. If it is a negative scalar then the current dash style is used. If it is a vector `style(i)` gives the style 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 ith 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`.

EXAMPLE :

```
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")
```

AUTHOR : J.Ph.C.

2.0.285 xaxis ________ draw an axis

CALLING SEQUENCE :

```
xaxis(alpha,nsteps,size,init)
```

PARAMETERS :

`alpha` : real, slope in degree of the axis.
`nsteps` : real vector of size 2, number of big and small intervals.
`size` : real vector of size 3, size of the small intervals, and small and big tics.
`init` : real vector of size 2, origin of the axis.

DESCRIPTION :

`xaxis` draws an axis.

The direction of the axis is given by `alpha` in degree.

`init=[x0 y0]` is the initial point of the axis.

`nsteps=[n1,n2]` gives the number of big and small intervals separated by tics.

`size=[s1,s2,c1]` where `s1` gives the size of the small intervals, `s2` gives the size of the small tics along the axis and `s2*c1` gives the size of the big tics. All the sizes are given using the current x-scale and y-scale and are given as dimensions along the drawn axis.

```
example : n1=3, n2=2, alpha=0
     (s2*c1)
|     (s2)   |            |           |
|______|_____|_____|______|_____|_____|
   s1
```

EXAMPLE :

```
x=[-%pi:0.1:%pi]';
// plot without axis
plot2d(x,sin(x),1,"010"," ",[-4 -1 4 1])
// draw x axis
xpoly([-4 4],[0 0],"lines")
xaxis(0,[2 2],[2 0.1 3],[-4 0])
xstring(-4.1,-0.25,"-4"); xstring(-0.2,-0.1,"0"); xstring(4,-0.25,"4")
// draw y axis
xpoly([0 0],[-1 1],"lines")
xaxis(90,[2 2],[0.5 0.025 3],[0 1])
xstring(-0.5,-1.05,"-1"); xstring(-0.35,0.95,"1")
```

AUTHOR : J.Ph.C.

2.0.286 xbasc ____ clear a graphics window and erase the associated recorded graphics

CALLING SEQUENCE :

```
xbasc([window-id])
```

PARAMETERS :

`window-id` : integer scalar or vector

DESCRIPTION :

Without any argument, this function clears the current graphics window and erases the recorded graphics. Otherwise it clears the graphics windows whose numbers are included in the vector `window-id`, and erases the corresponding recorded graphics. For example `xbasc(1:3)` clears windows 1, 2 and 3 and erases the corresponding recorded graphics. If one of the windows does not exist, then it is automatically created.

SEE ALSO : `xclear` 158

2.0.287 xbasimp ____ send graphics to a Postscript printer or in a file

CALLING SEQUENCE :

```
xbasimp(win_num,[filen,printer])
```

PARAMETERS :

`win_num` : integer scalar or vector
`filen` : string, Postscript file name (default value is "file"). The window number is appended to `filen`.
`printer` : string, printer name. If `printer` is present or if there is only one argument in the calling sequence, the created file is printed on printer `printer`.

DESCRIPTION :
`xbasimp` sends the recorded graphics of the window `win_num` into the Postscript file `filen` and prints the Postscript file with the command `Blpr`. This function works only if the selected driver is `"Rec"`.
If `win_num` is a vector, several files are generated, one for each selected window (with names `filenxx`), and the files are printed on a unique page with the command `Blpr`.
The window number is appended to `filen`.

SEE ALSO : `printing` 148, `xs2fig` 171

2.0.288 xbasr ____ redraw a graphics window

CALLING SEQUENCE :

```
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` 120, `replot` 149, `xtape` 178

AUTHOR : J.Ph.C.

2.0.289 xchange ____ transform real to pixel coordinates

CALLING SEQUENCE :

```
[x1,y1,rect]=xchange(x,y,dir)
```

PARAMETERS :

`x,y` : two matrices of size (n1,n2) (coordinates of a set of points).
`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].

EXAMPLE :


```
t=[0:0.1:2*%pi]';
plot2d(t,sin(t))
[x,y,rect]=xchange(1,1,"f2i")
[x,y,rect]=xchange(0,0,"i2f")
```

AUTHOR : J.Ph.C.

2.0.290 xclea ____ erase a rectangle

CALLING SEQUENCE :

```
xclea(x,y,w,h)
```

PARAMETERS :

`x,y,w,h` : real values defining the rectangle.

DESCRIPTION :
`xclea` clears the rectangle `[x,y,w,h]` (upper left point, width, height) in the current graphics window.

EXAMPLE :

```
x=[0:0.1:2*%pi]';
plot2d(x,sin(x))
xclea(1,1,1,1)
```

AUTHOR : J.Ph.C.

2.0.291 xclear ____ clear a graphics window

CALLING SEQUENCE :

```
xclear([window-id])
```

PARAMETERS :

`window-id` : integer scalar or vector

DESCRIPTION :
Without any argument, this function clears the current window. Otherwise it clears the graphics windows 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.
Warning: in recording mode `xclear` clears the window, but it does not erase the recorded commands. In this case you must use the function `xbasc`.

SEE ALSO : `xbasc` 156

AUTHOR : J.Ph.C.

2.0.292 xclick ____ wait for a mouse click

CALLING SEQUENCE :

```
[c_i,c_x,c_y,c_w,c_m]=xclick([flag])
```

PARAMETERS :

`c_i` : integer, mouse button number.
`c_x,c_y` : real scalars, position of the mouse.
`c_w` : integer, window number.
`c_m` : string, menu callback.
`flag` : integer. 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 returned values are described below.

`c_i` : an integer which gives the number of the mouse button that was pressed 0, 1 or 2 (for left, middle and right) or -1 in case of problems with xclick.
`c_x,c_y` : the coordinates of the position of the mouse click in the current graphics scale.
`c_w` : the window number where the click has occurred.
`c_m` : string associated with a dynamic menu. If xclick returns due to a click on a menu, `c_i`, `c_x`, `c_y`, and `c_w` take arbitrary values.

KNOWN TROUBLES :
`xclick` can return the message "Can't grab the pointer" if the graphics window is iconified when calling it.

SEE ALSO : `locate` 136, `xgetmouse` 164

AUTHOR : J.Ph.C.

2.0.293 xclip ____ set a clipping zone

CALLING SEQUENCE :

```
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()`.

EXAMPLE :


```
x=0:0.2:2*%pi;
x1=[sin(x);100*sin(x)];
y1=[cos(x);100*cos(x)];
y1=y1+20*ones(y1);
// No clip
plot2d([-100,500],[-100,600],[-1,-1],"022")
xsegs(10*x1+200*ones(x1),10*y1+200*ones(y1))
// rectangle clipping zone
xbasc(); plot2d([-100,500],[-100,600],[-1,-1],"022")
xrect(150,460,100,150)
xclip(150,460,100,150)
xsegs(10*x1+200*ones(x1),10*y1+200*ones(y1))
// usual rectangle boundaries clipping zone
xbasc(); plot2d([-100,500],[-100,600],[-1,-1],"022")
xclip("clipgrf")
xsegs(10*x1+200*ones(x1),10*y1+200*ones(y1));
// clipping of
xclip()
```

AUTHOR : J.Ph.C.

2.0.294 xdel ____ delete a graphics window

CALLING SEQUENCE :

```
xdel([win-num])
```

DESCRIPTION :
`xdel` deletes the graphics window `win-num` or the current graphics window if no argument is given.

AUTHOR : J.Ph.C.

2.0.295 xend ____ close a graphics session

CALLING SEQUENCE :

```
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`.

EXAMPLE :

```
driver("Pos")
xinit("foo.ps")
plot2d()
xend()
driver("X11")
```

SEE ALSO : `xbasimp` 157, `xend` 160

AUTHOR : J.Ph.C.

2.0.296 xfarc — fill a part of an ellipse

CALLING SEQUENCE :

```
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 graphics style and scale.

EXAMPLE :

```
// isoview scaling
plot2d(0,0,-1,"031"," ",[-2,-2,2,2])
xfarc(-0.5,0.5,1,1,0,90*64)
xset("pattern",2)
xfarc(0.5,0.5,1,1,0,360*64)
```

SEE ALSO : `xarc` 154, `xarcs` 154, `xfarcs` 161

AUTHOR : J.Ph.C.

2.0.297 xfarcs — fill parts of a set of ellipses

CALLING SEQUENCE :

```
xfarcs(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` 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`). `style(i)` gives the dash style used to draw ellipse number `i`.

EXAMPLE :

```
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` 154, `xfarc` 161, `xfarc` 161

AUTHOR : J.Ph.C.

2.0.298 xfpoly — fill a polygon

CALLING SEQUENCE :

```
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 pattern. If `close` is equal to 1 a point is added to the polyline `xv,yv` to define a polygon.

EXAMPLE :

```
x=sin(2*%pi*(0:5)/5);
y=cos(2*%pi*(0:5)/5);
plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("pattern",5)
xfpoly(x,y)
xset("default")
```

SEE ALSO : `xfpolys` 162, `xpoly` 168, `xpolys` 169

AUTHOR : J.Ph.C.

2.0.299 xfpolys — fill a set of polygons

CALLING SEQUENCE :

```
xfpolys(xpols,ypols,[fill])
```

PARAMETERS :

`xpols,ypols` : matrices of the same size (p,n) (points of the polygons).
`fill` : vector of size 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 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).
- 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.

EXAMPLE :

```
plot2d(0,0,[-1],"012"," ",[0,-10,210,40])
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];
// setting the current dash (or line color)
xset("dashes",5)
xfpolys(xpols,ypols,[-1,0,1,2])
xset("default")
```

SEE ALSO : `xfpoly` 162, `xpoly` 168, `xpolys` 169

AUTHOR : J.Ph.C.

2.0.300 xfrect ______ fill a rectangle

CALLING SEQUENCE :

```
xfrect(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) using the current scale and style.

EXAMPLE :

```
plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("pattern",5)
xfrect(-1,1,2,2)
xset("default")
```

SEE ALSO : `xrect` 169, `xrects` 170

AUTHOR : J.Ph.C.

2.0.301 xgetech ______ get the current graphics scale

CALLING SEQUENCE :

```
[wrect,frect,logflag]=xgetech()
```

PARAMETERS :

`wrect,frect` : real vectors.
`logflag` : string of size 2 "xy".

DESCRIPTION :
`xgetech` returns the current graphics scale. 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 and "l" stands for logscale. x stands for the x-axis and y stands for the y-axis.

SEE ALSO : `xsetech` 173

AUTHOR : J.Ph.C.

2.0.302 xget ______ get current values of the graphics context

CALLING SEQUENCE :

```
[x1]=xget(str,[flag])
xget()
```

PARAMETERS :

`str` : string.
`flag` : optional. Set to 1 gives a verbose mode.

DESCRIPTION :
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=xset("auto clear")` : Get the auto clear status (`"on"` or `"off"`).
`color=xget("background")` : Get the background color of the current graphics window.
`rect=xget("clipping")` : Get the clipping zone as a rectangle rect=[x,y,w,h] (Upper-Left point Width Height).
`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.
`font=xget("font")` : Get font=[fontid,fontsize], the current font and its current size.
`color=xget("foreground")` : Get the foreground color of the current graphics window.
`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)
`mark=xget("mark")` : Get the current mark and current mark size mark=[markid,marksize].
`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",x,y);` : Get the position of the upper left point of the graphics window pos=[x,y].

SEE ALSO : `xset` 174, `colormap` 118

AUTHOR : J.Ph.C.

2.0.303 xgetmouse ______ get the current position of the mouse

CALLING SEQUENCE :

```
rep=xgetmouse([flag])
```

PARAMETERS :

`rep` : vector of size 3, [x,y,ibutton].
`flag` : integer. If present, the click event queue is not cleared when entering xgetmouse.

DESCRIPTION :
If the mouse is located in the current graphics window, `xgetmouse` returns in `rep` the current mouse position (x,y) and the value `ibutton`.

The `ibutton` value indicates the action of the button at this point:

- `if ibutton` is -1 then no button was clicked.
- `if ibutton` is 0, 1 or 2, then the left, middle or right button was clicked. If the mouse is not located in the current graphics window, `xgetmouse` waits.

EXAMPLE :

```
xselect(); xbasc(); xsetech([0 0 1 1],[0 0 100 100])
xset("alufunction",6)
xtitle(" drawing a rectangle ")
[b,x0,y0]=xclick(); rep=[x0,y0,-1]; x=x0; y=y0;
xrect(x0,y0,x-x0,y-y0)
while rep(3)==-1 then
    rep=xgetmouse(0)
    xrect(x0,y0,x-x0,y0-y)
    x=rep(1); y=rep(2);
    xrect(x0,y0,x-x0,y0-y)
end
xset("alufunction",3)
```

SEE ALSO : `locate` 136, `xclick` 159

AUTHOR : S. Steer

2.0.304 xgrid — add a grid on a 2D plot

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
x=[0:0.1:2*%pi]';
plot2d(sin(x))
xgrid(2)
```

SEE ALSO : `xset` 174, `plot2d` 142

AUTHOR : J.Ph.C.

2.0.305 xinfo — draw an info string in the message subwindow

CALLING SEQUENCE :

```
xinfo(info)
```

PARAMETERS :

`info` : string

DESCRIPTION :
`xinfo` draws the string `info` in the message subwindow of the current graphics window.

2.0.306 xinit — initialisation of a graphics driver

CALLING SEQUENCE :

```
xinit([driver-name])
```

PARAMETERS :

`driver-name` : string.

DESCRIPTION :
Initialisation of the given driver.
For X Window `driver-name` can be a string which gives the name of a display and a new graphics window is created. If the argument is omitted the value of the environment variable DISPLAY is used if it exists or the value "unix:0.0" is used.
For the Postscript, Xfig or Gif driver, `driver-name` is a name of the file where all the graphics operations are recorded.

EXAMPLE :

```
driver("Pos")
xinit("foo.ps")
plot2d()
xend()
driver("X11")
```

SEE ALSO : `xbasimp` 157, `xend` 160

AUTHOR : J.Ph.C.

2.0.307 xlfont — load a font in the graphics context or query loaded font

CALLING SEQUENCE :

```
xlfont(font-name,font-id)
fonts=xlfont()
```

PARAMETERS :

`font-name` : string, name of the font family.
`font-id` : integer between 0 and 6.

`fonts` : a column vector of currently loaded font names.

DESCRIPTION :
Without any argument, `xlfont()` returns the list of currently loaded fonts.
With arguments, `xlfont` is used to load a new font at different sizes in the graphics context. The font must be available with size 8, 10, 12, 14, 18 and 24.
Default fonts are "Courier Roman" (0), "Symbol" (1), "Times Roman" (2), "Times Italic" (3), "Times Bold" (4) and "Times Bold Italic" (5).
font-name can be of 2 types:

- if it contains the character "%", it is supposed to be an X11 font name with %s in the size field of the name, for example `"-b&h-lucidabright-demibold-r-normal--%s-*-75-75-p-*-iso8859-1"`
- if it does not contain the character "%", it is supposed to be an alias name and the fonts aliased by `font-name08,...,font-name24` are loaded.

`font-id` : give the id `font-id` to the loaded font `font-name`.

SEE ALSO : `xset` 174

AUTHOR : J.Ph.C.

2.0.308 xload ______ load a saved graphics

CALLING SEQUENCE :

```
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`.

SEE ALSO : `xsave` 171

AUTHOR : J.Ph.C.

2.0.309 xname ______ change the name of the current graphics window

CALLING SEQUENCE :

```
xname(name)
```

PARAMETERS :

`name` : string, new name of the graphics window.

DESCRIPTION :
`xname` changes the name of the current graphics window.

AUTHOR : J.Ph.C.

2.0.310 xnumb ______ draw numbers

CALLING SEQUENCE :

```
xnumb(x,y,nums,[box])
```

PARAMETERS :

`x,y,nums` : vectors of same size.
`box` : integer value.

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.

EXAMPLE :

```
plot2d([-100,500],[-100,600],[-1,-1],"022")
x=0:100:200;
xnumb(x,500*ones(x),[10,20,35],1)
```

SEE ALSO : `xstring` 177

AUTHOR : J.Ph.C.

2.0.311 xpause ______ suspend Scilab

CALLING SEQUENCE :

```
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.

AUTHOR : J.Ph.C.

2.0.312 xpoly ______ draw a polyline or a polygon

CALLING SEQUENCE :

```
xpoly(xv,yv,dtype,[close])
```

PARAMETERS :

`xv,yv` : matrices of the same size (points of the polyline).
`dtype` : string (drawing style).
`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.

EXAMPLE :

```
x=sin(2*%pi*(0:5)/5);
y=cos(2*%pi*(0:5)/5);
plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("dashes",5)
xpoly(x,y,"lines",1)
xset("default")
```

SEE ALSO : `xfpoly` 162, `xfpolys` 162, `xpolys` 169

AUTHOR : J.Ph.C.

2.0.313 xpolys ______ draw a set of polylines or polygons

CALLING SEQUENCE :

```
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 `abs(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("pattern",3)
xpolys(rand(3,5),rand(3,5),[-1,-2,0,1,2])
xset("default")
```

SEE ALSO : `xfpoly` 162, `xfpolys` 162, `xpoly` 168

AUTHOR : J.Ph.C.

2.0.314 xrect ______ draw a rectangle

CALLING SEQUENCE :

```
xrect(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) using the current scale and style.

EXAMPLE :

```
plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("pattern",5)
xrect(-1,1,2,2)
xset("default")
```

SEE ALSO : `xfrect` 163, `xrects` 170

AUTHOR : J.Ph.C.

2.0.315 xrects ______ draw or fill a set of rectangles

CALLING SEQUENCE :

```
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): `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(n)`
if `fill(i)=0`, rectangle `i` is drawn using the current line style (or color).

EXAMPLE :

```
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` 163, `xrect` 169

AUTHOR : J.Ph.C.

2.0.316 xrpoly ______ draw a regular polygon

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
plot2d(0,0,-1,"012"," ",[0,0,10,10])
xrpoly([5,5],5,5)
```

SEE ALSO : `xrect` 169

2.0.317 xs2fig — send graphics to a file in Xfig syntax

CALLING SEQUENCE :

```
xs2fig(win_num,filen,[color])
```

PARAMETERS :

- `win_num` : integer scalar or vector .
- `filen` : string, file name.
- `color` : optional integer. 0 means black and white and 1 means color. The default value is to use a value compatible with the screen status.

DESCRIPTION :
`xs2fig` sends the recorded graphics of the window `win_num` in the file `filen` in Xfig syntax. This function works only if the selected driver is `"Rec"`.

SEE ALSO : `xbasimp` 157

2.0.318 xsave — save graphics into a file

CALLING SEQUENCE :

```
xsave(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 :
`xsave` saves the graphics contained in the graphics window `win-num` in the binary file `file-name`. The graphics are stored in a machine independent way (using the xdr library) and can be reloaded with `xload`.

SEE ALSO : `xload` 167

AUTHOR : J.Ph.C.

2.0.319 xsegs — draw unconnected segments

CALLING SEQUENCE :

```
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 dash style to use for all segments. If `style` is a negative scalar, then current dash style is used. If `style` is a vector, then `style(i)` gives the style 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)`.

EXAMPLE :

```
x=2*%pi*(0:9)/8;
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)
```

AUTHOR : J.Ph.C.

2.0.320 xselect — raise the current graphics window

CALLING SEQUENCE :

```
xselect()
```

DESCRIPTION :
`xselect` raises the current graphics window. It creates the window if none exists. If the current graphics window is iconified nothing is done.

AUTHOR : J.Ph.C.

2.0.321 xsetech set the sub window of a graphics window for plotting

CALLING SEQUENCE :

```
xsetech(wrect,[frect,logflag])
```

PARAMETERS :

`wrect` : vector of size 4, defining the sub window.
`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.

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"` in `plot2d`).

`frect=[xmin,ymin,xmax,ymax]` is used to set the graphics scale and is just like the `rect` argument of `plot2d`. Its default value is `[0,0,1,1]`.

EXAMPLE :

```
// 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])
xset("default")
```

SEE ALSO : `isoview` 135, `square` 152, `xgetech` 163

AUTHOR : J.Ph.C.

2.0.322 xset set values of the graphics context

CALLING SEQUENCE :

```
xset(choice-name,x1,x2,x3,x4,x5)
xset()
```

PARAMETERS :

`choice-name` : string
`x1,...,x5` : depending on `choice-name`
`xset("alufunction",number)` : Used to set the logical function for drawing. It works only under X11. The logical function used is set by `x1`. src is the source ie what we want to draw. dst is the destination ie what is already drawn where we want to draw.

Usual values are: 3 for copying (default), 6 for animation and 0 for clearing.

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"

`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 graphics window.
`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("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)` : Set the dash style to style i (0 for solid line). This is used only when in black and white mode: `xset("use color",0)`. Use `xset()` to see the styles.
`xset("default")` : Reset the graphics context to default values.
`xset("font",fontid,fontsize)` : Set the current font and its current size.
`xset("foreground",color)` : Set the foreground color of the current graphics window.
`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 hidden faces in `plot3d`.

`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("mark",markid,marksize)` : Set the current mark and the current mark size. Use `xset()` to see the marks.

`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")`.

`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("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.

DESCRIPTION :

`xset` is used to set values of the graphics context. When called no argument, a choice menu is created showing the current values and changes can be performed through toggle buttons.

SEE ALSO : `colormap` 118, `xget` 164, `getcolor` 129, `getsymbol` 129

AUTHOR : J.Ph.C.

2.0.323 xsetm ______ dialog to set values of the graphics context

CALLING SEQUENCE :

```
xsetm()
```

DESCRIPTION :

`xsetm()` is the same as `xset()`.

SEE ALSO : `xset` 174

AUTHOR : J.Ph.C.

2.0.324 xstringb ______ draw strings into a box

CALLING SEQUENCE :

```
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 the current graphic scale. 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.

EXAMPLE :

```
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` 153, `xstring` 177, `xstringl` 176, `xtitle` 178

AUTHOR : J.Ph.C.

2.0.325 xstringl ______ compute a box which surrounds strings

CALLING SEQUENCE :

```
rect=xstringl(x,y,str)
```

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.

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.

EXAMPLE :

```
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)
```

SEE ALSO : `titlepage` 153, `xstring` 177, `xstringl` 176, `xtitle` 178

AUTHOR : J.Ph.C.

2.0.326 xstring _______ draw strings

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
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` 153, `xnumb` 168, `xstringb` 176, `xstringl` 176, `xtitle` 178

AUTHOR : J.Ph.C.

2.0.327 xtape _______ set up the record process of graphics

CALLING SEQUENCE :

```
xtape(str,[num,rect])
```

PARAMETERS :

`str` : string, "on", "clear", "replay" or "replaysc".
`num` : integer.
`rect` : row vector of size 4.

DESCRIPTION :

`xtape` is used to set up the record process of graphics:

xtape("on") just selects the driver "Rec" which records all the graphics operations.

xtape("clear",num) clears the graphics window `num` and clears the recorded graphics associated with window `num`.

xtape("replay",num) redisplays all the recorded graphics in the window `num`.

xtape("replaysc",num,rect) replots the graphics window `num` using `rect=[xmin,ymin,xmax,ymax]` as x and y bounds.

SEE ALSO : `driver` 120, `replot` 149, `xbasc` 156, `xbasr` 157

AUTHOR : J.Ph.C.

2.0.328 xtitle _______ add titles on a graphics window

CALLING SEQUENCE :

```
xtitle(xtit,[xax,yax,encad])
```

PARAMETERS :

`xtit,xax,yax` : matrices of strings.
`encad` : integer value. If it is 1, a box is drawn around each title.

DESCRIPTION :

`xtitle` add titles on a 2D or 3D plot. `xtit` is the general title, `xax` is the title on the x-axis and `yax` is the title on the y-axis. `xtitle` must be called after a call to a high level plotting function (`plot2d`, `plot3d`, ...). If the arguments are matrices, each line of the matrices is displayed on a different line.

Enter the command `xtitle()` to see a demo.

SEE ALSO : `titlepage` 153

AUTHOR : J.Ph.C.

2.0.329 zgrid _______ zgrid plot

CALLING SEQUENCE :

```
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` 273, `freson` 275

Chapter 3

Utilities and Elementary Functions

3.0.330 abs — absolute value, magnitude

CALLING SEQUENCE :

```
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`.

EXAMPLE :

```
abs([1,%i,-1,-%i,1+%i])
```

3.0.331 acosh — hyperbolic cosine inverse

CALLING SEQUENCE :

```
[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[. It takes his values in

$$]0,+\infty[\times]-\pi,\pi] \quad \text{and} \quad [0]\times[0,\pi]$$

EXAMPLE :

```
x=[0,1,%i];
cosh(acosh(x))
```

3.0.332 acoshm — matrix hyperbolic inverse cosine

CALLING SEQUENCE :

```
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-ey`
For non symmetric matrices result may be inaccurate.

EXAMPLE :

```
A=[1,2;3,4];
coshm(acoshm(A))
A(1,1)=A(1,1)+%i;
coshm(acoshm(A))
```

SEE ALSO : `acosh` 180, `logm` 215, `sqrtm` 234

3.0.333 acos ______ element wise cosine inverse

CALLING SEQUENCE :

```
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].
`acos` takes values in :

$$]0,\pi[\times]-\infty+\infty[$$

$$[0]\times[0,+\infty] \quad \text{and} \quad [\pi]\times]-\infty,0] \quad \text{(real x imag)}$$

EXAMPLE :

```
x=[1,%i,-1,-%i]
cos(acos(x))
```

3.0.334 acosm ______ matrix wise cosine inverse

CALLING SEQUENCE :

```
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))`

EXAMPLE :

```
A=[1,2;3,4];
cosm(acosm(A))
```

SEE ALSO : `acos` 181, `sqrtm` 234, `logm` 215

3.0.335 addf ______ symbolic addition

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
addf('0','1')
addf('1','a')
addf('1','2')
'a'+'b'
```

SEE ALSO : `mulf` 219, `subf` 239, `ldivf` 212, `rdivf` 222, `eval` 203, `evstr` 21

3.0.336 addmenu ______ interactive button or menu definition

CALLING SEQUENCE :

```
addmenu(button [,submenus] [,action])
addmenu(gwin,button [,submenus] [,action])
```

PARAMETERS :

`button` : a character string. The button name. On Windows operating systems (not X_window), an & can be placed before the character in the name to be used for keyboard shortcut; this character will be underlined on the GUI.
`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
`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` 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 `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`.

Actions associated with the kth sub_menu must be defined by scilab instructions stored in the kth element of the character string variable which name is

`button` for a main window command
`button_gwin` for a graphic window command

EXAMPLE :

```
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'')'
```

SEE ALSO : setmenu 225, unsetmenu 248, delmenu 199

3.0.337 adj2sp ______ converts adjacency form into sparse matrix.

CALLING SEQUENCE :
A = adj2sp(xadj,adjncy,anz) A = adj2sp(xadj,adjncy,anz,mn)

PARAMETERS :

```
.TP 7
xadj
:  integer vector of length (n+1).
.TP 7
adjncy
:  integer vector of length nz containing the row indices
   for the corresponding elements in anz
.TP 7
anz
:  column vector of length nz, containing the non-zero
   elements of A
.TP 7
mn
: row vector with 2 entries, \fVmn=size(A)\fR (optional).
.TP 7
A
:  real or complex sparse matrix (nz non-zero entries)
```

DESCRIPTION :

```
\fVsp2adj\fR converts an adjacency form representation of a matrix
into its standard Scilab representation (utility fonction).
\fVxadj, adjncy, anz\fR = adjacency representation of \fVA\fR i.e:
.LP
\fVxadj(j+1)-xadj(j)\fR = number of non zero entries in row j.
\fVadjncy\fR = column index of the non zeros entries
in row 1, row 2,..., row n.
\fVanz\fR = values of non zero entries in row 1, row 2,..., row n.
\fVxadj\fR is a (column) vector of size n+1 and
\fVadjncy\fR is an integer (column) vector of size \fVnz=nnz(A)\fR.
\fVanz\fR is a real vector of size \fVnz=nnz(A)\fR.
```

EXAMPLE :

```
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 229, spcompack 230

3.0.338 amell ______ Jacobi's am function

CALLING SEQUENCE :

```
[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 198, %sn 394, %asn 362

3.0.339 asinh ______ hyperbolic sine inverse

CALLING SEQUENCE :

```
[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[It takes his values in sets

$$]-\infty+\infty[\times]-\pi/2,\pi/2[$$

$$[-\infty,0[\times[-\pi/2] \quad \text{and} \quad [0,\infty]\times[\pi/2] \quad \text{(real x imag)}$$

EXAMPLE :

```
A=[1,2;2,3]
sinh(asinh(A))
```

3.0.340 asinhm ______ matrix hyperbolic inverse sine

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
A=[1,2;2,3]
sinhm(asinhm(A))
```

SEE ALSO : asinh 184, logm 215, sqrtm 234

3.0.341 asin ____ sine inverse

CALLING SEQUENCE :

```
[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]. It takes his values in sets

$$]-\pi/2,\pi/2[\times]-\infty+\infty[$$

$$[-\pi/2]\times[0,+\infty] \quad \text{and} \quad [\pi/2]\times]-\infty,0] \quad \text{(real x imag)}$$

EXAMPLE :

```
A=[1,2;3,4]
sin(asin(A))
```

SEE ALSO : `sin` 226, `sinm` 227, `asinm` 185

3.0.342 asinm ____ matrix wise sine inverse

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
A=[1,2;3,4]
sinm(asinm(A))
asinm(A)+%i*logm(%i*A+sqrtm(eye()-A*A))
```

SEE ALSO : `asin` 185, `sinm` 227

3.0.343 atanh ____ hyperbolic tangent inverse

CALLING SEQUENCE :

```
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[
This function takes values in

$$]-\infty+\infty[\times]-\pi/2,\pi/2[$$

$$[-\infty,0[\times[-\pi/2] \quad \text{and} \quad]0,\infty[\times[\pi/2] \quad \text{(real x imag)}$$

EXAMPLE :

```
x=[0,%i,-%i]
tanh(atanh(x))
```

3.0.344 atanhm ____ matrix hyperbolic tangent inverse

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
A=[1,2;3,4];
tanhm(atanhm(A))
```

SEE ALSO : `atanh` 186, `tanhm` 242

3.0.345 atan ____ tangent inverse

CALLING SEQUENCE :

```
[t]=atan(x)
```

PARAMETERS :

`x` : real or complex vector

`t` : real or complex vector

DESCRIPTION :

The components of vector `t` are the arctangent of the corresponding entries of vector `x`.
`atan(x,y)` is the same as `atan(x/y)` but `y` is allowed to be zero.

EXAMPLE :

```
x=[1,%i,-1,%i]
phasex=atan(imag(x),real(x))
```

SEE ALSO : `tan` 243, `ieee` 44

3.0.346 atanm _ square matrix tangent inverse

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
tanm(atanm([1,2;3,4]))
```

SEE ALSO : `atan` 186

3.0.347 besseli _ Modified I sub ALPHA Bessel functions of the first kind.

CALLING SEQUENCE :

```
y = besseli(alpha,x)
y = besseli(alpha,x,ice)
```

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 :

`besseli(alpha,x)` computes I sub ALPHA modified Bessel functions of the first kind, for real, non-negative order `alpha` and 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 `besseli(alpha(j),x(i))`.
If `ice is equal to 2 exponentialy scaled Bessel functions is computed`

EXAMPLE :

```
besseli(0.5:3,1:4)
besseli(0.5:3,1:4,2)
```

SEE ALSO : `besselj 188, besselk 188`

3.0.348 besselj _ Modified J sub ALPHA Bessel functions of the first kind.

CALLING SEQUENCE :

```
y = besselj(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 :

`besselj(alpha,x)` computes J sub ALPHA modified Bessel functions of the first kind, for real, non-negative order `alpha` and 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 `besselj(alpha(j),x(i))`.

EXAMPLE :

```
besselj(0.5:3,1:4)
```

SEE ALSO : `besseli` 187, `besselk` 188

3.0.349 besselk _ Modified K sub ALPHA Bessel functions of the second kind.

CALLING SEQUENCE :

```
y = besselk(alpha,x)
y = besselk(alpha,x,ice)
```

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 :

`besselk(alpha,x)` computes K sub ALPHA modified Bessel functions of the second kind, for real, non-negative order `alpha` and 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 `besselk(alpha(j),x(i))`.
If `ice is equal to 2 exponentialy scaled Bessel functions is computed`

EXAMPLE :

```
besselk(0.5:3,1:4)
besselk(0.5:3,1:4,2)
```

SEE ALSO : `besselj 188, besseli 187, bessely 189`

3.0.350 bessely _____ Modified Y sub ALPHA Bessel functions of the second kind.

CALLING SEQUENCE :

```
y = bessely(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)
```

DESCRIPTION :

`bessely(alpha,x)` computes K sub ALPHA modified Bessel functions of the second kind, for real, non-negative order `alpha` and 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 `bessely(alpha(j),x(i))`.

EXAMPLE :

```
bessely(0.5:3,1:4)
```

SEE ALSO : besselj 188, besseli 187, besselk 188

3.0.351 bloc2exp _____ block-diagram to symbolic expression

CALLING SEQUENCE :

```
[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_portr`
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())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 191

AUTHOR : S. S., F. D. (INRIA)

3.0.352 bloc2ss ________ block-diagram to state-space conversion

CALLING SEQUENCE :

```
[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_port
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 189

AUTHOR : S. S., F. D. (INRIA)

3.0.353 calerf ________ computes error functions.

CALLING SEQUENCE :
y = calerf(x,flag)
PARAMETERS :

`x` : real vector
`flag` : integer indicator
`y` : real vector (of same size)

DESCRIPTION :
`calerf(x,0)` computes the error function:

$$y = 2/\sqrt{\pi}\int_0^x exp(-t^2)dt$$

`calerf(x,1)` computes the complementary error function:

$$y = 2/\sqrt{\pi}\int_x^\infty exp(-t^2)dt$$

$$y = 1 - erf(x)$$

`calerf(x,2)` computes the scaled complementary error function:

$$y = exp(x^2)erfc(x)\ \frac{1}{xsqrt\pi}\text{for large x}$$

EXAMPLE :

```
deff('y=f(t)','y=exp(-t^2)');
calerf(1,0)
2/sqrt(%pi)*intg(0,1,f)
```

SEE ALSO : `erf` 202, `erfc` 202, `calerf` 192

3.0.354 c_link ______ check dynamic link

CALLING SEQUENCE :

```
c_link('routine-name')
[test,ilib]=c_link('routine-name')
test=c_link('routine-name',num)
```

DESCRIPTION :
`c_links` 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).
EXAMPLE :

```
if c_link('foo') then link('foo.o','foo');end
// to unlink all the shared libarries which contain foo
a=%t; while a ;[a,b]=c_link('foo'); ulink(b);end
```

SEE ALSO : `link` 54, `fort` 31

3.0.355 cmb_lin ______ symbolic linear combination

CALLING SEQUENCE :

```
[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` 219, `addf` 181

3.0.356 conj ______ conjugate

CALLING SEQUENCE :

```
[y]=conj(x)
```

PARAMETERS :

`x,y` : real or complex matrix.

DESCRIPTION :
`conj(x)` is the complex conjugate of `x`.
EXAMPLE :

```
x=[1+%i,-%i;%i,2*%i];
conj(x)
x'-conj(x)  //x'  is conjugate transpose
```

3.0.357 convstr ______ case conversion

CALLING SEQUENCE :

```
[y]=convstr(str-matrix, ["flag"])
```

PARAMETERS :

`str-matrix, y` : matrices of strings
`"flag"` : string (`"u"` for upper or `"l"` for lower (default value))

DESCRIPTION :
converts the matrix of strings `str-matrix` into lower case (for `"l"` ;default value) or upper case (for `"u"`).
EXAMPLE :

```
A=['this','is';'my','matrix'];
convstr(A,'u')
```

3.0.358 cosh ______ hyperbolic cosine

CALLING SEQUENCE :

```
[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`.
EXAMPLE :

```
x=[0,1,%i]
acosh(cosh(x))
```

SEE ALSO : `cos` 195, `acosh` 180

3.0.359 coshm — matrix hyperbolic cosine

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
A=[1,2;2,4]
acoshm(coshm(A))
```

SEE ALSO : `cosh` 194, `expm` 433

3.0.360 cos — cosine function

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
x=[0,1,%i]
acos(cos(x))
```

SEE ALSO : `cosm` 195

3.0.361 cosm — matrix cosine function

CALLING SEQUENCE :

```
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))`.

EXAMPLE :

```
A=[1,2;3,4]
cosm(A)-0.5*(expm(%i*A)+expm(-%i*A))
```

SEE ALSO : `cos` 195, `expm` 433

3.0.362 cotg — cotangent

CALLING SEQUENCE :

```
[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)`

EXAMPLE :

```
x=[1,%i];
cotg(x)-cos(x)./sin(x)
```

SEE ALSO : `tan` 243

3.0.363 coth — hyperbolic cotangent

CALLING SEQUENCE :

```
[t]=coth(x)
```

DESCRIPTION :
the elements of vector `t` are the hyperbolic cotangent of elements of the vector `x`.

EXAMPLE :

```
x=[1,2*%i]
t=exp(x);
(t-ones(x)./t).\(t+ones(x)./t)
coth(x)
```

SEE ALSO : `cotg` 196

3.0.364 cothm — matrix hyperbolic cotangent

CALLING SEQUENCE :

```
[t]=cothm(x)
```

DESCRIPTION :
`cothm(x)` is the matrix hyperbolic cotangent of the square matrix x.

EXAMPLE :

```
A=[1,2;3,4];
cothm(A)
```

SEE ALSO : `coth` 196

3.0.365 cumprod _______________ cumulative product

CALLING SEQUENCE :

```
y=cumprod(x)
y=cumprod(x,'r') or y=cumprod(x,1)
y=cumprod(x,'c') or y=cumprod(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=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))`.

EXAMPLE :

```
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 : `cumprod` 197, `sum` 239

3.0.366 cumsum _______________ cumulative sum

CALLING SEQUENCE :

```
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))`

EXAMPLE :


```
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` 197, `sum` 239

3.0.367 debug _______________ debugging level

CALLING SEQUENCE :

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

PARAMETERS :

`level-int` : integer (0 to 4)

DESCRIPTION :
For the values 0,1,2,3,4 of `level-int` , `debug` defines various levels of debugging. (For Scilab experts only).

3.0.368 dec2hex _______________ hexadecimal representation of integers

CALLING SEQUENCE :

```
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

EXAMPLE :

```
dec2hex([2748 10;11 3])
```

3.0.369 delip _______________ elliptic integral

CALLING SEQUENCE :

```
[r]=delip(x,ck)
```

PARAMETERS :

`x` : real number (or real vector)

`ck` : real number between -1 and 1
`r` : real or complex number (or vector)

DESCRIPTION :
returns the value of the elliptic integral with parameter `ck` :

$$r = \int_0^x \frac{1}{\sqrt{(1-t^2)(1-ck^2t^2)}}dx$$

`x` real and positive. When called with `x` a real vector `r` is evaluated for each entry of `x`.
EXAMPLE :

```
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` 184, `%asn` 362, `%sn` 394

3.0.370 delmenu ______ interactive button or menu deletion

CALLING SEQUENCE :

```
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.
EXAMPLE :

```
addmenu('foo')
delmenu('foo')
```

SEE ALSO : `setmenu` 225, `unsetmenu` 248, `addmenu` 182

3.0.371 demos ______ guide for scilab demos

CALLING SEQUENCE :

```
demos()
```

DESCRIPTION :
`demos()` is an interactive guide to execute various scilab demonstrations The source code of each demo is in the directory SCIDIR/demos/...

3.0.372 diag ______ diagonal including or extracting

CALLING SEQUENCE :

```
[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,...).
EXAMPLE :

```
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` 240, `sparse` 229

3.0.373 dlgamma ______ derivative of gammaln function.

CALLING SEQUENCE :

```
 y = dlgamma(x)
```

PARAMETERS :

`x` : real vector
`y` : real vector with same size.

DESCRIPTION :
`dlgamma(x)` evaluates the derivative of gammaln function at all the elements of `x`. `x` must be real.
EXAMPLE :

```
dlgamma(0.5)
```

SEE ALSO : `gamma` 204, `gammaln` 204

3.0.374 edit _____ function editing

CALLING SEQUENCE :

```
newname=edit(functionname [, editor])
```

PARAMETERS :

`functionname` : character string
`editor` : character string

DESCRIPTION :
If functionname is the name of a defined scilab function `edit(functionname ,[editor])` try to open the associated file `functionname.sci`. If this file can't be modified `edit` first create a copy of this file in the TMPDIR directory.

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

When leaving the editor the modified or defined function is loaded into Scilab under the name `newname`.

The editor character string can be used to specify your favourite text editor.

Default editor is Emacs. This function should be customized according to your needs.

EXAMPLE :

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

SEE ALSO : `manedit` 216

3.0.375 emptystr _____ zero length string

CALLING SEQUENCE :

```
s=emptystr()
s=emptystr(a)
s=emptystr(m,n)
```

PARAMETERS :

`a` : any type of matrix
`s` : character string matrix
`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

EXAMPLE :

```
x=emptystr();for k=1:10, x=x+','+string(k);end
```

SEE ALSO : `part` 79, `length` 52, `string` 100

3.0.376 erfc _____ The complementary error function.

CALING SEQUENCE :

```
y = erfc(x)
```

PARAMETERS :

`x` : real vector
`y` : real vector (of same size)

DESCRIPTION :
`erfc` computes the complementary error function:

$$y = 2/\sqrt{(\pi)} \int_x^{\infty} exp(-t^2)dt$$

$$y = 1 - erf(x)$$

EXAMPLE :

```
erf([0.5,0.2])+erfc([0.5,0.2])
```

SEE ALSO : `erf` 202, `erfcx` 202, `calerf` 192

3.0.377 erfcx _____ scaled complementary error function.

CALING SEQUENCE :

```
y = erfcx(x)
```

PARAMETERS :

`x` : real vector
`y` : real vector (of same size)

DESCRIPTION :
`erfcx` computes the scaled complementary error function:

$$y = exp(x^2)erfc(x)\ (1/\sqrt{pi})1/x\text{for large}x$$

SEE ALSO : `erf` 202, `erfc` 202, `calerf` 192

3.0.378 erf _____ The error function.

CALLING SEQUENCE :

```
y = erf(x)
```

PARAMETERS :

`x` : real vector
`y` : real vector (of same size)

DESCRIPTION :
`erf` computes the error function:

$$y = 2/\sqrt(\pi)\int_0^x exp(-t^2)dt$$

EXAMPLE :

```
deff('y=f(t)','y=exp(-t^2)');
erf(0.5)-2/sqrt(%pi)*intg(0,0.5,f)
```

SEE ALSO : `erfc` 202, `erfcx` 202, `calerf` 192

3.0.379 eval ______ evaluation of a matrix of strings

CALLING SEQUENCE :

```
[H]= eval(Z)
```

DESCRIPTION :
returns the evaluation of the matrix of character strings `Z`.

EXAMPLE :

```
a=1; b=2; Z=['a','sin(b)'] ; eval(Z)  //returns the matrix [1,0.909];
```

SEE ALSO : `evstr` 21, `execstr` 23

3.0.380 full ______ sparse to full matrix conversion

CALING SEQUENCE :

```
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`).

EXAMPLE :

```
sp=sparse([1,2;5,4;3,1],[1,2,3]);
A=full(sp)
```

SEE ALSO : `sparse` 229, `sprand` 233, `speye` 231

3.0.381 gammaln ______ The logarithm of gamma function.

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
gammaln(0.5)
```

SEE ALSO : `gamma` 204, `dlgamma` 200

3.0.382 gamma ______ The gamma function.

CALLING SEQUENCE :

```
y = gamma(x)
```

PARAMETERS :

`x` : real vector
`y` : real vector with same size.

DESCRIPTION :
`gamma(x)` evaluates the gamma function at all the elements of `x`. `x` must be real.

$$y = \int_0^\infty t^(x-1)exp(-t)dt$$

`gamma(n+1) = n!`

EXAMPLE :

```
gamma(0.5)
gamma(6)-prod(1:5)
```

SEE ALSO : `gammaln` 204, `dlgamma` 200

3.0.383 getvalue ______ xwindow dialog for data acquisition

CALLING SEQUENCE :

```
[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(typ1,dim1,..,typn,dimn)`
`typi` : 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
`dimi` : 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 as 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.

EXAMPLE :

```
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` 250, `x_matrix` 250, `x_dialog` 249

AUTHOR : S. Steer

3.0.384 G_make ______ call make or nmake

CALLING SEQUENCE :

```
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 (i.e unix or windows/gcwin32) `G_make` calls the `make` utility for building target `files` and returns the value of `files` in the variable `Rfiles`. On windows platforms, (i.e when Scilab was compiled with Microsoft VisualC++). `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. and such examples can be found in the directory `SCIDIR/examples/addinter-examples`

```
files=G_make([TMPDIR+'/ex1cI.o',TMPDIR+'/ex1c.o'],'ex1c.dll');// compilation
addinter(files,'foobar','foubare'); // link
```

SEE ALSO : `addinter` 6

3.0.385 gsort ______ decreasing order sorting

CALLING SEQUENCE :

```
[s, [k]]=gsort(v )
[s, [k]]=gsort(v,flag1)
[s, [k]]=gsort(v,flag1,flag2)
```

PARAMETERS :

`v,s` : real vector/matrix; character string vector/matrix
`flag1` : a string `'r'`, `'c'`,`'g'`,`'lr'` and `'lc'`.
`flag2` : a string `'i'` for increasing and `'d'` for decreasing order. k : vector or matrix of integers

DESCRIPTION :
`gsort` is similar to `sort` with additional properties. The third argument can be used to chose between increasing or decreasing order. The second argument can be used for lexical orders.
`[s,k]=gsort(a,'g')` and `[s,k]=gsort(a,'g','d')` are the same as `[s,k]=gsort(a)`. They perform a sort of the entries of matrix `a`, `a` being seen as the stacked vector `a(:)` (columnwise).
`[s,k]=gsort(a,'g','i')` performs the same operation but in increasing order.
`[s,k]=gsort(a,'lr')` sort the rows of the matrix `int(a)` (if `a` is a real matrix) or `a` (if `a` is a character string matrix) in lexical decreasing order. `s` is obtained by a permutation of the rows of matrix `int(a)` (or `a`) given by the column vector `k`) in such a way that the rows of `s` verify `s(i,:) > s(j,:)` if i<j. `[s,k]=gsort(a,'lr','i')` performs the same operation for increasing lexical order
`[s,k]=gsort(a,'lc')` sort the columns of the matrix `int(a)` (if `a` is a real matrix) or `a` (if `a` is a character string matrix) in lexical decreasing order. `s` is obtained by a permutation of the columns of matrix `int(a)` (or `a`) given by the row vector `k`) in such a way that the columns of `s` verify `s(:,i) > s(:,j)` if i<j. `[s,k]=gsort(a,'lc','i')` performs the same operation for increasing lexical order

EXAMPLE :

```
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')
```

SEE ALSO : `find` 29

3.0.386 halt ______ stop execution

CALLING SEQUENCE :

```
halt()
```

DESCRIPTION :
stops execution until something is entered in the keyboard.

SEE ALSO : `pause` 79, `return` 90, `exec` 22

3.0.387 havewindow ______ return scilab window mode

CALLING SEQUENCE :

```
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".

3.0.388 hex2dec __ converts hexadecimal representation of integers to numbers

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
hex2dec(['ABC','0','A'])
```

3.0.389 input ______ prompt for user input

CALLING SEQUENCE :

```
[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`.
Invoked with two arguments, the output is a character string which is the expression entered at keyboard.

EXAMPLE :

```
//x=input("How many iterations?")
//x=input("What is your name?","string")
```

SEE ALSO : `file` 28, `read` 88, `write` 111, `evstr` 21, `x_dialog` 249, `x_mdialog` 250

3.0.390 integrate ______ integration by quadrature

CALLING SEQUENCE :

```
[x]=integrate(expr,v,x0,x1 [,ea [,er]])
```

PARAMETERS :

`expr` : external Scilab
`v` : string (integration variable)
`x0,x1` : real numbers (bounds of integration)
`ea,er` : real numbers (absolute error bound) Default value: 0
`er` : real number, (relative error bound) Default value: 1.d-8

DESCRIPTION :
computes :

$$x = \int_{x_0}^{x_1} f(v)dv$$

The evaluation hopefully satisfies following claim for accuracy: `abs(I-x)<= max(ea,er*abs(I))` where I stands for the exact value of the integral.

EXAMPLE :

```
integrate('sin(x)','x',0,%pi)
integrate(['if x==0 then 1,';
           'else sin(x)/x,end'],'x',0,%pi)
```

SEE ALSO : `intg` 342

3.0.391 interpln ______ linear interpolation

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
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` 232, `interp` 209, `smooth` 227

3.0.392 interp ______ interpolation

CALLING SEQUENCE :

```
[f0 [,f1 [,f2 [,f3]]]]=interp(xd,x,f,d)
```

PARAMETERS :

`xd` : real vector
`x,f,d` : real vectors from spline
`fi` : vectors (derivatives)

DESCRIPTION :
given three vectors `(x,f,d)` defining a spline function (see `splin`) with `fi=S(xi), di = S'(xi)` this function evaluates `S` (resp. `S', S'', S'''`) at `xd(i)`.

`x` : vector of `xi (x(1) < x(2) < ...)`
`f` : vector of `S(xi)`
`d` : vector of `S'(xi)`
`f0` : vector `[S(xd(1),S(xd(2)),S(xd(3)),...]`
`f(1 2 3)` : vector of first, second, third derivative of `S` at `xd=[xd(1),xd(2),...]` i.e.

```
f1 = [S'(xd(1)),S'(xd(2)),...]
f2 = [S''(xd(1)),S''(xd(2)),...]
```

SEE ALSO : `splin` 232, `smooth` 227, `interpln` 208

3.0.393 intersect returns the vector of common values of two vectors

CALLING SEQUENCE :

```
[v, [ka,kb]]=intersect(a,b)
```

PARAMETERS :

`a` : vector of real numbers or strings
`b` : vector of real numbers or strings
`v` : row vector of real 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)`.

EXAMPLE :

```
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))
```

SEE ALSO : `unique` 247, `sort` 228, `union` 247

3.0.394 intsplin ______ integration of experimental data by spline interpolation

CALLING SEQUENCE :

```
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 :

$$v = \int_{x_0}^{x_1} f(x)dx$$

Where f is a function described by a set of experimental value:

$$s(i) = f(x(i))$$

and

$$x_0 = x(1), x_1 = x(n)$$

Between mesh points function is interpolated using spline's.

EXAMPLE :

```
t=0:0.1:%pi
intsplin(t,sin(t))
```

SEE ALSO : `intg` 342, `integrate` 208, `inttrap` 210, `splin` 232

3.0.395 inttrap ______ integration of experimental data by trapezoidal interpolation

CALLING SEQUENCE :

```
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 :

$$v = \int_{x_0}^{x_1} f(x)dx$$

Where f is a function described by a set of experimental value:

$$s(i) = f(x(i))$$

and

$$x_0 = x(1), x_1 = x(n)$$

Between mesh points function is interpolated linearly.

EXAMPLE :

```
t=0:0.1:%pi
inttrap(t,sin(t))
```

SEE ALSO : `intg` 342, `integrate` 208, `intsplin` 210, `splin` 232

3.0.396 isdef ______ check variable existence

CALLING SEQUENCE :

```
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 `'var-name'` exists and `%F` otherwise.
`isdef(name,'local')` returns `%T` if the variable `'var-name'` exists in the local environment of the current function and `%F` otherwise.

EXAMPLE :

```
A=1;
isdef('A')
clear A
isdef('A')
```

SEE ALSO : `exists` 23, `whereis` 108, `type` 103, `typeof` 246, `clear` 13

3.0.397 isinf ______ check for infinite entries

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
isinf([1 0.01 -%inf %inf])
```

SEE ALSO : `isnan` 211

3.0.398 isnan ______ check for "Not a Number" entries

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
isnan([1 0.01 -%nan %inf-%inf])
```

SEE ALSO : `isinf` 211

3.0.399 isreal ______ check if a variable as real or complex entries

CALLING SEQUENCE :

```
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 stores complex numbers.
`isreal(x,eps)` returns true if x is stored as a real variable or if maximum absolute value of imaginary floating points if less or equal than eps.

EXAMPLE :

```
isreal([1 2])
isreal(1+0*%i)
isreal(1+0*%i,0)
isreal(1+%s)
isreal(sprand(3,3,0.1))
```

3.0.400 kron ______ Kronecker product (.*.)

CALLING SEQUENCE :

```
kron(x,y)
x.*.y
```

DESCRIPTION :
Kronecker tensor product of two matrices `x` and `y`. Same as `x.*.y` . `x` and `y` can be sparse matrices.

EXAMPLE :

```
A=[1,2;3,4];
kron(A,A)
A.*.A
sparse(A).*.sparse(A)
A(1,1)=%i;
kron(A,A)
```

3.0.401 ldivf ______ left symbolic division

CALLING SEQUENCE :

```
ldivf('d','c')
```

DESCRIPTION :
returns the string `'c\d'` Trivial simplifications such as `'1\c' = 'c'` are performed.

EXAMPLE :

```
ldivf('1','1')
ldivf('a','0')
ldivf('a','x')
ldivf('2','4')
```

SEE ALSO : rdivf 222, addf 181, mulf 219, evstr 21

3.0.402 lex_sort _______ lexicographic matrix rows sorting

CALLING SEQUENCE :

```
[N, [k]]=lex_sort(M [,sel] [,'unique'])
```

PARAMETERS :

M : real matrix
N : real matrix
k : column vector of integers

DESCRIPTION :
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.
EXAMPLE :

```
M=round(2*rand(20,3));

lex_sort(M)
lex_sort(M,'unique')
[N,k]=lex_sort(M,[1 3],'unique')
```

SEE ALSO : sort 228

3.0.403 linspace _______ linearly spaced vector

CALLING SEQUENCE :

```
[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.
EXAMPLE :

```
linspace(1,2,10)
```

SEE ALSO : logspace 215

3.0.404 log10 _______ logarithm

CALLING SEQUENCE :

```
y=log10(x)
```

PARAMETERS :

x : vector or matrix

DESCRIPTION :
decimal logarithm.If x is a vector log10(x)=[log10(x1),...,log10(xn)].
EXAMPLE :

```
10.^log10([1,%i,-1,-%i])
```

SEE ALSO : log 214, hat 41, ieee 44

3.0.405 log _______ natural logarithm

CALLING SEQUENCE :

```
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.
EXAMPLE :

```
exp(log([1,%i,-1,-%i]))
```

SEE ALSO : exp 432, logm 215, ieee 44

3.0.406 logm — square matrix logarithm

CALLING SEQUENCE :

```
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`

EXAMPLE :

```
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` 433, `log` 214

3.0.407 logspace — logarithmically spaced vector

CALLING SEQUENCE :

```
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`.

EXAMPLE :

```
logspace(1,2,10)
```

SEE ALSO : `linspace` 213

3.0.408 macr2lst — function to list conversion

CALLING SEQUENCE :

```
[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. For use with `mac2for`.

SEE ALSO : `macrovar` 216

3.0.409 macrovar — variables of function

CALLING SEQUENCE :

```
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)`)

EXAMPLE :

```
deff('y=f(x1,x2)','loc=1;y=a*x1+x2-loc')
vars=macrovar(f)
```

SEE ALSO : `string` 100, `macr2lst` 215

3.0.410 manedit — editing a manual item

CALLING SEQUENCE :

```
manedit(manitem ,[editor])
```

PARAMETERS :

`manitem` : character string (usually, name of a function)
`editor` : character string

DESCRIPTION :

`edit(manitem ,[editor])` opens the file `manitem` in the editor given by `editor`.
Default editor is Emacs. This function should be customized according to your needs.

EXAMPLE :

```
//manedit('lqg')
```

SEE ALSO : `whereis` 108, `edit` 201

3.0.411 mean ____ mean (row mean, column mean) of vector/matrix entries

CALLING SEQUENCE :

```
y=mean(x)
y=mean(x,'r')
y=mean(x,'c')
```

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,2)`) is the rowwise mean. It returns in each entry of the column vector `y` the mean of each row of `x`.
`y=mean(x,'c')` (or, equivalently, `y=mean(x,1)`) is the columnwise mean. It returns in each entry of the row vector `y` the mean of each column of `x`.

EXAMPLE :

```
A=[1,2,10;7,7.1,7.01];
mean(A)
mean(A,'r')
mean(A,'c')
```

SEE ALSO : `sum` 239, `median` 217, `st_deviation` 236

3.0.412 median ________ median (row median, column median) of vector/matrix entries

CALLING SEQUENCE :

```
y=median(x)
y=median(x,'r')
y=median(x,'c')
```

PARAMETERS :

`x` : real vector or matrix
`y` : scalar or vector

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,2)`) is the rowwise median. It returns in each entry of the column vector `y` the median of each row of `x`.
`y=median(x,'c')` (or, equivalently, `y=median(x,1)`) is the columnwise median. It returns in each entry of the row vector `y` the median of each column of `x`.

EXAMPLE :

```
A=[1,2,10;7,7.1,7.01];
median(A)
median(A,'r')
median(A,'c')
```

SEE ALSO : `sum` 239, `mean` 217

3.0.413 modulo ____________ arithmetic remainder modulo m

CALLING SEQUENCE :

```
i=modulo(n,m)
```

PARAMETERS :
n,m: integers

DESCRIPTION :
computes `i= n (modulo m)` i.e. remainder of `n` divided by `m` (`n` and `m` integers).
i = n - m .* int (n ./ m)

EXAMPLE :

```
n=[1,2,10,15];m=[2,2,3,5];
modulo(n,m)
```

3.0.414 mps2linpro ____ convert lp problem given in MPS format to linpro format

CALLING SEQUENCE :

```
lp = mps2linpro(mps)
[p,C,b,ci,cs,mi] = mps2linpro(mps)
```

PARAMETERS :

`mps` : either a character string, path of the MPS file, or an mps data structure returned by `readmps`
`lp` : a linpro data tlist with following fields:
`p` : real (column) vector (dimension `n`)
`C` : real matrix (dimension `(mi + md) x n`) (If no constraints are given, you can set `C = []`)
`b` : RHS vector (dimension `1 x (mi + 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) = -(% eps)^(-1)`.
`cs` : (column) vector of upper-bounds. (Same remarks as above).
`mi` : number of equality constraints (i.e. `C(1:mi,:)*x = b(1:mi)`)

DESCRIPTION :
mps2linpro forms Linear programing data compatible with linpro out of MPS data format.

SEE ALSO : `linpro` 345, `readmps` 222

3.0.415 mtlb_sparse ________________ convert sparse matrix

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
X=sparse(rand(2,2)); Y=mtlb_sparse(X);
Y, full(Y), [ij,v,mn]=spget(Y)
```

SEE ALSO : `full` 203, `spget` 232

3.0.416 mulf ______ symbolic multiplication

CALLING SEQUENCE :

```
mulf('d','c')
```

DESCRIPTION :
returns the string `'c*d'` Trivial simplifications such as `'1*c' = 'c'` are performed.

EXAMPLE :

```
mulf('1','a')
mulf('0','a')
'a'+'b'   //Caution...
```

SEE ALSO : `rdivf` 222, `addf` 181, `subf` 239

3.0.417 nnz ______ number of non zero entries in a matrix

CALLING SEQUENCE :

```
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

EXAMPLE :

```
sp=sparse([1,2;4,5;3,10],[1,2,3]);
nnz(sp)
a=[1 0 0 0 2];
nnz(a)
```

SEE ALSO : `spget` 232

3.0.418 norm ______ matrix norms

CALLING SEQUENCE :

```
[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)))`.

EXAMPLE :

```
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` 318, `dhnorm` 311, `h2norm` 316, `abs` 180

3.0.419 pen2ea ______ pencil to E,A conversion

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
E=[1,0];A=[1,2];s=poly(0,'s');
[E,A]=pen2ea(s*E-A)
```

3.0.420 pertrans ______ pertranspose

CALLING SEQUENCE :

```
[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).

EXAMPLE :

```
A=[1,2;3,4]
pertrans(A)
```

3.0.421 prod ______ product

CALLING SEQUENCE :

```
y=prod(x)
y=prod(x,'r') or y=prod(x,1)
y=prod(x,'c') or y=prod(x,2)
```

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(i)=prod(x(i,:))`.
`y=prod(x,'c')` (or, equivalently, `y=prod(x,2)`) computes the columns elementwise product of `x`. `y` is the column vector: `y(i)=prod(x(:,i))`.
`prod` is not implemented for sparse matrices.

EXAMPLE :

```
A=[1,2;0,100];
prod(A)
prod(A,'c')
prod(A,'r')
```

SEE ALSO : `sum` 239, `cumprod` 197

3.0.422 rdivf ______ right symbolic division

CALLING SEQUENCE :

```
["r"]=ldivf("d","c")
```

PARAMETERS :

`"d","c","r"` : strings

DESCRIPTION :
returns the string `"c/d"` Trivial simplifications such as `"c/1" = "c"` are performed.

EXAMPLE :

```
ldivf('c','d')
ldivf('1','2')
ldivf('a','0')
```

SEE ALSO : `ldivf` 212

3.0.423 readc_ ______ read a character string

CALLING SEQUENCE :

```
[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` 88

3.0.424 readmps ______ reads a file in MPS format

CALLING SEQUENCE :

```
mps= readmps (file-name,bounds [,maxsizes]);
```

PARAMETERS :

`file-name` : character string, path of the MPS file
`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.

EXAMPLE :

```
//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'];
u=file('open','/tmp/test.mps','unknown')
write(u,txt,'(a)');file('close',u)
//Read the MPS file
P=readmps('/tmp/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 218

3.0.425 sci2exp ____ converts variable to expression

CALLING SEQUENCE :

```
t=sci2exp(a [,nam] [,lmax])
```

PARAMETERS :

`a` : a scilab variable, may be
- constant,
- polynomial
- string matrix
- list
- boolean matrix
`nam` : character string
`t` : vector of string, contains the expression or instruction definition
`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 variable to an instruction if `nam` is given or to an expression .

EXAMPLE :

```
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')
```

3.0.426 sci2map _____ Scilab to Maple variable conversion

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
txt=[sci2map([1 2;3 4],'a');
     sci2map(%s^2+3*%s+4,'p')]
```

3.0.427 setmenu _____ interactive button or menu activation

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
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` 199, `unsetmenu` 248, `addmenu` 182

3.0.428 sinh _____ hyperbolic sine

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
asinh(sinh([0,1,%i]))
```

SEE ALSO : `asinh` 184

3.0.429 sinhm _____ matrix hyperbolic sine

CALLING SEQUENCE :

```
t=sinhm(x)
```

PARAMETERS :

`x,t` : real or complex square matrix

DESCRIPTION :
sinhm is the matrix hyperbolic sine of the matrix x. `t=(expm(x)-expm(-x))/2`

EXAMPLE :

```
A=[1,2;2,3]
asinhm(sinhm(A))
A(1,1)=%i;sinhm(A)-(expm(A)-expm(-A))/2   //Complex case
```

SEE ALSO : `sinh` 226

3.0.430 sin _____ sine function

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
asin(sin([1,0,%i]))
```

SEE ALSO : `sinm` 227

3.0.431 sinm _______________ matrix sine function

CALLING SEQUENCE :

```
t=sinm(x)
```

PARAMETERS :

x : real or complex square matrix

DESCRIPTION :
sinm(x) is matrix sine of x matrix.

EXAMPLE :

```
A=[1,2;2,4];
sinm(A)+0.5*%i*(expm(%i*A)-expm(-%i*A))
```

SEE ALSO : sin 226, asinm 185

3.0.432 smooth _______________ smoothing by spline functions

CALLING SEQUENCE :

```
[pt]=smooth(ptd [,step])
```

PARAMETERS :

ptd : (2xn) real vector
step : real (discretization step of abscissae) (default=0.01*magnitude(v))
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

EXAMPLE :

```
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 232, interp 209, interpln 208

3.0.433 solve _______________ symbolic linear system solver

CALLING SEQUENCE :

```
[x]=solve(A,b)
```

PARAMETERS :

A,b,c : matrix (resp. vectors) of character strings

DESCRIPTION :
solves A*x = b when A is an upper triangular matrix made of character strings.

EXAMPLE :

```
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 244

3.0.434 sort _______________ decreasing order sorting

CALLING SEQUENCE :

```
[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 :
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(:). [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 matrices or vectors are sorted w.r.t their magnitude.
y=sort(A) is valid when A is a sparse vector. Column/row sorting is not implemented for sparse matrices.

EXAMPLE :

```
[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 29

3.0.435 sp2adj _ converts sparse matrix into adjacency form

CALLING SEQUENCE :

[xadj,adjncy,anz]= sp2adj(A)

PARAMETERS :

```
.TP 7
A
:  real or complex sparse matrix (nz non-zero entries)
.TP 7
xadj
:  integer vector of length (n+1).
.TP 7
adjncy
:  integer vector of length nz containing the row indices
   for the corresponding elements in anz
.TP 7
anz
:  column vector of length nz, containing the non-zero
   elements of A
```

DESCRIPTION :

```
\fVsp2adj\fR converts a sparse matrix into its adjacency form (utility
fonction).
\fVA =  n x m\fR sparse matrix. \fVxadj, adjncy, anz\fR = adjacency
representation of \fVA\fR i.e:
.LP
\fVxadj(j+1)-xadj(j)\fR = number of non zero entries in row j.
\fVadjncy\fR = column index of the non zeros entries
in row 1, row 2,..., row n.
\fVanz\fR = values of non zero entries in row 1, row 2,..., row n.
\fVxadj\fR is a (column) vector of size n+1 and
\fVadjncy\fR is an integer (column) vector of size \fVnz=nnz(A)\fR.
\fVanz\fR is a real vector of size \fVnz=nnz(A)\fR.
```

EXAMPLE :

```
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 183, sparse 229, spcompack 230, spget 232

3.0.436 sparse _ sparse matrix definition

CALLING SEQUENCE :

```
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)
`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.

EXAMPLE :

```
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 203, spget 232, sprand 233, speye 231, lufact 443

3.0.437 spcompack _ converts a compressed adjacency representation

CALLING SEQUENCE :

adjncy = spcompak(xadj,xlindx,lindx)

PARAMETERS :

```
.TP 7
xadj
:  integer vector of length (n+1).
.TP 7
xlindx
:  integer vector of length n+1 (pointers).
.TP 7
lindx
:  integer vector
.TP 7
adjncy
:  integer vector
```

DESCRIPTION :

```
Utility fonction \fVspcompak\fR is used to convert a compressed adjacency
representation into standard adjacency representation.
```

EXAMPLE :

```
// 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 229, adj2sp 183, spget 232

3.0.438 speye ________ sparse identity matrix

CALING SEQUENCE :

```
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)`.

EXAMPLE :


```
eye(3,3)-full(speye(3,3))
```

SEE ALSO : sparse 229, full 203, eye 27, spzeros 233, spones 233

3.0.439 spget ________ retrieves entries of sparse matrix

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
sp=sparse([1,2;4,5;3,10],[1,2,3])
[ij,v,mn]=spget(sp);
```

SEE ALSO : sparse 229, sprand 233, speye 231, lufact 443

3.0.440 splin ________ spline function

CALLING SEQUENCE :

```
d=splin(x,f [,"periodic"])
```

PARAMETERS :

`x` : real vector
`f` : real vector of same size as `x`
`"periodic"` : string flag (a periodic spline is looked for)

DESCRIPTION :
Given values `fi` of a function f at given points `xi` (`fi=f(xi)`) this primitive computes a third order spline function S which interpolates the function `f`. The components of `x` must be in increasing order. For a periodic spline `f(1)` must equal `f(n)`; S is defined through the triple `(x,f,d)` where `d=spline(x,f)` is the vector of the estimated derivatives of S at `xi` (`fi=S(xi),di=S'(xi)`). This function should be used in conjunction with `interp`.

EXAMPLE :

```
x=0:0.5:10;f=sin(x);
d=splin(x,f);
S=interp(0:0.1:10,x,f,d);
plot2d(x',f',-1);
plot2d((0:0.1:10)',S',2,'000')
```

SEE ALSO : interp 209, smooth 227

3.0.441 spones _______________ sparse matrix

SYNTAX :
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;

EXAMPLE :

```
A=sprand(10,12,0.1);
sp=spones(A)
B = A~=0
bool2s(B)
```

SEE ALSO : `sparse` 229, `full` 203, `eye` 27, `speye` 231, `spzeros` 233

3.0.442 sprand _______________ sparse random matrix

CALING SEQUENCE :

```
sp=sprand(nrows,ncols,fill [,typ])
```

PARAMETERS :

`nrows` : integer (number of rows)
`ncols` : integer (number os 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 are generated. If `typ='normal'` normally distributed values are generated.

EXAMPLE :

```
W=sprand(100,1000,0.001);
```

SEE ALSO : `sparse` 229, `full` 203, `rand` 86, `speye` 231

3.0.443 spzeros _______________ sparse zero matrix

SYNTAX :
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,fill)` 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)`.

EXAMPLE :

```
sum(spzeros(1000,1000))
```

SEE ALSO : `sparse` 229, `full` 203, `eye` 27, `speye` 231, `spones` 233

3.0.444 sqrt _______________ square root

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
sqrt([2,4])
sqrt(-1)
```

SEE ALSO : `hat` 41, `sqrtm` 234

3.0.445 sqrtm _______________ matrix square root

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
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` 433, `sqroot` 461

3.0.446 squarewave _____ generates a square wave with period 2*%pi

CALLING SEQUENCE :

```
x=squarewave(t [,%])
```

PARAMETERS :

t : real vector, time discretization
x : real vector, the wave value at each time point in set (-1,+1)
% : 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.
EXAMPLE :

```
t=(0:0.1:5*%pi)';
plot2d1('onn',t,[2*sin(t),1.5*squarewave(t),squarewave(t,10)])
```

SEE ALSO : sin 226, cos 195

3.0.447 ssprint _____ pretty print for linear system

CALLING SEQUENCE :

```
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.
EXAMPLE :

```
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 688

3.0.448 ssrand _____ random system generator

CALLING SEQUENCE :

```
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".
EXAMPLE :

```
//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 241

3.0.449 st_deviation _____ standard deviation (row or column-wise) of vector/matrix entries

CALLING SEQUENCE :

```
y=st_deviation(x)
y=st_deviation(x,'r')
y=st_deviation(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,2)`) 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,1)`) is the columnwise st_deviation. It returns in each entry of the row vector `y` the standard deviation of each column of `x`.

EXAMPLE :

```
A=[1,2,10;7,7.1,7.01];
st_deviation(A)
st_deviation(A,'r')
st_deviation(A,'c')
```

SEE ALSO : `sum` 239, `median` 217, `mean` 217

3.0.450 strcat _____ catenate character strings

CALLING SEQUENCE :

```
[txt]=strcat(vstr [,strp])
```

PARAMETERS :

`txt,strp` : strings
`vstr` : vector of strings

DESCRIPTION :
catenates character strings : `txt=strs(1)+...+strs(n)`
`txt=strcat(strs,opt)` returns `txt=strs(1)+opt+...+opt+strs(n)`. The plus symbol does the same: `"a"+"b"` is the same as `strcat(["a","b"])`

EXAMPLE :

```
strcat(string(1:10),',')
```

SEE ALSO : `string` 100, `strings` 100

3.0.451 strindex _____ search position of a character string in an other string.

CALLING SEQUENCE :

```
ind=strindex(str1,str2)
```

PARAMETERS :

`str1` : a character string. The string where to search occurrences of `str2`
`str2` : a character string or character string vector . The string(s) to search in `str1`
`ind` : vector of indexes

DESCRIPTION :
`strindex` searches indexes where `str2(i)` is found in `str1` for each `k` it exist a `ishuch that part(str1,ind(k)+(0:length(str2(i))-1))` is the same string than `str2(i)`
When str2 is a vector and str2

EXAMPLE :

```
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'])
```

SEE ALSO : `string` 100, `strings` 100

3.0.452 stripblanks _____ strips leading and trailing blanks of strings

CALLING SEQUENCE :

```
txt=stripblanks(txt)
```

PARAMETERS :

`txt` : string or matrix of strings

DESCRIPTION :
stripblanks strips leading and trailing blanks of strings

EXAMPLE :

```
a='  123   ';
'!'+a+'!'
'!'+stripblanks(a)+'!'
a=['  123   ',' xyz']
strcat(stripblanks(a))
```

3.0.453 strsubst _____ substitute a character string by another in a character string.

CALLING SEQUENCE :

```
str=strsubst(str1,str2,str3)
```

PARAMETERS :

`str1` : a matrix of character string. The strings where to search occurrences of `str2`
`str2` : a character string. The string to search in `str1`.
`str3` : a character string. The replacement string.
`str` : a matrix of character string. The result of the substitution on `str2` by `str3` in `str1`

DESCRIPTION :
`strsubst` replaces all occurrences of `str2` in `str1` by `str3`.

EXAMPLE :

```
strsubst('SCI/demos/scicos','SCI','.')
strsubst('SCI/demos/scicos','/',' ')
```

SEE ALSO : `string` 100, `strings` 100

3.0.454 subf ______ symbolic subtraction

CALLING SEQUENCE :

```
["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.

EXAMPLE :

```
subf('0','a')
subf('2','1')
subf('a','0')
```

SEE ALSO : `mulf` 219, `ldivf` 212, `rdivf` 222, `eval` 203, `evstr` 21

3.0.455 sum ___ sum (row sum, column sum) of vector/matrix entries

CALLING SEQUENCE :

```
y=sum(x)
y=sum(x,'r') or y=sum(x,1)

y=sum(x,'c') or y=sum(x,2)
```

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. It returns in each entry of the row vector `y` the sum of the rows of `x` (The sum is performed on the row indice : `y(j)= sum(x(i,j),i=1,m)`).
`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 of the columns of `x` (The sum is performed on the column indice: `y(i)= sum(x(i,j),j=1,n)`)).

EXAMPLE :

```
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` 197, `prod` 221

3.0.456 sysconv ______ system conversion

CALLING SEQUENCE :

```
[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`

EXAMPLE :

```
s1=ssrand(1,1,2);
s2=ss2tf(s1);
[s1,s2]=sysconv(s1,s2);
```

SEE ALSO : `syslin` 241, `ss2tf` 299, `tf2ss` 302

3.0.457 sysdiag ______ block diagonal system connection

CALLING SEQUENCE :

```
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` 9, `insertion` 46, `feedback` 271

3.0.458 syslin _____ linear system definition

CALLING SEQUENCE :

```
[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 x  = A*x + B*u
  y  = C*x + D*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`
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` 102, `lsslist` 57, `rlist` 90, `ssrand` 235, `ss2tf` 299, `tf2ss` 302, `dscr` 268, `abcd` 254

3.0.459 tanh _____ hyperbolic tangent

CALLING SEQUENCE :

```
t=tanh(x)
```

DESCRIPTION :
the elements of `t` are the hyperbolic tangents of the elements of `x`

EXAMPLE :

```
x=[1,%i,-1,-%i]
tanh(x)
sinh(x)./cosh(x)
```

SEE ALSO : `atanh` 186, `tan` 243, `tanhm` 242

3.0.460 tanhm _____ matrix hyperbolic tangent

CALLING SEQUENCE :

```
t=tanhm(x)
```

PARAMETERS :

`x,t` : real or complex square matrix

DESCRIPTION :
tanhm is the matrix hyperbolic tangent of the matrix x.

SEE ALSO : `tanh` 242

3.0.461 tan — tangent

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
x=[1,%i,-1,-%i]
tan(x)
sin(x)./cos(x)
```

SEE ALSO : `atan` 186, `tanm` 243

3.0.462 tanm — matrix tangent

CALLING SEQUENCE :

```
[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

EXAMPLE :

```
A=[1,2;3,4];
tanm(A)
```

SEE ALSO : `tan` 243, `expm` 433, `sinm` 227, `atanm` 187

3.0.463 timer — cpu time

CALLING SEQUENCE :

```
timer()
```

DESCRIPTION :
Returns the CPU time from the preceding call to `timer()`.

EXAMPLE :

```
timer();A=rand(100,100);timer()
```

SEE ALSO : `unix_g` 104

3.0.464 toeplitz — toeplitz matrix

CALING SEQUENCE :

```
A=toeplitz(c [,r])
```

PARAMETERS :

`a,c,r` : constant, polynomial or character chain 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.

EXAMPLE :

```
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` 59

3.0.465 trfmod — poles and zeros display

CALLING SEQUENCE :

```
[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` 81, `simp` 421

3.0.466 trianfml — symbolic triangularization

CALLING SEQUENCE :

```
[f [,sexp]]=trianfml(f [,sexp])
```

DESCRIPTION :
Triangularization of the symbolic matrix `f` ; triangularization is performed by elementary row operations; `sexp` is a set of common expressions stored by the algorithm.

EXAMPLE :

```
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` 181, `mulf` 219, `solve` 227, `trisolve` 245

3.0.467 tril ________ lower triangular part of matrix

CALLING SEQUENCE :

```
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).

EXAMPLE :

```
s=poly(0,'s');
tril([s,s;s,1])
tril([1/s,1/s;1/s,1])
```

SEE ALSO : `triu` 246, `ones` 75, `eye` 27, `diag` 200

3.0.468 trisolve ________ symbolic linear system solver

CALLING SEQUENCE :

```
[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`.

EXAMPLE :


```
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` 244, `solve` 227

AUTHOR : F.D, S.S

3.0.469 triu ________ upper triangle

DESCRIPTION :
Upper triangle. See tril.

3.0.470 typeof ________ object type

CALLING SEQUENCE :

```
[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
`"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.

EXAMPLE :

```
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)
```

SEE ALSO : `type` 103, `strings` 100, `syslin` 241, `poly` 81

3.0.471 union _________ extract union components of a vector

CALLING SEQUENCE :

```
[v, [ka,kb]]=union(a,b)
```

PARAMETERS :

`a` : vector of real numbers or strings
`b` : vector of real numbers or strings
`v` : row vector of real 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(:)]`.
`[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)`.

EXAMPLE :

```
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` 247, `sort` 228

3.0.472 unique _________ extract unique components of a vector

CALLING SEQUENCE :

```
[N, [k]]=unique(M)
```

PARAMETERS :

`M` : vector of real numbers or strings
`N` : vector of real numbers or strings
`k` : vector of integers

DESCRIPTION :
`unique(M)` returns a vector which retains the unique entries of `M` in ascending order.
If required the output argument `k` contains the position of the first encountered unique entries.

EXAMPLE :

```
M=round(2*rand(20,1));

unique(M)
[N,k]=unique(M)

unique(string(M))
[N,k]=unique(string(M))
```


SEE ALSO : `union` 247, `sort` 228, `lex_sort` 213

3.0.473 unsetmenu _________ interactive button or menu or submenu de-activation

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
//addmenu('foo')
//unsetmenu('foo')
//unsetmenu('File',2)
```

SEE ALSO : `delmenu` 199, `setmenu` 225, `addmenu` 182

3.0.474 x_choices _________ interactive Xwindow choices through toggle buttons

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
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));
```

3.0.475 x_choose interactive Xwindow choice

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
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_choices` 248, `x_mdialog` 250, `getvalue` 204, `unix_g` 104

3.0.476 x_dialog Xwindow dialog

CALLING SEQUENCE :

```
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

EXAMPLE :

```
//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` 250, `x_matrix` 250, `evstr` 21, `execstr` 23

3.0.477 xgetfile dialog to get a file path

CALLING SEQUENCE :

```
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

EXAMPLE :

```
xgetfile()
xgetfile('*.sci','SCI/macros/xdess')
xgetfile(title='Choose a file name ')
```

SEE ALSO : `x_dialog` 249, `file` 28, `read` 88, `write` 111, `exec` 22, `getf` 39

3.0.478 x_matrix Xwindow editing of matrix

CALLING SEQUENCE :

```
[result]=x_matrix(label,matrix-init)
```

PARAMETERS :

`label` : character string (name of matrix)
`matrix-init` : real matrix

DESCRIPTION :
For reading or editing a matrix .

EXAMPLE :

```
//m=evstr(x_matrix('enter a  3x3 matrix ',rand(3,3)))
```

SEE ALSO : `x_mdialog` 250, `x_dialog` 249

3.0.479 x_mdialog Xwindow dialog

CALLING SEQUENCE :

```
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 '],[' ';' '])


n=5;m=4;mat=rand(n,m);
row='row';labelv=row(ones(1,n))+string(1:n)
col='col';labelh=col(ones(1,m))+string(1:m)
new=evstr(x_mdialog('Matrix to edit',labelv,labelh,string(mat)))
```

SEE ALSO : `x_dialog` 249, `x_choose` 249, `x_message` 251, `getvalue` 204, `evstr` 21, `execstr` 23

3.0.480 x_message — X window message

CALLING SEQUENCE :

```
[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, user-guide ...)

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` 249, `x_mdialog` 250

3.0.481 zeros — matrix made of zeros

CALLING SEQUENCE :

```
[y]=zeros(m,n)
[y]=zeros(x)
```

PARAMETERS :

`x,y` : matrices
`m,n` : integers

DESCRIPTION :
Matrix made of zeros (same as `0*ones`).

`zeros(m,n)` : for an `(m,n)` matrix.
`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!

If `x` is a `syslin` list (linear system in state-space or transfer form), `zeros(x)` is also valid and returns a zero matrix.

EXAMPLE :

```
zeros(3)
zeros(3,3)
```

SEE ALSO : `eye` 27, `ones` 75, `spzeros` 233

Chapter 4

General System and Control macros

4.0.482 abcd _____ state-space matrices

CALLING SEQUENCE :

```
[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)`
.

EXAMPLE :

```
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` 241, `ssrand` 235

4.0.483 abinv _____ AB invariant subspace

CALLING SEQUENCE :

```
[X,dims,F,U,k,Z]=abinv(Sys,alfa,beta,flag)
```

PARAMETERS :

`Sys` : `syslin` list containing the matrices `[A,B,C,D]`.
`alfa` : (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 `alfa` is given as input, the `dimR` controllable modes of `(A+BF)` in `V` are set to `alfa` (or to `[alfa(1), alfa(2), ...]`. (`alfa` can be a vector (real or complex pairs) or a (real) number). Default value `alfa=-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 `alfa(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)
```

EXAMPLE :

```
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);alfa=-1;beta=-2;
[X,dims,F,U,k,Z]=abinv(Sys,alfa,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.
alfa=-1;beta=-2;
[X,dims,F,U,k,Z]=abinv(sl,alfa,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,alfa,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'))
```

AUTHOR : F.D.

SEE ALSO : cainv 259, st_ility 300, ssrand 235, ss2ss 297, ddp 266

4.0.484 arhnk _______ Hankel norm approximant

CALLING SEQUENCE :

```
[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].
EXAMPLE :

```
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 271, equil1 270, hankelsv 316

4.0.485 arl2 ___ SISO model realization by L2 transfer approximation

CALLING SEQUENCE :

```
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 :

$$y(z) = y(1)(\frac{1}{z}) + y(2)(\frac{1}{z})^2 + \ldots + y(ny)(\frac{1}{z})^n y$$

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')
EXAMPLE :

```
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 417, imrep2ss 277, time_id 302, armax 403, frep2tf 273

4.0.486 balreal _______ balanced realization

CALLING SEQUENCE :

```
[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.
EXAMPLE :

```
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 265, obs_gram 287, hankelsv 316, equil 271, equil1 270

4.0.487 bilin ______ general bilinear transform

CALLING SEQUENCE :

```
[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])`

EXAMPLE :

```
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` 414, `cls2dls` 262

4.0.488 cainv ______ Dual of abinv

CALLING SEQUENCE :

```
[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 detectability)
//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` 254, `dt_ility` 269, `ui_observer` 304

4.0.489 calfrq ______ frequency response discretization

CALLING SEQUENCE :

```
[frq,split]=calfrq(h,[fmin,fmax])
```

PARAMETERS :

`h` : SISO linear system (`syslin` list)
`fmin,fmax` : real scalars (min and max frequencies)
`frq` : row vector (discretization of interval)
`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.
Default values for `fmin` and `fmax` are `1.d-3`, `1.d+3` if `h` is continuous-time or `1.d-3`, `1/(2*h('dt'))` if `h` is discrete-time.
Singularities are located between `frq(split(k))` and `frq(split(k)+1)` for `k>1`.

EXAMPLE :

```
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,spl]=calfrq(h1,0.01,1000);
rf=repfreq(h1,f1);
plot2d(real(rf)',imag(rf)')
```

SEE ALSO : `bode` 115, `black` 114, `nyquist` 138, `freq` 274, `repfreq` 291, `logspace` 215

4.0.490 canon ______ canonical controllable form

CALLING SEQUENCE :

```
[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).

EXAMPLE :

```
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` 287, `cont_mat` 263, `ctr_gram` 265, `contrss` 264, `ppol` 290, `contr` 264, `stabil` 299

AUTHOR : F.D.

4.0.491 cls2dls ______ bilinear transform

CALLING SEQUENCE :

```
[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)`.

EXAMPLE :

```
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` 414

4.0.492 colregul ______ removing poles and zeros at infinity

CALLING SEQUENCE :

```
[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` 277, `inv` 439, `rowregul` 293, `rowshuff` 455

AUTHOR : F. D. , R. N.

4.0.493 cont_frm ________ transfer to controllable state-space

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
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` 302, `canon` 261, `contr` 264

4.0.494 cont_mat ________ controllability matrix

CALLING SEQUENCE :

```
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` 265, `contr` 264, `canon` 261, `st_ility` 300

4.0.495 contr ________ controllability, controllable subspace

CALLING SEQUENCE :

```
[n [,U]]=contr(A,B [,tol])
[A1,B1,U,ind]=contr(A,B [,tol])
```

PARAMETERS :

`A, B` : real matrices
`tol` : may be the constant rtol or the 2 vector `[rtol atol]`
`rtol` :tolerance used when evaluating ranks (QR factorizations).
`atol` :absolute tolerance (the `B` matrix is assumed to be 0 if `norm(B)<atol`)
`n` : dimension of controllable subspace.
`U` : orthogonal change of basis which puts `(A,B)` in canonical form.
`A1` : block Hessenberg matrix
`B1` : is `U'*B`.
`ind` : 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.
`[A1,B1,U,ind]=contr(A,B)` returns the Hessenberg controllable form of `(A,B)`.

EXAMPLE :

```
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` 261, `cont_mat` 263, `unobs` 305, `stabil` 299

4.0.496 contrss ________ controllable part

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
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` 263, `ctr_gram` 265, `cont_frm` 263, `contr` 264

4.0.497 csim ______ simulation (time response) of linear system

CALLING SEQUENCE :

```
[y [,x]]=csim(u,t,sl,[x0])
```

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

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`)

EXAMPLE :

```
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'])
```

SEE ALSO : `syslin` 241, `dsimul` 269, `flts` 272, `ltitr` 282, `rtitr` 293, `ode` 350, `impl` 338

4.0.498 ctr_gram ______ controllability gramian

CALLING SEQUENCE :

```
[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).

$$Gc = \int_0^\infty e^{At} BB' e^{A't} dt \qquad Gc = \sum_0^\infty A^k BB' A'^k$$

EXAMPLE :

```
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` 270, `obs_gram` 287, `contr` 264, `cont_mat` 263, `cont_frm` 263, `contrss` 264

AUTHOR : S. Steer INRIA 1988

4.0.499 dbphi ______ frequency response to phase and magnitude representation

CALLING SEQUENCE :

```
[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` 291, `bode` 115

4.0.500 ddp ______ disturbance decoupling

CALLING SEQUENCE :

```
[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'.
`alfa` : 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).

EXAMPLE :

```
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.
alfa=-1;beta=-2;flag='ge';
[X,dims,F,U,k,Z]=abinv(Sys12,alfa,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',alfa,beta);
closed=syslin('c',A+B2*F,B1+B2*G,C1+D12*F,D11+D12*G);
ss2tf(closed)
```

AUTHOR : F.D.

SEE ALSO : `abinv` 254, `ui_observer` 304

4.0.501 des2tf _ descriptor to transfer function conversion

CALLING SEQUENCE :

```
[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);

EXAMPLE :

```
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` 435, `pol2des` 418, `tf2des` 326, `ss2tf` 299, `des2ss` 311, `rowshuff` 455

AUTHOR : F. D.

4.0.502 dscr _ discretization of linear system

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
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 241, flts 272, dsimul 269

4.0.503 dsimul _________ state space discrete time simulation

CALLING SEQUENCE :

```
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).

EXAMPLE :

```
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(20,1);
x2=dsimul(sl,u);  //Step response
```

SEE ALSO : syslin 241, flts 272, ltitr 282

4.0.504 dt_ility _________ detectability test

CALLING SEQUENCE :

```
[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).

EXAMPLE :

```
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 264, st_ility 300, unobs 305, stabil 299

4.0.505 equil1 _________ balancing (nonnegative) pair of matrices

CALLING SEQUENCE :

```
[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

EXAMPLE :

```
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 258, minreal 284, equil 271, hankelsv 316

AUTHOR : S. Steer 1987

4.0.506 equil _____ balancing of pair of symmetric matrices

CALLING SEQUENCE :

```
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.
EXAMPLE :

```
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` 270, `balanc` 427, `ctr_gram` 265

4.0.507 feedback _____ feedback operation

CALLING SEQUENCE :

```
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!)
EXAMPLE :

```
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` 319, `sysdiag` 240, `augment` 308, `obscont` 285

4.0.508 flts _____ time response (discrete time, sampled system)

CALLING SEQUENCE :

```
[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 `syslin` list containing the matrices of the following linear system
`sl=syslin('d',A,B,C,D)` (see `syslin`):

```
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))`) :

$$D(z) = D_0 + zD_1 + z^2D_2 + .. + z^pD_p$$

$$y_t = Cx_t + D_0u_t + D_1u_{t+1} + .. + D_pu_{t+p}$$

$$u = [u_0, u_1, ...u_n](input)$$

$$y = [y_0, y_1, ...y_{n-p}](output)$$

$$x = x_{n-p+1}$$

(final state, used as x0 at next call to flts)
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`).

$$past = \begin{bmatrix} u_{-nd} & \dots & u_{-1} \\ y_{-nd} & \dots & u_{-1} \end{bmatrix}$$

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
EXAMPLE :

```
sl=syslin('d',1,1,1);u=1:10;
y=flts(u,sl);
plot2d2("onn",(1:size(u,'c'))',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=(1-2*z)/(z^2+0.3*z+1)
u=zeros(1,20);u(1)=1;
imprep=flts(u,tf2ss(h));   //Impulse response
plot2d2("onn",(1:size(u,'c'))',imprep')
u=ones(1,20);
stprep=flts(u,tf2ss(h));   //Step response
plot2d2("onn",(1:size(u,'c'))',stprep')
//
// 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` 282, `dsimul` 269, `rtitr` 293

4.0.509 frep2tf _ transfer function realization from frequency response

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
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 of Sys
[frq,rep2]=repfreq(Sys2,frq); //Frequency response of Sys2
xbasc();bode(frq,[rep;rep2])   //Responses of Sys and Sys2
[sort(trzeros(Sys)),sort(roots(Sys2('num')))]  //zeros
[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` 277, `arl2` 257, `time_id` 302, `armax` 403, `frfit` 375

4.0.510 freq _ frequency response

CALLING SEQUENCE :

```
[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))

EXAMPLE :

```
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 291, horner 414

4.0.511 freson ______ peak frequencies

CALLING SEQUENCE :

```
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

EXAMPLE :

```
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 273, zgrid 178, h_norm 318

4.0.512 gfrancis ______ Francis equations for tracking

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
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 281, ppol 290

4.0.513 g_margin ______ gain margin

CALLING SEQUENCE :

```
[gm [,fr]]=g_margin(h)
```

PARAMETERS :

h : syslin list representing a linear system in state-space or transfer form

DESCRIPTION :
returns gm, the gain margin in dB of h (SISO plant) and fr, the achieved corresponding frequency in hz. The gain margin is values of the system gain at points where the nyquist plot crosses the negative real axis.

EXAMPLE :

```
h=syslin('c',-1+%s,3+2*%s+%s^2)
[g,fr]=g_margin(h)
[g,fr]=g_margin(h-10)
nyquist(h-10)
```

SEE ALSO : p_margin 289, black 114, chart 117, nyquist 138

4.0.514 imrep2ss ______ state-space realization of an impulse response

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
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 273, arl2 257, time_id 302, armax 403, markp2ss 283, ldiv 417

4.0.515 invsyslin ______ system inversion

CALLING SEQUENCE :

```
[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 293, inv 439

4.0.516 kpure ______ continuous SISO system limit feedback gain

CALLING SEQUENCE :

```
g=kpure(sys)
```

PARAMETERS :

sys : SISO linear system (syslin)
g : constant

DESCRIPTION :
kpure(sys) computes the gains g such that the system sys fedback by g (sys/.g) has poles on imaginary axis.

EXAMPLE :

```
s=poly(0,'s');
h=syslin('c',(s-1)/(1+5*s+s^2+s^3))
xbasc();evans(h)
g=kpure(h)
hf=h/.g(1)
roots(denom(hf))
```

SEE ALSO : evans 122, krac2 278

4.0.517 krac2 ______ continuous SISO system limit feedback gain

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
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 122, kpure 278

4.0.518 lin _____ linearization

CALLING SEQUENCE :

```
[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)
```

EXAMPLE :

```
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` 24, `derivat` 410

4.0.519 lqe _____ linear quadratic estimator (Kalman Filter)

CALLING SEQUENCE :

```
[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]`
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).
```

SEE ALSO : `lqr` 281

AUTHOR : F. D.

4.0.520 lqg2stan _____ LQG to standard problem

CALLING SEQUENCE :

```
[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.
EXAMPLE :

```
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` 281, `lqr` 281, `lqe` 279, `obscont` 285, `h_inf` 317, `augment` 308, `fstabst` 313, `feedback` 271

AUTHOR : F.D.

4.0.521 lqg ________ LQG compensator

CALLING SEQUENCE :

```
[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` 280, `lqr` 281, `lqe` 279, `h_inf` 317, `obscont` 285

AUTHOR : F.D.

4.0.522 lqr ________ LQ compensator (full state)

CALLING SEQUENCE :

```
[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).
EXAMPLE :

```
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=w(:,1:2);D12=w(:,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` 279, `gcare` 314, `leqr` 318

AUTHOR : F.D.

4.0.523 ltitr ________ discrete time response (state space)

CALLING SEQUENCE :

```
[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]`

EXAMPLE :

```
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` 293, `flts` 272

4.0.524 markp2ss ____ Markov parameters to state-space

CALLING SEQUENCE :

```
[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 `noutX(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),....
```

EXAMPLE :

```
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` 273, `tf2ss` 302, `imrep2ss` 277

4.0.525 minreal ____ minimal balanced realization

CALLING SEQUENCE :

```
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`.

EXAMPLE :

```
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` 284, `balreal` 258, `arhnk` 257, `equil` 271, `equil1` 270

AUTHOR : S. Steer INRIA 1987

4.0.526 minss ____ minimal realization

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
sl=syslin('c',[1 0;0 2],[1;0],[2 1]);
ssprint(sl);
ssprint(minss(sl))
```

SEE ALSO : `contr` 264, `minreal` 284, `arhnk` 257, `contrss` 264, `obsvss` 288, `balreal` 258

4.0.527 obscont _ observer based controller

CALLING SEQUENCE :

```
[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];

EXAMPLE :

```
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 290, lqg 281, lqr 281, lqe 279, h_inf 317, lft 319, syslin 241, feedback 271, observer 286

AUTHOR : F.D.

4.0.528 observer _ observer design

CALLING SEQUENCE :

```
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).

EXAMPLE :

```
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).
```

SEE ALSO : dt_ility 269, unobs 305, stabil 299

AUTHOR : F.D.

4.0.529 obs_gram ______ observability gramian

CALLING SEQUENCE :

```
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

$$Go = \int_0^\infty e^{A't} C'Ce^{At}dt \qquad Go = \sum_0^\infty A'^k C'CA^k$$

EXAMPLE :

```
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 265, obsvss 288, obsv_mat 287, lyap 445

4.0.530 obsv_mat ______ observability matrix

CALLING SEQUENCE :

```
[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 264, obsvss 288, obs_gram 287

4.0.531 obsvss ______ observable part

CALLING SEQUENCE :

```
[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 264, contrss 264, obsv_mat 287, obs_gram 287

4.0.532 pfss ______ partial fraction decomposition

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
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 447, bdiag 427, coffg 409, dtsi 312

AUTHOR : F.D.

4.0.533 phasemag _______ phase and magnitude computation

CALLING SEQUENCE :

```
[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))`

EXAMPLE :

```
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);
xbasc(0);
plot2d(frq',phasemag(rf,'c')');
xbasc(1);
plot2d(frq',phasemag(rf,'m')');
```

SEE ALSO : `repfreq` 291, `gainplot` 127, `atan` 186, `bode` 115

4.0.534 p_margin _______ phase margin

CALLING SEQUENCE :

```
[phm,fr]=p_margin(h)
phm=p_margin(h)
```

PARAMETERS :

`h` : SISO linear system (`syslin` list).
`phm` : phase margin (in degree)
`fr` : corresponding frequency (hz)

DESCRIPTION :
The phase margin is the values of the phase at points where the nyquist plot of `h` crosses the unit circle.

EXAMPLE :

```
h=syslin('c',-1+%s,3+2*%s+%s^2)
[p,fr]=p_margin(h)
[p,fr]=p_margin(h+0.7)
nyquist(h+0.7)
t=(0:0.1:2*%pi)';plot2d(sin(t),cos(t),-3,'000')
```

SEE ALSO : `chart` 117, `black` 114, `g_margin` 276, `nyquist` 138

AUTHOR : S. S.

4.0.535 ppol _______ pole placement

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
A=rand(3,3);B=rand(3,2);
F=ppol(A,B,[-1,-2,-3]);
spec(A-B*F)
```

SEE ALSO : `canon` 261, `stabil` 299

4.0.536 projsl _______ linear system projection

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
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 447

AUTHOR : F. D.

4.0.537 repfreq _ frequency response

CALLING SEQUENCE :

```
[ [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.

EXAMPLE :

```
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 115, freq 274, calfrq 260, horner 414, nyquist 138, dbphi 266

AUTHOR : S. S.

4.0.538 ricc _ Riccati equation

CALLING SEQUENCE :

```
[X]=ricc(A,B,C,"cont")
[X]=ricc(F,G,H,"disc")
```

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)

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.

EXAMPLE :

```
//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]=gschur(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]=gschur(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 323, ric_desc 323, schur 456, gschur 436

4.0.539 rowregul ____________ removing poles and zeros at infinity

CALLING SEQUENCE :

```
[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).

EXAMPLE :

```
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 277, colregul 262

AUTHOR : F. D. , R. N.

4.0.540 rtitr ____________ discrete time response (transfer matrix)

CALLING SEQUENCE :

```
[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)

EXAMPLE :

```
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 282, exp 432, flts 272

4.0.541 sm2des ______ system matrix to descriptor

CALLING SEQUENCE :

```
[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 297, sm2ss 296

4.0.542 sm2ss ______ system matrix to state-space

CALLING SEQUENCE :

```
[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 297

4.0.543 specfact ______ spectral factor

CALLING SEQUENCE :

```
[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)`

EXAMPLE :

```
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 315, sfact 421, fspecg 312

AUTHOR : F. D.

4.0.544 ss2des ____ (polynomial) state-space to descriptor form

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
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` 418, `tf2des` 326, `des2ss` 311

AUTHOR : F. D.

4.0.545 ss2ss ____ state-space to state-space conversion, feedback, injection

CALLING SEQUENCE :

```
[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:

$$\texttt{Sl1} = \begin{pmatrix} T^{-1}(A+BF)T & T^{-1}(B) \\ (C+DF)T & D \end{pmatrix}$$

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:

$$\texttt{Sl1} = \begin{pmatrix} T^{-1}(A+GC)T & T^{-1}(B+GD) \\ CT & D \end{pmatrix}$$

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

  $$\texttt{Sl1} = \begin{pmatrix} T^{-1}(A+GC)T & T^{-1}(B+GD,-G) \\ CT & (D,0) \end{pmatrix}$$

  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)))
  ```

EXAMPLE :

```
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` 290, `feedback` 271

4.0.546 ss2tf _______ conversion from state-space to transfer function

CALLING SEQUENCE :

```
[h]=ss2tf(sl)
[Ds,NUM,chi]=ss2tf(sl)
```

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.

EXAMPLE :

```
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 302, syslin 241, nlev 446, glever 435

4.0.547 stabil _______ stabilization

CALLING SEQUENCE :

```
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.

EXAMPLE :


```
// 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 300, contr 264, ppol 290

4.0.548 st_ility _______ stabilizability test

CALLING SEQUENCE :

```
[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).

EXAMPLE :

```
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 269, contr 264, stabil 299, ssrand 235

AUTHOR : S. Steer INRIA 1988

4.0.549 svplot ______ singular-value sigma-plot

CALLING SEQUENCE :

```
[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)
```

EXAMPLE :

```
x=logspace(-3,3);
y=svplot(ssrand(2,2,4));
xbasc();plot2d1("oln",x',20*log(y')/log(10));
xgrid(12)
xtitle("Singular values plot","(Rd/sec)", "Db");
```

AUTHOR : F.D

4.0.550 syssize ______ size of state-space system

CALLING SEQUENCE :

```
[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` 96

4.0.551 tf2ss ______ transfer to state-space

CALLING SEQUENCE :

```
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)`

EXAMPLE :

```
s=poly(0,'s');
H=[2/s,(s+1)/(s^2-5)];
Sys=tf2ss(H)
clean(ss2tf(Sys))
```

SEE ALSO : `ss2tf` 299, `tf2des` 326, `des2tf` 268

4.0.552 time_id ______ SISO least square identification

CALLING SEQUENCE :

```
[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

EXAMPLE :

```
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
```

AUTHOR : Serge Steer INRIA

SEE ALSO : imrep2ss 277, arl2 257, armax 403, frep2tf 273

4.0.553 trzeros _____ transmission zeros and normal rank

CALLING SEQUENCE :

```
[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 & 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.

EXAMPLE :


```
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 437, kroneck 441

4.0.554 ui_observer _____ unknown input observer

CALLING SEQUENCE :

```
[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).

EXAMPLE :

```
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);
```

AUTHOR : F.D.

SEE ALSO : cainv 259, ddp 266, abinv 254

4.0.555 unobs _____ unobservable subspace

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
A=diag([1,2,3]);C=[1,0,0];
unobs(A,C)
```

SEE ALSO : contr 264, contrss 264, canon 261, cont_mat 263, spantwo 459, dt_ility 269

4.0.556 zeropen _____ zero pencil

CALLING SEQUENCE :

```
[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 423, kroneck 441

Chapter 5

Robust control toolbox

5.0.557 augment — augmented plant

CALLING SEQUENCE :

```
[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,-(`
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(e
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

EXAMPLE :

```
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` 319, `sensi` 325

5.0.558 bstap _______ hankel approximant

CALLING SEQUENCE :

```
[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`

$$\|Sl - Q\|_\infty = \|T\|$$

where is the H-infinity norm of the Hankel operator associated with `Sl`.

SEE ALSO : `syslin` 241

5.0.559 ccontrg _______ central H-infinity controller

CALLING SEQUENCE :

```
[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` 313, `h_inf` 317

AUTHOR : P. Gahinet (INRIA)

5.0.560 colinout _______ inner-outer factorization

CALLING SEQUENCE :

```
[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` 241, `rowinout` 324

5.0.561 copfac _______ right coprime factorization

CALLING SEQUENCE :

```
[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` 241, `lcf` 318

5.0.562 dcf _______ double coprime factorization

CALLING SEQUENCE :

```
[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

$$\begin{pmatrix} XT & -YT \\ -NT & MT \end{pmatrix} * \begin{pmatrix} M & Y \\ M & X \end{pmatrix} = eye$$

`G` must be stabilizable and detectable.
SEE ALSO : `copfac` 310

5.0.563 des2ss — descriptor to state-space

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
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` 268, `glever` 435, `rowshuff` 455

5.0.564 dhnorm — discrete H-infinity norm

CALLING SEQUENCE :

```
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` 318, `linfn` 321

5.0.565 dtsi — stable anti-stable decomposition

CALLING SEQUENCE :

```
[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` 241, `pbig` 447, `psmall` 451, `pfss` 288

5.0.566 fourplan — augmented plant to four plants

CALLING SEQUENCE :

```
[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` 281, `lqg2stan` 280, `lqr` 281, `lqe` 279, `lft` 319

5.0.567 fspecg — stable factorization

CALLING SEQUENCE :

```
[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.

5.0.568 fstabst _____ Youla's parametrization

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
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 285, lft 319, lqg 281, lqg2stan 280

5.0.569 gamitg _____ H-infinity gamma iterations

CALLING SEQUENCE :

```
[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 309, h_inf 317

AUTHOR : P. Gahinet

5.0.570 gcare _____ control Riccati equation

CALLING SEQUENCE :

```
[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 314

5.0.571 gfare _____ filter Riccati equation

CALLING SEQUENCE :

```
[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 314

5.0.572 gtild _______ tilde operation

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
//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` 241, `horner` 414, `factors` 412

5.0.573 h2norm _______ H2 norm

CALLING SEQUENCE :

```
[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)

5.0.574 hankelsv _______ Hankel singular values

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
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` 258, `equil` 271, `equil1` 270

5.0.575 h_cl _______ closed loop matrix

CALLING SEQUENCE :

```
[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` 319

AUTHOR : F. D.

5.0.576 h_inf ______ H-infinity (central) controller

CALLING SEQUENCE :

```
[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` 313, `ccontrg` 309, `leqr` 318

AUTHOR : F.D. (1990)

5.0.577 h_inf_st ______ static H_infinity problem

CALLING SEQUENCE :

```
[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]`

AUTHOR : F.D.

5.0.578 h_norm ______ H-infinity norm

CALLING SEQUENCE :

```
[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` 321, `linf` 320, `svplot` 301

5.0.579 lcf ______ normalized coprime factorization

CALLING SEQUENCE :

```
[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`

AUTHOR : F. D.

5.0.580 leqr ______ H-infinity LQ gain (full state)

CALLING SEQUENCE :

```
[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 281

AUTHOR : F.D.

5.0.581 lft ______________ linear fractional transformation

CALLING SEQUENCE :

```
[P1]=LFT(P,K)
[P1]=LFT(P,r,K)
[P1,r1]=LFT(P,r,P#,r#)
```

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
P# : linear system (syslin list), implicitly partitioned into four blocks (two input ports and two output ports).

r# : 1x2 row vector, dimension of P#22

DESCRIPTION :
Linear fractional transform between two standard plants P and P# in state space form or in transfer form (syslin lists).
r= size(P22) r#=size(P22#)
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,P#,r#) returns the generalized (2 ports) lft of P and P#.
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.

EXAMPLE :

```
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 325, augment 308, feedback 271, sysdiag 240

5.0.582 linf ______________ infinity norm

CALLING SEQUENCE :

```
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 318, linfn 321

5.0.583 linfn — infinity norm

CALLING SEQUENCE :

```
[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` 318

AUTHOR : P. Gahinet

5.0.584 lqg_ltr — LQG with loop transform recovery

CALLING SEQUENCE :

```
[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:

$$J_{lqg} = E(\int_0^\infty [z(t)' * z(t) + ro^2 * u(t)' * u(t)]dt)$$

The lqg/ltr approach looks for `L,mu,H,ro` such that: J(lqg) = J(freq) where

$$J_{freq} = \int_0^\infty tr[SWW^* S^*] + tr[TT^*]dw$$

and

```
 S = (I + G*K)^(-1)
 T = G*K*(I+G*K)^(-1)
```

SEE ALSO : `syslin` 241

5.0.585 macglov — Mac Farlane Glover problem

CALLING SEQUENCE :

```
[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)`

AUTHOR : F. D.

5.0.586 nehari — Nehari approximant

CALLING SEQUENCE :

```
[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
```

5.0.587 parrot _______________ Parrot's problem

CALLING SEQUENCE :

```
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` 317

5.0.588 riccati _______________ Riccati equation

CALLING SEQUENCE :

```
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 : `ric_desc` 323

5.0.589 ric_desc _______________ Riccati equation

CALLING SEQUENCE :

```
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 (one input): `ric_descr(H)`
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 (two inputs): `ric_descr(H,E)`
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` 323

5.0.590 rowinout _______________ inner-outer factorization

CALLING SEQUENCE :

```
[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 241, colinout 310

5.0.591 sensi ______ sensitivity functions

CALLING SEQUENCE :

```
[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)];
```

EXAMPLE :

```
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 308, lft 319, h_cl 316

5.0.592 tf2des ______ transfer function to descriptor

CALLING SEQUENCE :

```
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

EXAMPLE :

```
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 418, tf2ss 302, ss2des 297, des2tf 268

Chapter 6

Non-linear tools (optimization and simulation)

6.0.593 bvode _ boundary value problems for ODE

CALLING SEQUENCE :

```
[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

$$\texttt{mstar} = \sum_{i=1}^{\texttt{ncomp}} \texttt{m}(i) \leq 40$$

`aleft` left end of interval
`aright` right end of interval
`zeta` `zeta(j)` gives j-th side condition point (boundary point). must have $\texttt{zeta}(j) \leq \texttt{zeta}(j+1)$. 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)` (= nonlin)
`= 0` if the problem is linear

`= 1 if the problem is nonlinear` `ipar(2)` = number of collocation points per subinterval (= k) where $\max \texttt{m}(i) \leq k \leq 7$. if `ipar(2)=0` then bvode sets $k = \max(\max \texttt{m}(i) + 1, 5 - \max \texttt{m}(i))$
`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 $0 < \texttt{ntol} < \texttt{mstar}$.
`ipar(5)` = dimension of `fspace` (= ndimf) a real work array. its size provides a constraint on nmax. choose ipar(5) according to the formula

$$\texttt{ipar}(5) \geq \texttt{nmax}n_s \quad \text{where} \quad n_s = 4 + 3\texttt{mstar} + (5 + k_d)k_d m + (2\texttt{mstar} - \texttt{nrec})2\texttt{mstar}$$

`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

$$\texttt{ipar}(6) \geq \texttt{nmax}n_i \quad \text{where} n_i = 3 + k_d m \quad k_d m = k_d + \texttt{mstar} \quad k_d = k\texttt{ncomp}$$

`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

$$\texttt{aleft} = x(1) < x(2) < \ldots < x(n) < x(n+1) = \texttt{aright}$$

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)` 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

$$1 \leq \mathtt{ltol}(1) < \mathtt{ltol}(2) < \ldots < \mathtt{ltol}(ntol) \leq mstar$$

`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

$$|z(v) - z(u)|_{ltol(j)} \leq \mathtt{tol}(j) * |z(u)|_{\mathtt{ltol}(j)} + \mathtt{tol}(j)$$

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

$$f(x, z(u(x))) = (f_1, \ldots, f_{\mathtt{ncomp}})^t$$

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

$$z(u(x)) = (z_1, \ldots, z_{\mathtt{mstar}})^t$$

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

$$df(i,j) = df_i/dz_j, \quad i = 1, \ldots, \mathtt{ncomp} \quad j = 1, \ldots, \mathtt{mstar}.$$

`gsub` name of subroutine for evaluating the i-th component of

$$g(x, z(u(x))) = g_i(zeta(i), z(u(zeta(i))))$$

at a point `x = zeta(i)` where $1 \leq i \leq \mathtt{mstar}$. 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

$$dg(i,j) = dg_i/dz_j, \quad j = 1, \ldots, \mathtt{mstar}.$$

`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, $\mathtt{aleft} \leq x \leq \mathtt{aright}$.

DESCRIPTION :

this package solves a multi-point boundary value problem for a mixed order system of ode-s given by

$$u_i^{(m(i))} = f(x; z(u(x))) \quad i = 1, \ldots, \mathtt{ncomp} \quad \mathtt{aleft} < x < \mathtt{aright}$$

$$g_j(zeta(j); z(u(zeta(j)))) = 0 \quad j = 1, \ldots, \mathtt{mstar} \quad \mathtt{mstar} = \sum_{i=1}^{\mathtt{ncomp}} m(i)$$

where $u = (u_1, u_2, \ldots, u_{\mathtt{ncomp}})^t$ is the exact solution vector $u_i^{(m(i))}$ is the mi=m(i) th derivative of u_i.

$$z(u(x)) = (u_1(x), u_1^{(1)}(x), \ldots, u_1^{(m_1-1)}(x), \ldots, u_{\mathtt{ncomp}}^{(m_{\mathtt{ncomp}}-1)}(x))$$

$f_i(x, z(u))$ is a (generally) nonlinear function of $z(u) = z(u(x))$. $g_j(zeta(j); z(u))$ is a (generally) nonlinear function used to represent a boundary condition. the boundary points satisfy

$$\mathtt{aleft} \leq \mathtt{zeta}(1) \leq \ldots \leq \mathtt{zeta}(\mathtt{mstar}) \leq \mathtt{aright}$$

the orders mi of the differential equations satisfy $1 \leq m(i) \leq 4$.

```
function [z,z1]=col1()
 fixpnt=0
 m=[4]
 ncomp=1
 aleft=1
 aright=2
 zeta=[1,1,2,2]
 ipar=0*ones(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;

function [df]=dfsub(x,z)
 df=[0,0,-6/x**2,-6/x]

function [f]=fsub(x,z)
 f=(1 -6*x**2*z(4)-6*x*z(3))/x**3
```

```
function [g]=gsub(i,z)
 g=[z(1),z(3),z(1),z(3)]
 g=g(i)

function [dg]=dgsub(i,z)
 dg=[1,0,0,0;0,0,1,0;1,0,0,0;0,0,1,0]
 dg=dg(i,:)

function [z,mpar]=guess(x)
 // unused here
 z=0
 mpar=0

function [u]=trusol(x)
  u=0*ones(4,1)
      U(1) = .25* (10.*LOG(2.)-3.) * (1.-X) +0.5* (1./X+ (3.+X)*LOG(X) - X)
      U(2) = -.25* (10.*LOG(2.) - 3.) + .5 * (-1./X/X + LOG(X) + (3.+X)/X - 1.)
      U(3) = .5 * (2./X**3 + 1./X -3./X/X)
      U(4) = .5 * (-6./X**4 - 1./X/X + 6./X**3)
```

AUTHOR : 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

Fotran subroutine colnew.f

SEE ALSO : `fort` 31, `link` 54, `external` 24, `ode` 350, `dassl` 333

6.0.594 colnew _ boundary value problems for ODE

CALLING SEQUENCE :

This function has been renamed bvode.

6.0.595 dasrt _ DAE solver with zero crossing

CALLING SEQUENCE :

```
[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)`.

`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.

`list` : 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,...)`.

`string` : it must refer to the name of a fortran subroutine (see source code of `fresd.f`).

`jac` : external (function or list or string). Computes the value of `dg/dy+cj*dg/dydot` for a given value of parameter `cj`

`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

`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,x1,x2,...)`

`jac` still returns `dg/dy+cj*dg/dydot` as a function of `(t,y,ydot,cj,x1,x2,...)`.

`character string` : it must refer to the name of a fortran subroutine (see source code of `jacdd.f`).

`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.

`function` : Its calling sequence must be `surf(t,y)`

`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,...)`

`character string` : it must refer to the name of a fortran subroutine (see source code of `fsurfd.f`) in directory `SCDIR/default`

`info` : list which contains `7` elements:

`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

EXAMPLE :

```
//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;
info=list([],0,[],[],[],0,0);
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,info);
//(Should return nn=[2.4698972 2])
```

SEE ALSO : `ode` 350, `dassl` 333, `impl` 338, `fort` 31, `link` 54, `external` 24

6.0.596 dassl ______ differential algebraic equation

CALLING SEQUENCE :

```
[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)`.
- `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.
- `list` : 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,...)`.
- `string` : it must refer to the name of a fortran subroutine (see source code of of `Ex-dassl.f` in routines/defaut/).
- `jac` : external (function or list or string). Computes the value of `dg/dy+cj*dg/dydot` for a given value of parameter `cj`
- `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
- `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,x1,x2,...)`

  `jac` still returns `dg/dy+cj*dg/dydot` as a function of `(t,y,ydot,cj,x1,x2,...)`.
- `character string` : it must refer to the name of a fortran subroutine (see source code of `Ex-dassl.f` in routines/defaut/).
- `info` : list which contains `7` elements:
- `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
```

Detailed examples are given in SCIDIR/tests/dassldasrt.tst

EXAMPLES :

```
 deff('[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']);
 deff('[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       ]'])

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];
```

```
info=list([],0,[],[],[],0,0);
y=dassl([y0,yd0],0,t,chemres,info);
info(2)=1;
y=dassl([y0,yd0],0,4d10,chemres,info);
y=dassl([y0,yd0],0,4d10,chemres,chemjac,info);
```

SEE ALSO : `ode` 350, `dasrt` 331, `impl` 338, `fort` 31, `link` 54, `external` 24

6.0.597 datafit ____ Parameter identification based on measured data

CALLING SEQUENCE :

```
[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`.

EXAMPLE :


```
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]=datafit(G,Z,[3;5;10])

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')
//same probleme with a known
deff('e=G(p,z,a)','b=p(1),c=p(2),y=z(1),x=z(2),e=y-FF(x)')
[p,err]=datafit(list(G,a),Z,[5;10])

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]=datafit(G,DG,Z,[3;5;10])
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` 355, `leastsq` 344

6.0.598 fit_dat ____ Parameter identification based on measured data

CALLING SEQUENCE :

```
[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)`

EXAMPLE :

```
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 355, datafit 335

6.0.599 fsolve _______ find a zero of a system of n nonlinear functions

CALLING SEQUENCE :

```
[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 fortran routine which must be linked to Scilab (see `fsolvf.f` for the calling sequence). Dynamic 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 fortran routine which must be linked to Scilab (see `fsolvj.f` for the calling sequence). Dynamic link is possible (help `link`).

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 24, quapro 356, linpro 345, optim 355

6.0.600 impl _______ differential algebraic equation

DESCRIPTION :

```
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 `adda` 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` 333, `ode` 350, `external` 24

6.0.601 int2d ______ definite 2D integral by quadrature and cubature method

CALLING SEQUENCE :

```
[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` 342, `intl` 343, `int3d` 340, `intg` 342, `mesh2d` 494

REFERENCES :

Fortran routine twodq,Authors: Kahaner,D.K.,N.B.S., Rechard,O.W.,N.B.S., Barnhill,Robert,Univ. of UTAH

6.0.602 int3d ______ definite 3D integral by quadrature and cubature method

CALLING SEQUENCE :

```
[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,EPSR`

`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.

EXAMPLES :

```
X=[0;1;0;0];
Y=[0;0;1;0];
Z=[0;0;0;1];
deff('v=f(xyz,numfun)','v=exp(xyz''*xyz)')
[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.)
```

SEE ALSO : `intc` 342, `intl` 343, `int3d` 340

REFERENCES :

Fortran routine dqtet.f

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
```


6.0.603 intc — Cauchy integral

CALING SEQUENCE :

```
[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` 342, `intl` 343

AUTHOR : F. D.

6.0.604 intg — definite integral

CALLING SEQUENCE :

```
[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: 0
`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 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 subroutine (see source code of `fintg.f`)

EXAMPLE :

```
deff('[y]=f(x)','y=x*sin(30*x)/sqrt(1-((x/(2*%pi))^2))')
exact=-2.5432596188;
abs(exact-intg(0,2*%pi,f))
// See file routines/default/Ex-intg.f
abs(exact-intg(0,2*%pi,'intgex'))
```

SEE ALSO : `intc` 342, `intl` 343

6.0.605 intl _______ Cauchy integral

CALLING SEQUENCE :

```
[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` 342

AUTHOR : F. D.

6.0.606 karmarkar _______ karmarkar (test program)

CALLING SEQUENCE :

```
[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

```
c'*x
```

under constraints:

```
a*x = b
x>=0
```

EXAMPLE :

```
// 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);
```


6.0.607 leastsq _______ Solves non-linear least squares problems

CALLING SEQUENCE :

```
[f,xopt]=leastsq([imp,] fun [,Dfun],x0)
[f,[xopt,[gradopt]]]=leastsq(fun [,Dfun],[contr],x0,['algo'],[df0,[mem]],
     ,[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.
`fun` : external, i.e Scilab function or string (`fun` is the function defining the least square probleme: see below.
`x0` : real vector (initial value of variable to be minimized).
`f` : value of optimal least square value.
`xopt` : best value of `x` found.
`contr` : `'b',binf,bsup` with `binf` and `bsup` real vectors with same dimension as `x0`. `binf` and `bsup` are lower and upper bounds on `x`.
`algo` : A string with possible values :`'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).
`gradopt` : gradient of `fun` at `xopt`

DESCRIPTION :
Non-linear optimization routine for programs without constraints or with bound constraints:

```
min sum(f(x).^2  w.r.t x.
```

`fun` is an "external" i.e function, or list or Fortran routine (see "external"). This external must return (`f(x)`) for a given `x`.

If `fun` is a function, the calling sequence for `fun` must be:

```
[f]=fun(x, [optional parameters]).
```

Here, `fun` is a function from R^n to R^m which returns `f(x)`, a real vector (value of function at `x`)
If fun is defined by a Fortran or C routine first argument must be a list: ist(fun_name,m,...) If `fun_name` is a character string, it refers to the name of the routine which must be linked to Scilab. Here, the generic calling sequence for the Fortran subroutine is: `subroutine fun(m,n,x,td,f)`

`n` is the dimension of `x`, `x` is an `n` vector, `td` are working arrays which may also be used to pass parameters

If `fun` is given as a Fortran routine, it is possible to initialize parameters or to send Scilab variables to this routine using sequence of arguments `'td' , valtd`. Then, the Fortran function `fun(m,n , x, f, td)` is evaluated with `td=valtd`. Thus, the Scilab variables `valtd` are sent to the Fortran function `fun`.

`Dfun` is an "external".This external must return a matrix `g` such as (`g(i,j)=dfi/dxj`) for a given `x`.

If `Dfun` is a function, the calling sequence for `fun` must be:

`[g]=Dfun(x, [optional parameters])`.

If Dfun is defined by a Fortran or C routine first argument must be a list: ist(fun_name,m,...) If `fun_name` is a character string, it refers to the name of the routine which must be linked to Scilab. Here, the generic calling sequence for the Fortran subroutine is: `subroutine dfun(m,n,x,td,g)`

EXAMPLES :

```
a=rand(3,2);b=[1;1;1];x0=[1;-1];
deff('f=fun(x,a,b)','f=a*x-b');
deff('g=dfun(x,a,b)','g=a');

[f,xopt]=leastsq(fun,x0)      //Simplest call
xopt-a\b  //compare with linear algebra solution

[f,xopt]=leastsq(fun,dfun,x0)      //specify gradient

[f,xopt]=leastsq(list(fun,[1 2;3 4],[1;2]),x0)

deff('f=fun(x,a,b)','f=exp(a*x)-b');
deff('g=dfun(x,a,b)','g=a.*(exp(a*x)*ones(1,size(a,2)))');

[f,xopt]=leastsq(list(fun,[1 2;3 4],[1;2]),x0)
```

SEE ALSO : `external` 24, `quapro` 356, `linpro` 345

6.0.608 linpro ____ linear programming solver

CALLING SEQUENCE :

```
[x,lagr,f]=linpro(p,C,b [,x0])
[x,lagr,f]=linpro(p,C,b,ci,cs [,x0])
[x,lagr,f]=linpro(p,C,b,ci,cs,mi [,x0])
[x,lagr,f]=linpro(p,C,b,ci,cs,mi,x0 [,imp])
```

PARAMETERS :

- `p` : real (column) vector (dimension `n`)
- `C` : real matrix (dimension `(mi + md) x n`) (If no constraints are given, you can set `C = []`)
- `b` : RHS vector (dimension `1 x (mi + 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) = -(% eps)^(-1)`.
- `cs` : (column) vector of upper-bounds. (Same remarks as above).

- `mi` : number of equality constraints (i.e. `C(1:mi,:)*x = b(1:mi)`)
- `x0` : either an initial guess for `x` or one of the character strings `'v'` or `'g'`. If `x0='v'` the calculated initial feasible point is a vertex. If `x0='g'` the calculated initial feasible point is arbitrary.
- `imp` : verbose option (optional parameter) (Try `imp=7,8,...`)
- `x` : optimal solution found.
- `f` : optimal value of the cost function (i.e. `f=p'*x`).
- `lagr` : vector of Lagrange multipliers. If lower and upper-bounds `ci,cs` are provided, `lagr` has `n + mi + md` components and `lagr(1:n)` is the Lagrange vector associated with the bound constraints and `lagr (n+1 : n + mi + 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 `mi + md` components.

DESCRIPTION :

`[x,lagr,f]=linpro(p,C,b [,x0])`

Minimize `p'*x`

under the constraints

```
C*x <= b
```

`[x,lagr,f]=linpro(p,C,b,ci,cs [,x0])`

Minimize `p'*x`

under the constraints

```
C*x <= b        ci <= x <= cs
```

`[x,lagr,f]=linpro(p,C,b,ci,cs,mi [,x0])`

Minimize `p'*x`

under the constraints

```
C(j,:) x = b(j),  j=1,...,mi
C(j,:) x <= b(j), j=mi+1,...,mi+md
ci <= x <= cs
```

If no initial point is given the program computes a feasible initial point which is a vertex of the region of feasible points if `x0='v'`.

If `x0='g'`, the program computes a feasible initial point which is not necessarily a vertex. This mode is advisable when the quadratic form is positive definite and there are a few constraints in the problem or when there are large bounds on the variables that are security bounds and very likely not active at the optimal solution.

EXAMPLE :

```
//Find x in R^6 such that:
//C1*x = b1  (3 equality constraints i.e mi=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 p'*x with
p=[1;2;3;4;5;6]
//No initial point is given: x0='v';
```

```
C=[C1;C2]; b=[b1;b2] ; mi=3; x0='v';
[x,lagr,f]=linpro(p,C,b,ci,cs,mi,x0)
// Lower bound constraints 3 and 4 are active and upper bound
// constraint 5 is active --> lagr(3:4) < 0 and lagr(5) > 0.
// Linear (equality) constraints 1 to 3 are active --> lagr(7:9) <> 0
```

SEE ALSO : quapro 356

AUTHOR : E. Casas, C. Pola Mendez

6.0.609 lmisolver _____ linear matrix inequation solver

CALLING SEQUENCE :

```
[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)
`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,maxite`

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.

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.

EXAMPLE :

```
//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+Q2)),OBJ= -sum(diag(X))  ');
X=lmisolver(list(zeros(A1)),evalf);X=X(1)
[Y,Z,c]=evalf(X)
```

SEE ALSO : lmitool 348

6.0.610 lmitool _____ tool for solving linear matrix inequations

CALLING SEQUENCE :

```
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 347

6.0.611 odedc _____ discrete/continuous ode solver

CALLING SEQUENCE :

```
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`).

EXAMPLE :

```
//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` 350, `odeoptions` 353, `csim` 265, `external` 24

6.0.612 ode_discrete ___ ordinary differential equation solver, discrete time simulation

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
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` 350

6.0.613 ode ___ ordinary differential equation solver

CALLING SEQUENCE :

```
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 the solution vector `y=[y(t(1)),y(t(2)),...]`.

The input `f` to `ode` 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 function, its syntax must be as follows:

```
ydot = f(t,y)
```

where `t` is a real scalar (time) and `y` a real vector (state). This function is the RHS of the differential equation dy/dt=f(t,y).

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. This routine must have

the following calling sequence: `f(n,t,y,ydot)`. It can be dynamically linked to Scilab by the `link` function. Examples of such programs can be seen in the files `SCIDIR/routines/default/README` and `SCIDIR/routines/default/Ex-ode.f`.

The `f` argument can also be a list: if `ode(y0,t0,t,lst)` is the command, then `lst` must be a list with the following structure:

```
lst=list(f,u1,u2,...un)
```

where `f` is a function with syntax:

```
ydot = f(t,y,u1,u2,...,un)
```

this allows to use parameters as the arguments of `f`.

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 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 as follows:

```
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: `jac(n,t,y,ml,mu,J,nrpd)`. In most cases you have not to refer `ml`, `mu` and `nrpd` (see source code in `SCIDIR/routines/default/Ex-ode.f` for an example).
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. 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 help.

EXAMPLE :

```
// Simple one dimension ODE
// dy/dt=y^2-y sin(t)+cos(t), y(0)=0
deff("[ydot]=f(t,y)","ydot=y^2-y*sin(t)+cos(t)")
y0=0;t0=0;t=0:0.1:%pi;
y=ode(y0,t0,t,f)
plot(t,y)
// 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
deff("[xdot]=linear(t,x,A,u)","xdot=A*x+B*u(t)")
```

```
deff("[ut]=u(t)","ut=sin(omega*t)")
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 equation
// Xdot=A'*X + X*A - X'*B*X + C , X(0)=Identity
// Solution at t=[1,2]
deff("[Xdot]=ric(t,X)","Xdot=A''*X+X*A-X''*B*X+C")
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)
//
// Computation of exp(A)
A=[1,1;0,2];
deff("[xdot]=f(t,x)","xdot=A*x");
ode(eye(A),0,1,f)
ode("adams",eye(A),0,1,f)
// with stiff matrix, Jacobian given
A=[10,0;0,-1];
deff("[xdot]=f(t,x)","xdot=A*x");
deff("[J]=Jacobian(t,y)","J=A")
ode("stiff",[0;1],0,1,f,Jacobian)
```

SEE ALSO : `ode_discrete` 350, `ode_root` 354, `dassl` 333, `impl` 338, `odedc` 348, `odeoptions` 353, `csim` 265, `ltitr` 282, `rtitr` 293

6.0.614 odeoptions ______ set options for ode solvers

CALLING SEQUENCE :

```
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` 350

6.0.615 ode_root ______ ordinary differential equation solver with root finding

CALLING SEQUENCE :

```
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`.

CAUTION: at least 2 parameters `[y,rd]` MUST be given as the LHS of this call of `ode`.

`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.

EXAMPLE :

```
// 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)
```

SEE ALSO : dasrt 331, ode 350

6.0.616 optim _____ non-linear optimization routine

CALLING SEQUENCE :

```
[f,xopt]=optim(costf,x0)
[f,[xopt,[gradopt,[work]]]]=optim(costf,[contr],x0,['algo'],[df0,[mem]],
     [work],[stop],['in'],[imp=iflag])
```

PARAMETERS :

`costf` : external, i.e Scilab function or string (`costf` is the cost function: see below its calling sequence (Scilab or Fortran)).
`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` : `'b',binf,bsup` with `binf` and `bsup` real vectors with same dimension as `x0`. `binf` and `bsup` are lower and upper bounds on `x`.
`"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 `costf` 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 `costf` in given as a Fortran routine (see below).
`"imp=iflag"` : named argument used to set the trace mode. `iflag=0` nothing (execpt errors) is reported, `iflag=1` initial and final reports, `iflag=2` adds a report per iteration, `iflag>2` add reports on linear search. Warning, most of these reports are written on the Scilab standard output.
`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 function, or list or Fortran routine (see "external"). This external must return `f` (`costf(x)`) and `g` (gradient of `costf`) given `x`.
If `costf` is a 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 used by `optim` and is described below.
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.
If `costf` is a character string, it refers to the name of a Fortran routine which must be linked to Scilab (see examples in the routines `foptim.f` and e.g. `genros.f` in the directory SCIDIR/default)
Dynamic link of Fortran routine is also possible (help `link`).
Here, the generic calling sequence for the Fortran subroutine is: `function costf(ind,n,x,f,g,ti,tr,td)`
`ind` has the same meaning as above if set to `0,1,2` but the values `ind=10` and `ind=11` are now valid. These values are used for initializations (see below).
`n` is the dimension of `x`, `x` is an `n` vector, `ti,tr,td` are working arrays.
The Fortran function `costf` must return `f` and the vector `g`, given `x, ind, n, ti, tr, td`.
If `costf` is given as a Fortran routine, it is possible to initialize parameters or to send Scilab variables to this routine.
This facility is managed by the parameter `'in`.
If the string `'in'` is present, initialization is done by Fortran: `optim` makes two calls to the Fortran function `costf`, once with `ind=10` and once with `ind=11`. In this case, for `ind=10`, `costf` must set the dimensions `nti, ntr, ntd` of `ti, tr, td` in the `common/nird/nti, ntr, ntd` and, for `ind=11`, `costf` must initialize the vectors `ti , tr, td` (integer vector, real vector, double precision vector respectively).
In the calling sequence of `optim`, the string `'in'` can be replaced by `'ti', valti ,'td' , valtd`. Then, the Fortran function `costf(ind, x, f, g, ti, tr, td)` is evaluated with `ti=valti` and `td=valtd` whatever the value of `ind`. Thus, the Scilab variables `valti` and `valtd` (integer vector and real vector) are sent to the Fortran function `costf`.
It is also possible to save the content of of the working arrays `ti` and `td`. This is possible by adding the strings 'si' and/or 'sd' at the ned of the calling sequence of `optim`. Then, the output variables must be: `[f,[x,[g],[to]]],[ti],[td]]`.

EXAMPLES :

```
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');
[f,xopt]=optim(cost,x0)      //Simplest call
[f,xopt,gopt]=optim(cost,x0,'gc')  // By conjugate gradient
[f,xopt,gopt]=optim(cost,x0,'nd')  //Seen as non differentiable
[f,xopt,gopt]=optim(cost,'b',[-1;0;2],[0.5;1;4],x0) //  Bounds on x
[f,xopt,gopt]=optim(cost,'b',[-1;0;2],[0.5;1;4],x0,'gc') //  Bounds on x
[f,xopt,gopt]=optim(cost,'b',[-1;0;2],[0.5;1;4],x0,'gc','ar',3)
// Here, 3 calls to cost are allowed.
// Now calling the Fortran subroutine "genros" in SCIDIR/default/Ex-optim.f
// See also the link function for dynamically linking an objective function
[f,xopt,gopt]=optim('genros',[1;2;3])    //Rosenbrock's function
```

SEE ALSO : external 24, quapro 356, linpro 345, datafit 335, leastsq 344

6.0.617 quapro _____ linear quadratic programming solver

CALLING SEQUENCE :

```
[x,lagr,f]=quapro(Q,p,C,b [,x0])
[x,lagr,f]=quapro(Q,p,C,b,ci,cs [,x0])
[x,lagr,f]=quapro(Q,p,C,b,ci,cs,mi [,x0])
[x,lagr,f]=quapro(Q,p,C,b,ci,cs,mi,x0 [,imp])
```

PARAMETERS :

`Q` : real symmetric matrix (dimension `n x n`).
`p` : real (column) vector (dimension `n`)
`C` : real matrix (dimension `(mi + md) x n`) (If no constraints are given, you can set `C = []`)
`b` : RHS vector (dimension `1 x (mi + md)`)
`ci` : (column) vector of lower-bounds (dimension `1 x 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) = -(% eps)^(-1)`.
`cs` : (column) vector of upper-bounds. (Same remarks as above).
`mi` : number of equality constraints (i.e. `C(1:mi,:)*x = b(1:mi)`)
`x0` : either an initial guess for `x` or one of the character strings `'v'` or `'g'`. If `x0='v'` the calculated initial feasible point is a vertex. If `x0='g'` the calculated initial feasible point is arbitrary.
`imp` : verbose option (optional parameter) (Try `imp=7,8,...`).
`x` : optimal solution found.
`f` : optimal value of the cost function (i.e. `f=p'*x`).
`lagr` : vector of Lagrange multipliers. If lower and upper-bounds `ci,cs` are provided, `lagr` has `n + mi + md` components and `lagr(1:n)` is the Lagrange vector associated with the bound constraints and `lagr (n+1 : n + mi + 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 `mi + md` components.

DESCRIPTION :

`[x,lagr,f]=quapro(Q,p,C,b [,x0])`
Minimize `0.5*x'*Q*x + p'*x`
under the constraint

```
C*x <= b
```

`[x,lagr,f]=quapro(Q,p,C,b,ci,cs [,x0])`
Minimize `0.5*x'*Q*x + p'*x`
under the constraints

```
C*x <= b          ci <= x <= cs
```

`[x,lagr,f]=quapro(Q,p,C,b,ci,cs,mi [,x0])`
Minimize `0.5*x'*Q*x + p'*x`
under the constraints

```
 C(j,:) x = b(j),  j=1,...,mi
 C(j,:) x <= b(j), j=mi+1,...,mi+md
 ci <= x <= cs
```

If no initial point is given the program computes a feasible initial point which is a vertex of the region of feasible points if `x0='v'`.
If `x0='g'`, the program computes a feasible initial point which is not necessarily a vertex. This mode is advisable when the quadratic form is positive definite and there are few constraints in the problem or when there are large bounds on the variables that are just security bounds and very likely not active at the optimal solution.
Note that `Q` is not necessarily non-negative, i.e. `Q` may have negative eigenvalues.

EXAMPLE :


```
//Find x in R^6 such that:
//C1*x = b1 (3 equality constraints i.e mi=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] ;  //
mi=3;
[x,lagr,f]=quapro(Q,p,C,b,ci,cs,mi)
//Only linear constraints (1 to 4) are active (lagr(1:6)=0):
[x,lagr,f]=quapro(Q,p,C,b,[],[],mi)   //Same result as above
```

SEE ALSO : `linpro` 345

AUTHOR : E. Casas, C. Pola Mendez

6.0.618 semidef _____ semidefinite programming

CALLING SEQUENCE :

```
[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:

```
   minimize    c'*x
   subject to  F_0 + x_1*F_1 + ... + x_m*F_m  >= 0

and its dual

   maximize    -Tr F_0 Z
   subject to  Tr F_i Z = c_i, i=1,...,m
               Z >= 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

```
    [ pack(F_0^1)  pack(F_1^1) ...  pack(F_m^1) ]
    [ pack(F_0^2)  pack(F_1^2) ...  pack(F_m^2) ]
F=  [   ...        ...         ...              ]
    [ pack(F_0^L)  pack(F_1^L) ...  pack(F_m^L) ]
```

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,` (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 `-Tr 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.
`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
```

EXAMPLE :

```
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)
```

Chapter 7

Signal Processing toolbox

7.0.619 analpf — create analog low-pass filter

CALLING SEQUENCE :

```
[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')`

EXAMPLE :

```
//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)
```

AUTHOR : C. B.

7.0.620 %asn — elliptic integral

CALLING SEQUENCE :

```
[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

$$K = \int_0^x \frac{dt}{[(1-t^2)(1-mt^2)]^{1/2}}$$

If `x` is a vector, `y` is a vector of same dimension as `x`.

EXAMPLE :

```
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!
```

AUTHOR : F. D.

7.0.621 buttmag ______ response of Butterworth filter

CALLING SEQUENCE :

```
[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

EXAMPLE :

```
//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])
```

AUTHOR : F. D.

7.0.622 casc ______ cascade realization of filter from coefficients

CALLING SEQUENCE :

```
[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).

EXAMPLE :

```
x=[1,2,3;4,5,6;7,8,9;10,11,12]
cels=casc(x,'z')
```


7.0.623 cepstrum ______ cepstrum calculation

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
w=0.1:0.1:5;mag=1+abs(sin(w));
fresp=cepstrum(w,mag);
plot2d([w',w'],[mag(:),abs(fresp)])
```

SEE ALSO : `frfit` 375

7.0.624 cheb1mag ______ response of Chebyshev type 1 filter

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
//Chebyshev; ripple in the passband
n=13;epsilon=0.2;omegac=3;sample=0:0.05:10;
h=cheb1mag(n,omegac,epsilon,sample);
plot(sample,h,'frequencies','magnitude')
```

SEE ALSO : `buttmag` 363

7.0.625 cheb2mag — response of type 2 Chebyshev filter

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
//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 364

7.0.626 chepol — Chebychev polynomial

CALLING SEQUENCE :

```
[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)`.

EXAMPLE :

```
chepol(4,'x')
```

AUTHOR : F. D.

7.0.627 convol — convolution

CALLING SEQUENCE :

```
[y]=convol(h,x)
[y,e1]=convol(h,x,e0)
```

PARAMETERS :

`x,h` :input sequences (`h` is a "short" sequence, `x` a "long" one)
`e0` : old tail to overlap add (not used in first call)
`y` : output of 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. 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]

EXAMPLE :

```
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 366, fft 373, pspect 388

AUTHOR : F. D , C. Bunks Date 3 Oct. 1988

7.0.628 corr — correlation, covariance

CALLING SEQUENCE :

```
[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

```
            n - m
             ====
             \\                                          1
  cov(m) =    >        (x(k)  - xmean) (y(m+k)      - ymean) * ---
             /                                           n
             ====
             k = 1
```

$$cov(m) = 1/n \sum_{1}^{n-m} (x(k) - E(x))(y(m+k) - E(y))$$

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 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`.

EXAMPLE :

```
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` 373

7.0.629 cspect _____ spectral estimation (periodogram method)

CALLING SEQUENCE :

```
[sm,cwp]=cspect(nlags,ntp,wtype,x,y,wpar)
```

PARAMETERS :

`x` : data if vector, amount of input data if scalar
`y` : data if vector, amount of input data if scalar
`nlags` : number of correlation lags (positive integer)

`ntp` : number of transform points (positive integer)
`wtype` : string : `'re','tr','hm','hn','kr','ch'` (window type)
`wpar` : optional window parameters for `wtype='kr', wpar>0` and for `wtype='ch', 0 < wpar(1) < .5, wpar(2) > 0`
`sm` : power spectral estimate in the interval [0,1]
`cwp` : calculated value of unspecified Chebyshev window parameter

DESCRIPTION :
Spectral estimation using the modified periodogram method. Cross-spectral estimate of `x` and `y` is calculated when both `x` and `y` are given. Auto-spectral estimate of `x` is calculated if `y` is not given.

EXAMPLE :

```
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` 388

AUTHOR : C. Bunks

7.0.630 czt ______ chirp z-transform algorithm

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
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
```

AUTHOR : C. Bunks

7.0.631 dft ______ discrete Fourier transform

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
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` 373

AUTHOR : C. B.

7.0.632 ell1mag ______ magnitude of elliptic filter

CALLING SEQUENCE :

```
[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)`.

EXAMPLE :

```
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` 363

7.0.633 eqfir ______ minimax approximation of FIR filter

CALLING SEQUENCE :

```
[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

EXAMPLE :

```
hn=eqfir(33,[0 .2;.25 .35;.4 .5],[0 1 0],[1 1 1]);
[hm,fr]=frmag(hn,256);
plot(fr,hm),
```

AUTHOR : C. B.

7.0.634 eqiir ______ Design of iir filters

CALLING SEQUENCE :

```
[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 interface with eqiir (syredi)
The filter obtained is `h(z)=fact`*product of the elements of `cells`.
That is `hz=fact*prod(cells(2))./prod(cells(3))`

EXAMPLE :

```
[cells,fact,zzeros,zpoles]=...
eqiir('lp','ellip',[2*%pi/10,4*%pi/10],0.02,0.001)
transfer=fact*poly(zzeros,'z')/poly(zpoles,'z')
```

SEE ALSO : `eqfir` 371, `iir` 379

7.0.635 faurre ______ filter

CALLING SEQUENCE :

```
[Pn,Rt,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')
`Pn` : solution of the Riccati equation after `n` iterations.
`Rt, T` : gain matrix of the filter.

DESCRIPTION :
function which computes iteratively the minimal solution of the algebraic Riccati equation and gives the matrices `Rt` and `Tt` of the filter model.

AUTHOR : G. Le V.

7.0.636 ffilt ______ coefficients of FIR low-pass

CALLING SEQUENCE :

```
[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`

AUTHOR : C. B.

7.0.637 fft _____ fast Fourier transform.

CALLING SEQUENCE :

```
[x]=fft(a,-1)
[x]=fft(a,1)
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.
`dim` : integer
`incr` : integer

DESCRIPTION :
Short syntax (one or two dimensional fft):
`x=fft(a,-1)` gives a direct transform (the `-1` refers to the sign of the exponent..., NOT to "inverse"), that is

$$x(k) = \sum_{m=1}^{n} a(m)exp(-2i\pi(m-1)(k-1)/n)$$

for k varying from 1 to n (n=size of vector `a`).
`a=fft(x,1)` performs the inverse transform normalized by `1/n`.
(`fft(fft(.,-1),1)` is identity).
When the first argument given to `fft` is a matrix a two-dimensional FFT is performed.
Long syntax (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 `x1,x2,..,xp` i.e. `a(x1,x2,..,xp)` where `x1` lies in `1..dim1, x2` in `1.. dim2,...` one gets a p-dimensional FFT p by calling p times `fft` as follows

```
a1=fft(a ,-1,dim1,incr1)
a2=fft(a1,-1,dim2,incr2) ...
```

where `dimi` is the dimension of the current variable w.r.t which one is integrating and incri is the increment which separates two successive `xi` elements in `a`.
In particular,if `a` is an nxm matrix, `x=fft(a,-1)` is equivalent to the two instructions:

`a1=fft(a,-1,m,1)` and `x=fft(a1,-1,n,m)`.
if a is an hypermatrix (see hypermat) fft(a,flag) performs the N dimensional fft of a.

EXAMPLE :

```
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))
```

SEE ALSO : corr 366

7.0.638 filter _____ modelling filter

CALLING SEQUENCE :

```
[y,xt]=filter(n,F,H,Rt,T)
```

PARAMETERS :

`n` : number of computed points.
`F, H` : relevant matrices of the Markovian model.
`Rt, T` : gain matrices.
`y` : output of the filter.
`xt` : filter process.

DESCRIPTION :
This function computes the modelling filter

SEE ALSO : faurre 372

AUTHOR : G. Le V.

7.0.639 find_freq _____ parameter compatibility for elliptic filter design

CALLING SEQUENCE :

```
[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 381

AUTHOR : F. D.

7.0.640 findm _______________ for elliptic filter design

CALLING SEQUENCE :

```
[m]=findm(chi)
```

DESCRIPTION :

Search for `m` such that `chi = %k(1-m)/%k(m)` (For use with `find_freq`).

SEE ALSO : `%k` 381

AUTHOR : F. D.

7.0.641 frfit _______________ frequency response fit

CALLING SEQUENCE :

```
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)`.

EXAMPLE :

```
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` 273, `factors` 412, `cepstrum` 364, `mrfit` 386, `freq` 274, `calfrq` 260

7.0.642 frmag _______________ magnitude of FIR and IIR filters

CALLING SEQUENCE :

```
[xm,fr]=frmag(num[,den],npts)
```

PARAMETERS :

`npts` : integer (number of points in frequency response)
`xm` : mvector of magnitude of frequency response at the points `fr`
`fr` : points in the frequency domain where magnitude is evaluated
`num` : if `den` is omitted vector coefficients/polynomial/rational polynomial of filter
`num` : if `den` is given vector coefficients/polynomial of filter numerator
`den` : vector coefficients/polynomial of filter denominator

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 rational polynomial.

AUTHOR : C. B.

7.0.643 fsfirlin _______________ design of FIR, linear phase filters, frequency sampling technique

CALLING SEQUENCE :

```
[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

AUTHOR : G. Le Vey

EXAMPLE :

```
//
//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` 372, `wfir` 396

7.0.644 group _______ group delay for digital filter

CALLING SEQUENCE :

```
[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)`

EXAMPLE :

```
z=poly(0,'z');
h=z/(z-.5);
[tg,fr]=group(100,h);
plot(fr,tg)
```

AUTHOR : C. B.

7.0.645 hank _______ covariance to hankel matrix

CALLING SEQUENCE :

```
[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

AUTHOR : G. Le Vey

EXAMPLE :


```
//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` 244

7.0.646 hilb _______ Hilbert transform

CALLING SEQUENCE :

```
[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 the Hilbert transform centred around the origin.
That is, `xh=(2/(n*pi))*(sin(n*pi/2))^2`.

EXAMPLE :

```
plot(hilb(51))
```

AUTHOR : C. B.

7.0.647 iirgroup _______ group delay Lp IIR filter optimization

CALLING SEQUENCE :

```
[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).

7.0.648 iirlp — Lp IIR filter optimization

CALLING SEQUENCE :

```
[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).

7.0.649 iir — iir digital filter

CALLING SEQUENCE :

```
[hz]=iir(n,ftype,fdesign,frq,delta)
```

PARAMETERS :

`n` : filter order (pos. integer)
`ftype` : string specifying the filter type `'lp','hp','bp','sb'`
`fdesign` : string specifying the analog filter design `='butt','cheb1','cheb2','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.

EXAMPLE :

```
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 ...
hzd=horner(hz,1/q) //delay operator q=z^-1
```

SEE ALSO : `eqfir` 371, `eqiir` 371

AUTHOR : C. B.

7.0.650 intdec — Changes sampling rate of a signal

CALLING SEQUENCE :

```
[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

AUTHOR : C. B.

7.0.651 jmat — row or column block permutation

CALLING SEQUENCE :

```
[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

7.0.652 kalm — Kalman update

CALLING SEQUENCE :

```
[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

AUTHOR : C. B.

7.0.653 %k — Jacobi's complete elliptic integral

CALLING SEQUENCE :

```
[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 :

$$K = \int_0^1 \frac{dt}{[(1-t^2)(1-mt^2)]^{1/2}}$$

EXAMPLE :

```
m=0.4;
%asn(1,m)
%k(m)
```

REFERENCES :
Abramowitz and Stegun page 598

SEE ALSO : `%asn` 362

AUTHOR : F.D.

7.0.654 lattn — recursive solution of normal equations

CALLING SEQUENCE :

```
[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

$$cov = \begin{bmatrix} R_0 \\ R_1 \\ R_2 \\ \vdots \\ R_{nlag} \end{bmatrix}$$

`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

$$\begin{pmatrix} I & -A_1 & -A_2 & \dots & -A_n \end{pmatrix} \begin{pmatrix} R_{p+1} & R_{p+2} & \dots & R_{p+n} \\ R_p & R_{p+1} & \dots & R_{p+n-1} \\ \vdots & \vdots & \dots & \vdots \\ R_{p+1-n} & R_{p+2-n} & \dots & R_p \end{pmatrix} = 0$$

where {`Rk;k=1,nlag`} is the sequence of empirical covariances

AUTHOR : G. Le V.

7.0.655 lattp — lattp

CALLING SEQUENCE :

```
[la,lb]=lattp(n,p,cov)
```

DESCRIPTION :
see lattn

AUTHOR : G.Levey

7.0.656 levin — Toeplitz system solver by Levinson algorithm (multidimensional)

CALLING SEQUENCE :

```
[la,sig]=levin(n,cov)
```

PARAMETERS :

`n` : maximum order of the filter
`cov` : matrix containing the R_k (d x d matrices for a d-dimensional process). It must be given the following way :

$$\begin{pmatrix} R_0 \\ R_1 \\ R_2 \\ \vdots \\ R_{nlag} \end{pmatrix}$$

`la` : list-type variable, giving the successively calculated Levinson polynomials (degree 1 to n), with coefficients `Ak` `sig` : list-type variable, giving the successive mean-square errors.

DESCRIPTION :
function which solves recursively on n the following Toeplitz system (normal equations)

$$\begin{pmatrix} I & -A_1 \dots -A_n \end{pmatrix} \begin{pmatrix} R_1 & R_2 & \dots & R_n \\ R_0 & R_1 & \dots & R_{n-1} \\ R_{-1} & R_0 & \dots & R_{n-2} \\ \vdots & \vdots & \dots & \vdots \\ R_{2-n} & R_{3-n} & \dots & R_1 \\ R_{1-n} & R_{2-n} & \dots & R_0 \end{pmatrix} = 0$$

where {`Rk;k=1,nlag`} is the sequence of `nlag` empirical covariances

AUTHOR : G. Le Vey

EXAMPLE :

```
//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 help)
//
//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 387

7.0.657 lev Yule-Walker equations (Levinson's algorithm)

CALLING SEQUENCE :

`[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

$$\begin{pmatrix} R_0 & R_1 & \ldots & R_{N-1} \\ R_1 & R_0 & \ldots & R_{N-2} \\ \vdots & \vdots & \ldots & \vdots \\ R_{N-1} & R_{N-2} & \ldots & R_0 \end{pmatrix} \begin{pmatrix} ar_1 \\ ar_2 \\ \vdots \\ ar_{N-1} \end{pmatrix} = \begin{pmatrix} \sigma_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

where $R_i = r(i-1)$. using Levinson's algorithm.

AUTHOR : C. B.

7.0.658 lgfft ______ utility for fft

CALLING SEQUENCE :

```
[y]=lgfft(x)
```

PARAMETERS :

`x` : real or complex vector

DESCRIPTION :
returns the lowest power of 2 larger than `size(x)` (for FFT use).

7.0.659 lindquist ______ Lindquist's algorithm

CALLING SEQUENCE :

```
[Pn,Rt,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')
`Pn` : solution of the Riccati equation after n iterations.
`RtP Tt` : gain matrices of the filter.

DESCRIPTION :
computes iteratively the minimal solution of the algebraic Riccati equation and gives the matrices `Rt` and `Tt` of the filter model, by the Lindquist's algorithm.

AUTHOR : G. Le V.

7.0.660 mese ______ maximum entropy spectral estimation

CALLING SEQUENCE :

```
[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`

AUTHOR : C. B.

7.0.661 mfft ______ multi-dimensional fft

CALLING SEQUENCE :

```
[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]`

AUTHOR : C. B.

7.0.662 mrfit ______ frequency response fit

CALLING SEQUENCE :

```
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)`.

EXAMPLE :

```
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 364, frfit 375, freq 274, calfrq 260

7.0.663 phc _______ Markovian representation

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
//
//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 382

7.0.664 pspect _______ cross-spectral estimate between 2 series

CALLING SEQUENCE :

```
[sm,cwp]=pspect(sec_step,sec_leng,wtype,x,y,wpar)
```

PARAMETERS :

`x` : data if vector, amount of input data if scalar
`y` : data if vector, amount of input data if scalar
`sec_step` : offset of each data window
`sec_leng` : length of each data window
`wtype` : window type `(re,tr,hm,hn,kr,ch)`
`wpar` : optional parameters for `wtype='kr'`, `wpar>0` for `wtype='ch'`, `0<wpar(1)<.5`, `wpar(2)>0`
`sm` : power spectral estimate in the interval `[0,1]`

`cwp` : unspecified Chebyshev window parameter

DESCRIPTION :
Cross-spectral estimate between `x` and `y` if both are given and auto-spectral estimate of `x` otherwise. Spectral estimate obtained using the modified periodogram method.

EXAMPLE :

```
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));
//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` 368

AUTHOR : C. B.

7.0.665 remezb ____ Minimax approximation of magnitude response

CALLING SEQUENCE :

```
[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;
```

EXAMPLE :

```
// 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));
```

AUTHOR : C. B.

SEE ALSO : `eqfir` 371

7.0.666 remez ____ Remez's algorithm

CALLING SEQUENCE :

```
[an]=remez(nc,fg,ds,wt)
```

PARAMETERS :

`nc` : integer, number of cosine functions
`fg,ds,wt` : real vectors
`fg` : grid of frequency points in [0,.5)
`ds` : desired magnitude on grid `fg`
`wt` : weighting function on error on grid `fg`

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 `remez` by using the following commands:

```
hn(1nc-1)=an(nc-12)/2;
hn(nc)=an(1);
hn(nc+12*nc-1)=an(2nc)/2;
```

where `an` = cosine filter coefficients

SEE ALSO : `remezb` 389

7.0.667 rpem ____ RPEM estimation

CALLING SEQUENCE :

```
[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.

7.0.668 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
`optfir` : optimal design of linear phase filters using linear programming
`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

7.0.669 sincd — digital sinc function or Direchlet kernel

CALLING SEQUENCE :

```
[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)`

EXAMPLE :

```
plot(sincd(10,1))
```

AUTHOR : G. Le V.

7.0.670 sinc — samples of sinc function

CALLING SEQUENCE :

```
[x]=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).

EXAMPLE :

```
plot(sinc(100,0.1))
```

SEE ALSO : `sincd` 393

AUTHOR : C. B.

7.0.671 %sn — Jacobi 's elliptic function

CALLING SEQUENCE :

```
[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

EXAMPLE :

```
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` 362, `%k` 381

AUTHOR : F. D.

7.0.672 srfaur ______ square-root algorithm

CALLING SEQUENCE :

```
[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.

7.0.673 srkf ______ square root Kalman filter

CALLING SEQUENCE :

```
[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

AUTHOR : C. B.

7.0.674 sskf ______ steady-state Kalman filter

CALLING SEQUENCE :

```
[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

AUTHOR : C. B.

7.0.675 system ______ observation update

CALLING SEQUENCE :

```
[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)`.

AUTHOR : C. B.

7.0.676 trans ______ low-pass to other filter transform

CALLING SEQUENCE :

```
hzt=trans(pd,zd,gd,tr_type,frq)
```

PARAMETERS :

`hz` : input polynomial
`tr_type` : type of transformation
`frq` : frequency values
`hzt` : output polynomial

DESCRIPTION :
function for transforming standardized low-pass filter into one of the following filters: low-pass, high-pass, band-pass, stop-band.

AUTHOR : C. Bunks

7.0.677 wfir ______ linear-phase FIR filters

CALLING SEQUENCE :

```
[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.

AUTHOR : C. Bunks

7.0.678 wiener ______ Wiener estimate

CALLING SEQUENCE :

```
[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

AUTHOR : C. B.

7.0.679 wigner ______ 'time-frequency' wigner spectrum

CALLING SEQUENCE :

```
[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.

7.0.680 window ______ symmetric window

CALLING SEQUENCE :

```
[win_l,cwp]=window(wtype,n,par)
```

PARAMETERS :

`wtype` : window type (`re,tr,hn,hm,kr,ch`)
`n` : window length
`par` : parameter 2-vector (kaiser window: `par(1)=beta>0`) (Chebychev window `par =[dp,df]`), `dp` =main lobe width (`0<dp<.5`), `df`=side lobe height (`df>0`)
`win` : window
`cwp` : unspecified Chebyshev window parameter

DESCRIPTION :

function which calculates symmetric window

AUTHOR : C. B.

7.0.681 yulewalk ______ least-square filter design

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
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)')
```

7.0.682 zpbutt — Butterworth analog filter

CALLING SEQUENCE :

```
[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'))`

AUTHOR : F.D.

7.0.683 zpch1 — Chebyshev analog filter

CALLING SEQUENCE :

```
[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')
```

AUTHOR : F.D.

7.0.684 zpch2 — Chebyshev analog filter

CALLING SEQUENCE :

```
[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')
```

AUTHOR : F.D.

7.0.685 zpell — lowpass elliptic filter

CALLING SEQUENCE :

```
[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` 370, `eqiir` 371

AUTHOR : F.D.

7.0.686 arma2p _ extract polynomial matrices from ar representation

CALLING SEQUENCE :

```
[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.

$$A(z^{-1})y = B(z^{-1})u + D(z^{-1})sig * e(t)$$

EXAMPLE :

```
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` 402, `armax` 403, `armax1` 403, `arsimul` 404, `armac` 401

7.0.687 armac _ Scilab description of an armax process

CALLING SEQUENCE :

```
[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

$$A(z^{-1})y = B(z^{-1})u + D(z^{-1})sig * e(t)$$

`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')` .

EXAMPLE :


```
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` 402, `armax` 403, `armax1` 403, `arsimul` 404, `arma2p` 401, `tlist`. ??

7.0.688 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.
`arspec` : Spectral power estimation of armax processes. Test of mese and arsimul
`exar1` : An Example of ARMAX identification (K.J. Astrom) The armax process is described by : a=[1,-2.851,2.717,-0.865] b=[0,1,1,1] d=[1,0.7,0.2]
`exar2` : ARMAX example (K.J. Astrom). A simulation of a bi dimensional version of the example of exar1.
`exar3` : Spectral power estimation of arma processes from Sawaragi et all where a value of m=18 is used. Test of mese and arsimul
`gbruit` : noise generation
`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

AUTHOR : J.P.C

7.0.689 armax1 armax identification

CALLING SEQUENCE :

```
[a,b,d,sig,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
`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

AUTHOR : J.-Ph.C

7.0.690 armax armax identification

CALLING SEQUENCE :

```
[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))).

EXAMPLE :

```
//-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)
```

AUTHOR : J-Ph. Chancelier.

SEE ALSO : imrep2ss 277, time_id 302, arl2 257, armax 403, frep2tf 273

7.0.691 arsimul armax simulation

CALLING SEQUENCE :

```
[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

AUTHOR : J-Ph.C.

7.0.692 narsimul ________ armax simulation (using rtitr)

CALLING SEQUENCE :

```
[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

AUTHOR : J-Ph. Chancelier ENPC Cergrene

7.0.693 noisegen ________ noise generation

CALLING SEQUENCE :

```
[]=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`.

EXAMPLE :

```
noisegen(0.5,30,1.0);
x=-5:0.01:35;
y=feval(x,Noise);
plot(x,y);
```


7.0.694 odedi ________ test of ode

CALLING SEQUENCE :

```
[]=odedi()
```

DESCRIPTION :
Simple tests of ode and arsimul. Tests the option 'discret' of ode

7.0.695 prbs_a ________ pseudo random binary sequences generation

CALLING SEQUENCE :

```
[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]`.

EXAMPLE :

```
u=prbs_a(50,10);
plot2d2("onn",(1:50)',u',1,"151",' ',[0,-1.5,50,1.5]);
```

7.0.696 reglin ________ Linear regression

CALLING SEQUENCE :

```
[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)`, so the estimator` a `is a matrix of size (q,p)` and b `is a vector of size` (q,1)

EXAMPLE :

```
// 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)
```

Chapter 8

Polynomial calculations

8.0.697 bezout Bezout equation for polynomials

CALLING SEQUENCE :

```
[thegcd,U]=bezout(p1,p2)
```

PARAMETERS :

p1, p2 : two real polynomials

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))

EXAMPLE :

```
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])
```

SEE ALSO : poly 81, roots 420, simp 421, clean 408, lcm 416

8.0.698 clean cleans matrices (round to zero small entries)

CALLING SEQUENCE :

```
[B]=clean(A [,epsa [,epsr]])
```

PARAMETERS :

A : a numerical matrix (scalar, polynomial, sparse...)
epsa,epsr : real numbers (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 and 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.

EXAMPLE :

```
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))
```

8.0.699 cmndred _______ common denominator form

CALLING SEQUENCE :

```
[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` 421, `clean` 408

8.0.700 coffg _______ inverse of polynomial matrix

CALLING SEQUENCE :

```
[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)

EXAMPLE :

```
s=poly(0,'s')
a=[ s, s^2+1; s  s^2-1];
[a1,d]=coffg(a);
(a1/d)-inv(a)
```

SEE ALSO : `determ` 411, `detr` 411, `invr` 415, `penlaur` 448, `glever` 435

AUTHOR : F. D.

8.0.701 colcompr _______ column compression of polynomial matrix

CALLING SEQUENCE :

```
[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)

EXAMPLE :

```
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` 420

8.0.702 denom _______ denominator

CALLING SEQUENCE :

```
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)` or `r('den')` .

SEE ALSO : `numer` 417

8.0.703 derivat _______ rational matrix derivative

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
s=poly(0,'s');
derivat(1/s)  // -1/s^2;
```

8.0.704 determ ________ determinant of polynomial matrix

CALLING SEQUENCE :

```
res=determ(W [,k])
```

PARAMETERS :

`W` : real square polynomial matrix
`k` : integer (upper bound for the degree of the determinant of W)

DESCRIPTION :
`res=determ(W [,k])` returns the determinant of a real polynomial matrix (computation made by FFT).
`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: evaluate the determinant of `W` for the Fourier frequencies and apply inverse FFT to the coefficients of the determinant.

EXAMPLE :

```
s=poly(0,'s');
w=s*rand(10,10);
determ(w)
det(coeff(w,1))*s^10
```

SEE ALSO : `det` 431, `detr` 411, `coffg` 409

AUTHOR : F.D.

8.0.705 detr ________ polynomial determinant

CALLING SEQUENCE :

```
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` 431, `determ` 411

8.0.706 diophant ________ diophantine (Bezout) equation

CALLING SEQUENCE :

```
[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 $err = ||p1x1 + p2x2 - b||/||b||$ else `err=0`

EXAMPLE :

```
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)
```

8.0.707 factors ________ numeric real factorization

CALLING SEQUENCE :

```
[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'`).

EXAMPLE :

```
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` 421

8.0.708 gcd — gcd calculation

CALLING SEQUENCE :

```
[pgcd,U]=gcd(p)
```

PARAMETERS :

p : polynomial row vector p=[p1,..,pn]

DESCRIPTION :

[pgcd,u]=gcd(p) 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]

EXAMPLE :

```
s=poly(0,'s');
p=[s,s*(s+1)^2,2*s^2+s^3];
[pgcd,u]=gcd(p);
p*u
```

SEE ALSO : bezout 408, lcm 416, hermit 413

8.0.709 hermit — Hermite form

CALLING SEQUENCE :

```
[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

EXAMPLE :

```
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 414, htrianr 415

8.0.710 horner — polynomial/rational evaluation

CALLING SEQUENCE :

```
horner(P,x)
```

PARAMETERS :

P : polynomial or rational matrix
x : real number or polynomial or rational

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 :

```
s=poly(0,'s');M=[s,1/s];
horner(M,1)
horner(M,%i)
horner(M,1/s)
```

SEE ALSO : freq 274, repfreq 291, evstr 21

8.0.711 hrmt — gcd of polynomials

CALLING SEQUENCE :

```
[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].

EXAMPLE :

```
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 413, htrianr 415

8.0.712 htrianr ________ triangularization of polynomial matrix

CALLING SEQUENCE :

```
[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).

EXAMPLE :

```
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` 414, `colcompr` 409

8.0.713 invr ________ inversion of (rational) matrix

CALLING SEQUENCE :

```
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)

EXAMPLE :

```
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` 435, `coffg` 409, `inv` 439

8.0.714 lcmdiag ________ least common multiple diagonal factorization

CALLING SEQUENCE :

```
[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')

EXAMPLE :

```
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` 416, `gcd` 413, `bezout` 408

8.0.715 lcm ________ least common multiple

CALLING SEQUENCE :

```
[pp,fact]=lcm(p)
```

PARAMETERS :

`p` :
`fact` : polynomial vector
`pp` : polynomial

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)`

EXAMPLE :

```
s=poly(0,'s');
p=[s,s*(s+1)^2,s^2*(s+2)];
[pp,fact]=lcm(p);
p.*fact, pp
```

SEE ALSO : `gcd` 413, `bezout` 408

8.0.716 ldiv _____ polynomial matrix long division

CALLING SEQUENCE :

```
[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`

EXAMPLE :

```
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` 257, `markp2ss` 283, `pdiv` 417

8.0.717 numer _____ numerator

CALLING SEQUENCE :

```
NUM=numer(R)
```

PARAMETERS :

`R` : rational 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)` or `R('num')`

SEE ALSO : `denom` 410

8.0.718 pdiv _____ polynomial division

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
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` 417, `gcd` 413

8.0.719 pol2des _____ polynomial matrix to descriptor form

CALLING SEQUENCE :

```
[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`

EXAMPLE :

```
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` 297, `tf2des` 326

AUTHOR : F.D.

8.0.720 pol2str _____ polynomial to string conversion

CALLING SEQUENCE :

```
[str]=pol2str(p)
```

PARAMETERS :

`p` : real polynomial
`str` : character string

DESCRIPTION :
converts polynomial to character string (utility function).

SEE ALSO : `string` 100, `pol2tex` 687

8.0.721 polfact _______ minimal factors

CALLING SEQUENCE :

```
[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` 416, `cmndred` 409, `factors` 412

8.0.722 residu _______ residue

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
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` 288, `bdiag` 427, `roots` 420, `poly` 81, `gtild` 315

AUTHOR : F.D.

8.0.723 roots _______ roots of polynomials

CALLING SEQUENCE :

```
[x]=roots(p)
```

PARAMETERS :

`p` : polynomial with real or complex coefficients

DESCRIPTION :
`x=roots(p)` returns in the complex vector `x` the roots of the polynomial `p`. Degree of `p` must be <=100.

EXAMPLE :

```
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` 81

8.0.724 routh_t _______ Routh's table

CALLING SEQUENCE :

```
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.

8.0.725 rowcompr _______ row compression of polynomial matrix

CALLING SEQUENCE :

```
[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` 409

8.0.726 sfact _____ discrete time spectral factorization

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
//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 315, fspecg 312

8.0.727 simp _____ rational simplification

CALLING SEQUENCE :

```
[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 420, trfmod 244, poly 81, clean 408, simp_mode 422

8.0.728 simp_mode _____ toggle rational simplification

CALLING SEQUENCE :

```
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 421

8.0.729 sylm __________ Sylvester matrix

CALLING SEQUENCE :

```
[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`

8.0.730 systmat __________ system matrix

CALLING SEQUENCE :

```
[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` 297, `sm2des` 295, `sm2ss` 296

Chapter 9

Linear Algebra

9.0.731 aff2ab ________ linear (affine) function to A,b conversion

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
// 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` 442

9.0.732 balanc ________ matrix or pencil balancing

CALLING SEQUENCE :

```
[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=X*E*Y` `Ab=X*A*Y`

REMARK :
Balancing is made in the functions `bdiag` and `spec`.

EXAMPLE :

```
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` 427

9.0.733 bdiag ________ block diagonalization, generalized eigenvectors

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
//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` 456, `sylv` 462, `spec` 460

9.0.734 chfact ________ sparse Cholesky factorization

CALLING SEQUENCE :

```
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` 429, `sparse` 229, `lufact` 443, `luget` 443, `spchol` 460

9.0.735 chol ________ Cholesky factorization

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
W=rand(5,5)+%i*rand(5,5);
X=W*W';
R=chol(X);
norm(R'*R-X)
```

SEE ALSO : spchol 460, qr 451, svd 462, bdiag 427, fullrf 434

9.0.736 chsolve ______ sparse Cholesky solver

CALLING SEQUENCE :

```
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 `sol=A*rhs`, 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`.

EXAMPLE :

```
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 428, sparse 229, lufact 443, luget 443, spchol 460

9.0.737 coff ______ resolvent (cofactor method)

CALLING SEQUENCE :

```
[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)

EXAMPLE :

```
M=[1,2;0,3];
[N,d]=coff(M)
N/d
inv(%s*eye()-M)
```

SEE ALSO : coffg 409, ss2tf 299, nlev 446, poly 81

9.0.738 colcomp ______ column compression, kernel, nullspace

CALLING SEQUENCE :

```
[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)`

EXAMPLE :

```
A=rand(5,2)*rand(2,5);
[X,r]=colcomp(A);
norm(A*X(:,1:$-r),1)
```

SEE ALSO : rowcomp 455, fullrf 434, fullrfk 434, kernel 440

AUTHOR : F.D.

9.0.739 companion ______ companion matrix

CALLING SEQUENCE :

```
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)`.

EXAMPLE :

```
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 460, poly 81, randpencil 453

AUTHOR : F.D.

9.0.740 cond ______ condition number

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
A=testmatrix('hilb',6);
cond(A)
```

SEE ALSO : `rcond` 454, `svd` 462

9.0.741 det ______ determinant

CALLING SEQUENCE :

```
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)`.

EXAMPLE :

```
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` 411, `determ` 411

9.0.742 ereduc ______ computes matrix column echelon form by qz transformations

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
X=[1 2 3;4 5 6]
[E,Q,Z ,stair ,rk]=ereduc(X,1.d-15)
```

SEE ALSO : `fstair` 433

AUTHOR : Th.G.J. Beelen (Philips Glass Eindhoven). SLICOT

9.0.743 exp ______ element-wise exponential

CALLING SEQUENCE :

```
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`.

EXAMPLE :

```
x=[1,2,3+%i];
log(exp(x))  //element-wise
2^x
exp(x*log(2))
```

SEE ALSO : `coff` 429, `log` 214, `expm` 433

9.0.744 expm ______ square matrix exponential

CALLING SEQUENCE :

```
expm(X)
```

PARAMETERS :

`X` : square matrix with real or complex entries.

DESCRIPTION :

`X` is a square matrix `expm(X)` is the matrix

$$exp(X) = I + X + X^2/2 + ...$$

The computation is performed by first block-diagonalizing `X` and then applying a Pade approximation on each block.

EXAMPLE :

```
X=[1 2;3 4]
expm(X)
logm(expm(X))
```

SEE ALSO : `logm` 215, `bdiag` 427, `coff` 429, `log` 214, `exp` 432

9.0.745 fstair ______ computes pencil column echelon form by qz transformations

CALLING SEQUENCE :

```
[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)

AUTHOR : Th.G.J. Beelen (Philips Glass Eindhoven). SLICOT

SEE ALSO : `quaskro` 452, `ereduc` 432

9.0.746 fullrfk ______ full rank factorization of A^k

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
A=rand(5,2)*rand(2,5);[Bk,Ck]=fullrfk(A,3);
norm(Bk*Ck-A^3,1)
```

SEE ALSO : `fullrf` 434, `range` 453

AUTHOR : F.D (1990)

9.0.747 fullrf ______ full rank factorization

CALLING SEQUENCE :

```
[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'`.

EXAMPLE :

```
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` 462, `qr` 451, `fullrfk` 434, `rowcomp` 455, `colcomp` 430

AUTHOR : F.D.

9.0.748 givens _____ Givens transformation

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
A=[3,4;5,6];
U=givens(A(:,1));
U*A
```

SEE ALSO : `qr` 451

9.0.749 glever _____ inverse of matrix pencil

CALLING SEQUENCE :

```
[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 $(sE - A)^{-1}$ 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`.

EXAMPLE :

```
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)
```

AUTHOR : F. D. (1988)

SEE ALSO : `rowshuff` 455, `det` 431, `invr` 415, `coffg` 409, `pencan` 447, `penlaur` 448

9.0.750 gschur _____ generalized Schur form (matrix pencils).

CALLING SEQUENCE :

```
[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)
```

PARAMETERS :

`A, E` : two real square matrices
`flag` : character string (`'c'` or `'d'`)
`extern` : Scilab "external" function (usual case). Could be also a list or a character string
`As,Es` : two real square matrices
`Q, Z` : two non-singular real matrices
`dim` : integer (dimension of subspace)

DESCRIPTION :

Schur form of matrix pencils (QZ algorithm):

`[As,Es] = gschur(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] = gschur(A,E)`

returns in addition two unitary matrices `Q` and `Z` such that `As=Q*A*Z` and `Es=Q*E*Z`.
Ordered stable form:

```
[As,Es,Z,dim] = gschur(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] = gschur(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).
General subspace:

```
[As,Es,Z,dim] = gschur(A,E,extern)
```

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 scilab function extern. (See `schur` for definition of this function).

EXAMPLE :

```
s=%s;
F=[-1,s,0,0;0,-1,0,0;0,0,2+s,0;0,0,0,-2+s];
roots(det(F))
[E,A]=pen2ea(F);
[As,Es,Z,dim] = gschur(A,E,'c')
// Other example
a=rand(4,4);b=rand(4,4);[as,bs,qs,zs]=gschur(a,b);
norm(qs*a*zs-as)
norm(qs*b*zs-bs )
clear a;
a(8,8)=2;a(1,8)=1;a(2,[2,3,4,5])=[0.3,0.2,4,6];a(3,[2,3])=[-0.2,.3];
a(3,7)=.5;
a(4,4)=.5;a(4,6)=2;a(5,5)=1;a(6,6)=4;a(6,7)=2.5;a(7,6)=-10;a(7,7)=4;
b=eye(8,8);b(5,5)=0;
[al,be]=gspec(a,b);
[bs,as,q,n]=gschur(b,a,'disc');n-4
```

SEE ALSO : `external` 24, `gspec` 437, `pencan` 447, `penlaur` 448, `coffg` 409, `kroneck` 441

9.0.751 gspec ________ eigenvalues of matrix pencil

CALLING SEQUENCE :

```
[al,be]=gspec(A,E)
[al,be,Z]=gspec(A,E)
```

PARAMETERS :

`A, E` : real square matrices
`al, be` : real vectors
`Z` : real square non-singular matrix

DESCRIPTION :

`[al,be] = gspec(A,E)` returns the spectrum of the matrix pencil s E - A, i.e. the roots of the polynomial matrix s E - A. The eigenvalues are given by `al./be` and if `be(i) = 0` the ith eigenvalue is at infinity. (For `E = eye(A), al./be` is `spec(A)`).
`[al,be,Z] = gspec(A,E)` returns in addition the matrix `Z` of generalized right eigenvectors of the pencil.

EXAMPLE :

```
A=rand(3,3);
[al,be,Z] = gspec(A,eye(A));al./be
clean(inv(Z)*A*Z)  //displaying the eigenvalues (generic matrix)
A=A+%i*rand(A);E=rand(A);
roots(det(%s*E-A))   //complex case
```

SEE ALSO : `gschur` 436, `balanc` 427, `spec` 460, `kroneck` 441

9.0.752 hess ________ Hessenberg form

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
A=rand(3,3);[U,H]=hess(A);
and( abs(U*H*U'-A)<1.d-10 )
```

SEE ALSO : `qr` 451, `contr` 264, `schur` 456

9.0.753 householder ________ Householder orthogonal reflexion matrix

CALLING SEQUENCE :

```
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` 451, `givens` 435

9.0.754 im_inv ________ inverse image

CALLING SEQUENCE :

```
[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 $A^{-1}(\mathcal{B})$ i.e vectors whose image through `A` are in range(B)
The `dim` first columns of `X` span $A^{-1}(\mathcal{B})$.
`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.

EXAMPLE :

```
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` 455, `spaninter` 458, `spanplus` 458, `linsolve` 442

AUTHOR : F. D.

9.0.755 inv ________ matrix inverse

CALLING SEQUENCE :

```
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)`.

EXAMPLE :

```
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` 97, `backslash` 7, `pinv` 449, `qr` 451, `lufact` 443, `lusolve` 445, `invr` 415, `coff` 429, `coffg` 409

9.0.756 kernel ________ kernel, nullspace

CALLING SEQUENCE :

```
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`.
`flag` and `tol` are optional parameters: `flag = 'qr'` or `'svd'` (default is `'svd'`).
`tol` = tolerance parameter (of order `%eps` as default value).

EXAMPLE :

```
A=rand(3,1)*rand(1,3);
A*kernel(A)
A=sparse(A);
clean(A*kernel(A))
```

SEE ALSO : `colcomp` 430, `fullrf` 434, `fullrfk` 434, `linsolve` 442

AUTHOR : F.D.

9.0.757 kroneck ____________ Kronecker form of matrix pencil

CALLING SEQUENCE :

```
[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).

EXAMPLE :

```
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` 436, `gspec` 437, `systmat` 423, `pencan` 447, `randpencil` 453, `trzeros` 303

9.0.758 linsolve ____________ linear equation solver

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
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` 439, `pinv` 449, `colcomp` 430, `im_inv` 439

9.0.759 ludel ____________ utility function used with lufact

CALLING SEQUENCE :

```
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` 229, `lufact` 443, `luget` 443

9.0.760 lufact ______ sparse lu factorization

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
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` 229, `lusolve` 445, `luget` 443

9.0.761 luget ______ sparse lu factorization

CALLING SEQUENCE :


```
[P,L,U,Q]=luget(ptr)
```

PARAMETERS :

`ptr` : pointer, output of `lufact`
`P` : sparse permutation matrix
`L` : sparse matrix, lower triangular if `ptr` 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(ptr)` with `ptr` obtained by the command `[ptr,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`.

EXAMPLE :

```
a=rand(5,2)*rand(2,5);A=sparse(a);
[ptr,rk]=lufact(A);[P,L,U,Q]=luget(ptr);
full(L), P*L*U*Q-A
clean(P*L*U*Q-A)
ludel(ptr)
```

SEE ALSO : `sparse` 229, `lusolve` 445, `luget` 443, `clean` 408

9.0.762 lu ______ LU factors of Gaussian elimination

CALLING SEQUENCE :

```
[L,U]= lu(A)
[L,U,E]= lu(A)
```

PARAMETERS :

`A` : real or complex square matrix (n x n).
`L,U` : two real or complex matrices (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`.

REMARK :

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`.

```
[h,rk]=lufact(sparse(a))  // lufact works with sparse real matrices
[P,L,U,Q]=luget(h)
ludel(h)
P=full(P);L=full(L);U=full(U);Q=full(Q);
// P,Q are permutation matrices P*L*U*Q=A
```

SEE ALSO : `lufact` 443, `luget` 443, `lusolve` 445, `qr` 451, `svd` 462

9.0.763 lusolve ________ sparse linear system solver

CALLING SEQUENCE :

```
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 \fVA*x = b\fR.`

EXAMPLE :

```
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` 229, `lufact` 443, `slash` 97, `backslash` 7

9.0.764 lyap ________ Lyapunov equation

CALLING SEQUENCE :

```
[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').
```

EXAMPLE :


```
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` 462, `ctr_gram` 265, `obs_gram` 287

9.0.765 nlev ________ Leverrier's algorithm

CALLING SEQUENCE :

```
[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: $(zI - A)^{-1}$ by block diagonalization of A followed by Leverrier's algorithm on each block.

REMARK :
This algorithm is better than the usual leverrier algorithm but still not perfect!

EXAMPLE :

```
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` 429, `coffg` 409, `glever` 435, `ss2tf` 299

AUTHOR : F. D., S. S.

9.0.766 orth ________ orthogonal basis

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
A=rand(5,3)*rand(3,4);
[X,dim]=rowcomp(A);X=X';
svd([orth(A),X(:,1:dim)])
```

SEE ALSO : `qr` 451, `rowcomp` 455, `colcomp` 430, `range` 453

9.0.767 pbig ______ eigen-projection

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
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` 451, `projspec` 450, `fullrf` 434

AUTHOR : F. D. (1988)

9.0.768 pencan ______ canonical form of matrix pencil

CALLING SEQUENCE :

```
[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]
```

EXAMPLE :

```
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` 435, `penlaur` 448, `rowshuff` 455

AUTHOR : F. D.

9.0.769 penlaur ______ Laurent coefficients of matrix pencil

CALLING SEQUENCE :

```
[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`

EXAMPLE :

```
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` 435, `pencan` 447, `rowshuff` 455

AUTHOR : F. D. (1988,1990)

9.0.770 pinv _____ pseudoinverse

CALLING SEQUENCE :

```
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)`.

EXAMPLE :

```
A=rand(5,2)*rand(2,4);
norm(A*pinv(A)*A-A,1)
```

SEE ALSO : rank 454, svd 462, qr 451

9.0.771 polar _____ polar form

CALLING SEQUENCE :

```
[Ro,Theta]=polar(A)
```

PARAMETERS :

A : real or complex square matrix
Ro, Theta : real matrices

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.

EXAMPLE :

```
A=rand(5,5);
[Ro,Theta]=polar(A);
norm(A-Ro*expm(%i*Theta),1)
```

SEE ALSO : expm 433, svd 462

AUTHOR : F. D.

9.0.772 proj _____ projection

CALLING SEQUENCE :

```
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`.

EXAMPLE :

```
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 450, orth 446, fullrf 434

AUTHOR : F. D.

9.0.773 projspec _____ spectral operators

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
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 429

AUTHOR : F. D.

9.0.774 psmall ______ spectral projection

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
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` 447, `proj` 450, `projspec` 450

AUTHOR : F. D. (1988)

9.0.775 qr ______ QR decomposition

CALLING SEQUENCE :

```
[Q,R]=qr(X)
[Q,R,E]=qr(X)
[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*E`)

DESCRIPTION :
`[Q,R] = qr(X)` produces an upper triangular matrix `R` of the same dimension as `X` and a unitary matrix `Q` so that `X = Q*R`.
`[Q,R,E] = qr(X)` produces a (column) permutation matrix `E`, an upper triangular `R` with decreasing diagonal elements and a unitary `Q` so that `X*E = Q*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 `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 `R(1,1)*%eps*max(size(R)`.

EXAMPLE :

```
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 : `rank` 454, `svd` 462, `rowcomp` 455, `colcomp` 430

9.0.776 quaskro ______ quasi-Kronecker form

CALLING SEQUENCE :

```
[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     |
           |----------------|----------------|------------|
           |        O       | sE(inf)-A(inf) |      X     |
Q(sE-A)Z = |=================================|============|
           |                                 |            |
           |                O                | 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` 441, `gschur` 436, `gspec` 437

9.0.777 randpencil ______ random pencil

CALLING SEQUENCE :

```
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`.

EXAMPLE :

```
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` 441, `pencan` 447, `penlaur` 448

9.0.778 range ______ range (span) of A^k

CALLING SEQUENCE :

```
[X,dim]=range(A,k)
```

PARAMETERS :

`A` : real square matrix
`k` : integer
`X` : non-singular real matrix
`dim` : integer (dimension of subspace)

DESCRIPTION :
Computation of Range `A^k` ; the first dim columns of `X` span the range of `A^k`.

SEE ALSO : `fullrfk` 434, `rowcomp` 455

AUTHOR : F. D.

9.0.779 rank ______ rank

CALLING SEQUENCE :

```
[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`.

REMARK :
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 !.

EXAMPLE :

```
rank([1.d-80,0;0,1.d-80])
rank([1,0;0,1.d-80])
```

SEE ALSO : `svd` 462, `qr` 451, `rowcomp` 455, `colcomp` 430, `lu` 444

9.0.780 rcond ______ inverse condition number

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
A=diag([1:10]);
rcond(A)
A(1,1)=0.000001;
rcond(A)
```

SEE ALSO : `svd` 462, `cond` 431, `inv` 439

9.0.781 rowcomp ______________ row compression, range

CALLING SEQUENCE :

```
[W,rk]=rowcomp(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 :
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 (of order `sqrt(%eps)` as default value).
The `rk` first columns of `W'` span the range of `A`.
The `rk` first (top) rows of `W` span the row range of `A`.

REMARK :
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.

EXAMPLE :

```
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` 430, `fullrf` 434, `fullrfk` 434

AUTHOR : F. D.

9.0.782 rowshuff ______________ shuffle algorithm

CALLING SEQUENCE :

```
[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)`

EXAMPLE :

```
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` 447, `glever` 435, `penlaur` 448

AUTHOR : F. D.

9.0.783 rref computes matrix row echelon form by lu transformations

CALLING SEQUENCE :

```
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`

EXAMPLE :

```
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` 444, `qr` 451

9.0.784 schur ______________ [ordered] Schur decomposition

CALLING SEQUENCE :

```
[U,T] = schur(A)
[U,dim]=schur(A,flag)
[U,dim]=schur(A,myfunction)
```

PARAMETERS :

`A` : real or complex matrix. For ordered forms `A` is assumed real.
`flag` : character string (`'c'` or `'d'`)
`myfunction` : an “external” function (this parameter can also be a list or character string)

U : orthogonal or unitary square matrix
T : matrix
dim : integer

DESCRIPTION :
Schur forms, ordered Schur forms
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 STABLE FORM :
[T,dim]=schur(A,'c') returns an unitary matrix T which transforms A into schur form. In addition, the dim first columns of T make a basis of the eigenspace of A associated with eigenvalues with negative real parts (stable "continuous time" eigenspace).
[T,dim]=schur(A,'d') returns an unitary matrix T which transforms A into schur form. In addition, the dim first columns of T span a basis of the eigenspace of A associated with eigenvalues with magnitude lower than 1 (stable "discrete time" eigenspace).
GENERAL EIGENSPACE :
[T,dim]=schur(A,a_function) returns an unitary matrix T which transforms A into schur form. In addition, the dim first columns of T span a basis of the eigenspace of A associated with the eigenvalues which are selected by the function a_function.
This function must be of the following type (here a_function is "rule"):

```
function [flag]=rule(x)

flag=...
```

x is a vector with three components which characterizes either a real eigenvalue or a pair of complex conjugate eigenvalues.
If x(1)=1, a real eigenvalue is considered and this eigenvalue is x(2)/x(3).
If x(1)=2, a pair of complex conjugate eigenvalues is considered. The sum of these two eigenvalues (twice the real part) is x(2) and the product (squared magnitude) is x(3).
On return, flag should be 1 if the real eigenvalue is selected or the pair of eigenvalues is selected and 0 otherwise.
EXAMPLE OF FUNCTION :

```
function [flag]=disc(x)
ls =x(1);flag=0;
select  ls
   case 1 then if abs(x(2)) < ro*abs(x(3)) then flag=1;end
   case 2 then if x(3) < ro*ro then flag=1;end
end
```

The function disc selects the eigenvalues with magnitude lower than a given scalar ro. And for ro=1 the calling sequence [T,dim]=schur(A,'d') and [T,dim]=schur(A,disc) are equivalent.
Another useful example is %choose (see function code in SCIDIR/macros/percent)
EXAMPLE :

```
A=diag([-0.9,-2,2,0.9]);X=rand(A);A=inv(X)*A*X;
[U,d]=schur(A,'c');
A1=U'*A*U;
spec(A1(1:d,1:d))      //stable cont. eigenvalues
[U,d]=schur(A,'c');
A1=U'*A*U;
spec(A1(1:d,1:d))      //stable disc. eigenvalues
```

SEE ALSO : gschur 436, ricc 292, pbig 447, psmall 451

9.0.785 spaninter — subspace intersection

CALLING SEQUENCE :

```
[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 :
[X,dim]=spaninter(A,B) 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 $\mathcal{R}(A) \cap \mathcal{R}(B)$
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).
EXAMPLE :

```
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 458, spantwo 459

AUTHOR : F. D.

9.0.786 spanplus — sum of subspaces

CALLING SEQUENCE :

```
[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 :
[X,dim,dima]=spanplus(A,B) 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).

EXAMPLE :

```
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` 458, `im_inv` 439, `spantwo` 459

AUTHOR : F. D.

9.0.787 spantwo ______ sum and intersection of subspaces

CALLING SEQUENCE :

```
[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)).

EXAMPLE :

```
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` 458, `spaninter` 458

AUTHOR : F. D.

9.0.788 spchol ______ sparse cholesky factorization

CALLING SEQUENCE :

```
[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`.

EXAMPLE :

```
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` 229, `lusolve` 445, `luget` 443, `chol` 428

9.0.789 spec ______ eigenvalues

CALLING SEQUENCE :

```
evals=spec(A)
```

PARAMETERS :

`A` : real or complex square matrix
`evals` : real or complex vector

DESCRIPTION :

`evals=spec(A)` returns in vector `evals` the eigenvalues of `A`.
Eigenvectors are obtained by `bdiag`.

EXAMPLE :

```
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)
```

```
//
[Ab,X,bs]=bdiag(A);
Ab
clean(inv(X)*A*X)
```

SEE ALSO : poly 81, det 431, gspec 437, schur 456, bdiag 427, colcomp 430

9.0.790 sqroot ______ W*W' hermitian factorization

CALLING SEQUENCE :

```
sqroot(X)
```

PARAMETERS :

X : symmetric non negative definite real or complex matrix

DESCRIPTION :

W=sqroot(X) returns W such that X=W*W' (uses SVD).

EXAMPLE :

```
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 428, svd 462

9.0.791 sva ______ singular value approximation

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
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 462

9.0.792 svd ______ singular value decomposition

CALLING SEQUENCE :

```
s=svd(X)
[U,S,V]=svd(X)
[U,S,V]=svd(X,0)
[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.

EXAMPLE :

```
X=rand(4,2)*rand(2,4)
svd(X)
sqrt(spec(X*X'))
```

SEE ALSO : rank 454, qr 451, colcomp 430, rowcomp 455, sva 461, spec 460

9.0.793 sylv ______ Sylvester equation.

CALLING SEQUENCE :

```
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
```

EXAMPLE :

```
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` 445

9.0.794 trace — trace

CALLING SEQUENCE :

```
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))`.

EXAMPLE :

```
A=rand(3,3);
trace(A)-sum(spec(A))
```

SEE ALSO : `det` 431

Chapter 10

Metanet

10.0.795 add_edge adds an edge or an arc between two nodes

CALLING SEQUENCE :

```
g1 = add_edge(i,j,g)
```

PARAMETERS :

`i` : integer, number of start node
`j` : integer, number of end node
`g` : graph list
`g1` : graph list of the new graph with the added edge

DESCRIPTION :
`add_edge` returns the graph `g1` with a new edge from node number `i` to node number `j`. 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.

EXAMPLE :

```
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);
g=add_edge(1,7,g);
g('edge_color')=[ones(ta) 11];
show_graph(g);
```

SEE ALSO : `add_node` 466, `delete_arcs` 476, `delete_nodes` 477

10.0.796 add_node adds a disconnected node to a graph

CALLING SEQUENCE :

```
g1 = add_node(g,[xy,name])
```

PARAMETERS :

`g` : graph list
`xy` : optional row vector of coordinates of new node
`name` : optional name of the added node
`g1` : graph list of the new graph with the added node

DESCRIPTION :
`add_node` adds a disconnected node to graph `g` and returns the new graph `g1`.

The coordinates of the new node can be given as a row 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 name of the new node, otherwise the number of nodes plus 1 is taken as the name of the new node.

EXAMPLE :

```
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);
n=g('node_number');
g1=add_node(g,[270 140]);
g1('node_color')=[ones(1,n) 11];
show_graph(g1);
```

SEE ALSO : add_edge 466, delete_arcs 476, delete_nodes 477

10.0.797 adj_lists ______ computes adjacency lists

CALLING SEQUENCE :

```
[lp,la,ls] = adj_lists(g)
[lp,la,ls] = adj_lists(directed,n,tail,head)
```

PARAMETERS :

`g` : graph list
`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`.

EXAMPLE :

```
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('node_x')=[129 200 283 281 128 366 122 333];
g('node_y')=[61 125 129 189 173 135 236 249];
show_graph(g);
[lp,la,ls]=adj_lists(g)
[lp,la,ls]=adj_lists(1,g('node_number'),ta,he)
```

SEE ALSO : chain_struct 471, graph_2_mat 480

10.0.798 arc_graph ______ graph with nodes corresponding to arcs

CALLING SEQUENCE :

```
g1 = arc_graph(g)
```

PARAMETERS :

`g` : graph list of the old graph
`g1` : graph list of the new 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.

EXAMPLE :

```
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('node_x')=[281  284  360  185  405  182  118  45];
g('node_y')=[262  179  130  154  368  248  64  309];
show_graph(g);
g1=arc_graph(g);
show_graph(g1,'new');
```

SEE ALSO : line_graph 489

10.0.799 arc_number ______ number of arcs of a graph

CALLING SEQUENCE :

```
ma = arc_number(g)
```

PARAMETERS :

`g` : graph list
`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 478, node_number 503

10.0.800 articul ______ finds one or more articulation points

CALLING SEQUENCE :

```
nart = articul([i],g)
```

PARAMETERS :

`g` : graph list
`i` : integer
`nart` : integer row vector

DESCRIPTION :

`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.

EXAMPLE :

```
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('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];
g('node_diam')=[1:(g('node_number'))]+20;
show_graph(g);
nart = articul(g)
show_nodes(nart);
```

10.0.801 bandwr ______ bandwidth reduction for a sparse matrix

CALLING SEQUENCE :

```
[iperm,mrepi,profile,ierr] = bandwr(sp,[iopt])
[iperm,mrepi,profile,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
`profile` : 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). `profile` 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(profile(2:$)-profile(1:($-1)))` returns the bandwidth of the matrix. `ierr` is an integer indicating an error if its value is not zero.

EXAMPLE :

```
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('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];
// THE GRAPH
show_graph(g);
a=graph_2_mat(g,'node-node');
ww=tril(a)'+eye();
ww1=full(ww);
xset('window',0)
hist3d((ww1+tril(ww1',-1)+tril(ww1,-1)'),52,85);
// BANDWIDTH REDUCTION FOR THE MATRIX
[iperm,mrepi,profile,ierr]=bandwr(ww);
max(profile(2:$)-profile(1:($-1)))
// GRAPH WITH THE NEW NUMBERING
g2=g;g2('node_name')=string(iperm);
show_graph(g2,'new')
// NEW MATRIX
n=g('node_number');
yy=ww1(mrepi,mrepi);
xset('window',1)
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('node_x')=g('node_x');g1('node_y')=g('node_y');
// GRAPH
//show_graph(g1,'rep');
[lp,la,ls] = adj_lists(1,n,g1('tail'),g1('head'));
[iperm,mrepi,profile,ierr]=bandwr(lp,ls,n,0);
g2=g;g2('node_name')=string(iperm);
show_graph(g2,'new');
```

10.0.802 best_match ______ best matching of a graph

CALLING SEQUENCE :

```
[card,match] = best_match(g)
```

PARAMETERS :

`g` : graph list
`card` : integer
`match` : integer row vector

DESCRIPTION :
`best_match` finds an optimal 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 optimal matching or 0 if `i` is unmatched.

EXAMPLE :

```
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('node_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('node_y')=[yy 198 252 183 148 172 256 259 258 167 109 104 253];
show_graph(g);
[card,match] = best_match(g);
sp=sparse([ta' he'],[1:size(ta,2)]',[n,n]);
sp1=sparse([[1:n]' match'],ones(1,size(match,2))',[n,n]);
[ij,v,mn]=spget(sp.*sp1);
show_arcs(v');
//
// WITH A LARGER GRAPH
g=load_graph(SCI+'/demos/metanet/mesh1000');
g('directed')=0;
ta=g('tail');he=g('head');n=node_number(g);
show_graph(g,'new',[3000,1000]);
[card,match] = best_match(g);
sp=sparse([ta' he'],[1:size(ta,2)]',[n,n]);
sp1=sparse([[1:n]' match'],ones(1,size(match,2))',[n,n]);
[ij,v,mn]=spget(sp.*sp1);
show_arcs(v');
```

SEE ALSO : `perfect_match` 505

10.0.803 chain_struct ____ chained structure from adjacency lists of a graph

CALLING SEQUENCE :

```
[fe,che,fn,chn] = chain_struct(g)
[fe,che,fn,chn] = chain_struct(lp,la,ls)
```

PARAMETERS :

`g` : graph list
`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)`.

EXAMPLE :

```
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('node_x')=[116 231 192 323 354 454 305 155];
g('node_y')=[118 116 212 219 117 185 334 316];
show_graph(g);
[fe,che,fn,chn] = chain_struct(g)
```

SEE ALSO : `adj_lists` 467, `graph_2_mat` 480

10.0.804 check_graph ____ checks a Scilab graph list

CALLING SEQUENCE :

```
check_graph(g)
```

PARAMETERS :

`g` : graph list to check

DESCRIPTION :
`check_graph` checks its argument `g` to see if it is a graph list. 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.

Moreover, the names of the node must be different. In fact, this do not give errors in Scilab, but strange behaviour can appear when using the Metanet window. So, this is not checked by `check_graph` because it is time consuming. It is only checked when loading, saving or showing a graph.

SEE ALSO : `graph-list` 482

10.0.805 circuit ____ finds a circuit or the rank function in a directed graph

CALLING SEQUENCE :

```
[p,r] = circuit(g)
```

PARAMETERS :

`g` : graph list
`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 :

`circuit` tries to find 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 `[]`.

EXAMPLE :

```
// 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('node_x')=[116 231 192 323 354 454 305 155];
g('node_y')=[ 118 116 212 219 117 185 334 316];
show_graph(g);
p=circuit(g)
show_arcs(p)
// graph without circuit
g=make_graph('foo',1,4,[1 2 2 3],[2 3 4 4]);
[p,r]=circuit(g)
```

10.0.806 connex ____ connected components

CALLING SEQUENCE :

```
[nc,ncomp] = connex(g)
```

PARAMETERS :

`g` : graph list
`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.

EXAMPLE :


```
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('node_x')=[197 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[76 181 276 278 276 83 174 281 177 86 175 90 290 397 399];
show_graph(g);
[nc,ncomp]=connex(g)
g('node_color')=10+ncomp;
g('node_diam')=10+10*ncomp;
x_message('Displaying the connected components of this graph');
show_graph(g);
```

SEE ALSO : con_nodes 474, is_connex 488, strong_connex 514, strong_con_nodes 514

10.0.807 con_nodes ____ set of nodes of a connected component

CALLING SEQUENCE :

```
ns = con_nodes(i,g)
```

PARAMETERS :

`i` : integer, number of the connected component
`g` : graph list
`ns` : row vector, node numbers of the 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.

EXAMPLE :

```
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('node_x')=[197 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[76 181 276 278 276 83 174 281 177 86 175 90 290 397 399];
show_graph(g);
con_nodes(2,g)
x_message('Displaying the nodes of component #2');
n=g('node_number');
nodecolor=0*ones(1,n);
nodecolor(1,con_nodes(2,g))=11*ones(con_nodes(2,g));
g('node_color')=nodecolor;
nodediam=20.*ones(1,n);
nodediam(1,con_nodes(2,g))=30*ones(con_nodes(2,g));
g('node_diam')=nodediam;
show_graph(g);
```

SEE ALSO : connex 473, is_connex 488, strong_connex 514, strong_con_nodes 514

10.0.808 contract_edge ______ contracts edges between two nodes

CALLING SEQUENCE :

```
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` : graph list
`g1` : graph list of 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.

EXAMPLE :

```
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);
g1=contract_edge(10,13,g);
show_graph(g1,'new');
```

SEE ALSO : `add_edge` 466, `add_node` 466, `delete_arcs` 476, `delete_nodes` 477

10.0.809 convex_hull ______ convex hull of a set of points in the plane

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
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('node_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('node_y')=[yy 198 252 183 148 172 256 259 258 167 109 104 253];
show_graph(g);
xy=[g('node_x');g('node_y')];
[nhull,ind] = convex_hull(xy)
show_nodes(ind);
```

10.0.810 cycle_basis ______ basis of cycle of a simple undirected graph

CALLING SEQUENCE :

```
spc = cycle_basis(g)
```

PARAMETERS :

`g` : graph list
`spc` : sparse matrix

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. These cycles are returned in a sparse matrix `spc`: each line of this matrix corresponds to a cycle.

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).

EXAMPLE :

```
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('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
gt('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
gt('edge_color')=modulo([1:(edge_number(gt))],15)+1;
gt('node_diam')=[1:(gt('node_number'))]+20;
show_graph(gt);
g=graph_simp(gt);
g('edge_color')=modulo([1:(edge_number(g))],15)+1;
g('node_diam')=gt('node_diam');
g('default_edge_hi_width')=12;
show_graph(g);
spc=cycle_basis(g);
for kk=1:(size(spc,1)),
  aaa=spc(kk,:);aaa=full(aaa);aaa(aaa==0)=[];
  show_arcs(aaa);
end;
```

SEE ALSO : `min_weight_tree` 501, `graph_simp` 485

10.0.811 delete_arcs ____ deletes all the arcs or edges between a set of nodes

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
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` 466, `add_node` 466, `delete_nodes` 477

10.0.812 delete_nodes ______ deletes nodes

CALLING SEQUENCE :

```
g1 = delete_nodes(v,g)
```

PARAMETERS :

`v` : vector of integers, numbers of nodes to be deleted
`g` : graph list
`g1` : graph list of the new graph with deleted nodes

DESCRIPTION :
`delete_nodes` returns the graph `g1`, with the nodes given by the vector `v` being deleted.

EXAMPLE :

```
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);
v=[10 13 4];
gt=delete_nodes(v,g);
show_graph(gt,'new');
```

SEE ALSO : `add_edge` 466, `add_node` 466, `delete_arcs` 476

10.0.813 edge_number ______ number of edges of a graph

CALLING SEQUENCE :

```
ma = edge_number(g)
```

PARAMETERS :

`g` : graph list
`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. If the graph is undirected, it is half the number of edges. It is always equal to the dimension of `g('tail')` and `g('head')`.

SEE ALSO : `arc_number` 468, `node_number` 503

10.0.814 find_path ______ finds a path between two nodes

CALLING SEQUENCE :

```
p = find_path(i,j,g)
```

PARAMETERS :

`i` : integer, number of start node
`j` : integer, number of end node
`g` : graph list
`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.

EXAMPLE :

```
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);
p=find_path(1,14,g);
edgecolor=1*ones(ta); edgecolor(p)=11*ones(p); g('edge_color')=edgecolor;
show_graph(g); show_arcs(p);
```

SEE ALSO : `nodes_2_path` 504, `shortest_path` 510

10.0.815 gen_net ______ generation of a network

CALLING SEQUENCE :

```
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` : graph list

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.

EXAMPLE :

```
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` 494

10.0.816 girth ________ girth of a directed graph

CALLING SEQUENCE :

```
d = girth(g)
```

PARAMETERS :

`g` : graph list
`d` : integer

DESCRIPTION :

`girth` computes the length (number of arcs) of the shortest cycle in an unweighted directed graph `g`.

EXAMPLE :


```
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('node_x')=[285  284  335  160  405  189  118  45];
g('node_y')=[266  179   83  176  368  252  64  309];
show_graph(g);
d=girth(g)
```

10.0.817 glist ________ graph list creation

CALLING SEQUENCE :

```
g = glist(a1, ... ,a34)
```

DESCRIPTION :

`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_siz '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_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` 472, `graph-list` 482, `make_graph` 490

10.0.818 graph_2_mat ___ node-arc or node-node incidence matrix of a graph

CALLING SEQUENCE :

```
a = graph_2_mat(g,mat)
```

PARAMETERS :

`g` : graph list
`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`

EXAMPLE :

```
g=load_graph(SCI+'/demos/metanet/colored');
a=graph_2_mat(g)
a=graph_2_mat(g,'node-node')
```

SEE ALSO : `mat_2_graph` 491

10.0.819 graph_center _______ center of a graph

CALLING SEQUENCE :

```
[no,rad] = graph_center(g)
```

PARAMETERS :

`g` : graph list
`no` : integer
`rad` : integer

DESCRIPTION :
`graph_center` computes the 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.

EXAMPLE :

```
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('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];
g('node_diam')=[1:(g('node_number'))]+20;
show_graph(g);
[no,rad] = graph_center(g)
show_nodes(no);
```

SEE ALSO : `graph_diameter` 482

10.0.820 graph_complement _______ complement of a graph

CALLING SEQUENCE :

```
g1 = graph_complement(g,[gmax])
```

PARAMETERS :

`g` : graph list
`gmax` : graph list
`g1` : graph list of the new graph

DESCRIPTION :
`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.

EXAMPLE :


```
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('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];
g('edge_color')=modulo([1:(edge_number(g))],15)+1;
g('node_diam')=[1:(g('node_number'))]+20;
show_graph(g);
g1=graph_complement(g);
show_graph(g1,'new');
g=graph_complement(g1);
show_graph(g);
```

SEE ALSO : `graph_sum` 486, `graph_simp` 485

10.0.821 graph_diameter _______ diameter of a graph

CALLING SEQUENCE :

```
[d,p] = graph_diameter(g)
```

PARAMETERS :

`g` : graph list
`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.

EXAMPLE :

```
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('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];
g('node_diam')=[1:(g('node_number'))]+20;
show_graph(g);
[d,p] = graph_diameter(g)
show_arcs(p);
```

SEE ALSO : `graph_center` 481

10.0.822 graph-list _______ description of graph list

DESCRIPTION :
A graph in Scilab is represented by a Scilab typed list. We call it a graph list.

You will find below the complete description of the list. Each element is described by one or more lines. The first line gives the name of the element and its definition. Additional informations, such as the default for elements that can have one, are given in the other lines. Indeed, only the 5 first elements must have a value in the list, all the others can be given the empty vector `[]` as a value, and then the default is used when it is needed by functions or by the Metanet window. For instance, you can define a graph list by

```
g=make_graph('min',1,1,[1],[1]);
```

which is the simplest graph you can create in Metanet (it is directed, has one node and one loop arc on this node).

The name of the element in the list is very important because it is used to access the elements of the list. For instance, if `g` is a graph list, to get the name of the graph, you only have to do:

```
g('name')
```

and if you want to change the name of the graph to `'toto'`, you have to do:

```
g('name')='toto';
```

Moreover, you can get the number of edges and the number of arcs of the graph by using `edge_number(g)` and `arc_number(g)` (these names do not correspond to elements of the list). For compatibility, `node_number(g)` can also be used instead of `g('node_number')`.

A graph list can be syntactically correct but not represent a good graph. You can use the function `check_graph` to check it. Moreover it is a good idea to give nodes different names. In fact, this does not give errors in Scilab, but strange behaviour can appear when using the Metanet window. This is not checked by `check_graph` because it is time consuming. It is only checked when loading, saving or showing a graph.

The elements of a graph list are given below:

`name:` - the name of the graph
- it is a string with a maximum of 80 characters
`directed:` - flag giving the type of the graph
- it is equal to 1 (graph directed) or equal to 0 (graph undirected)
`node_number:` - number of nodes
`tail:` - row vector of the tail node numbers
`head:` - row vector of the head node numbers
`node_name:` - row vector of node names
- the names of the nodes must be different
- default is the node numbers as node names
`node_type:` - row vector of the node types
- the type is an integer from 0 to 2, default is 0 (plain node):
- 0 = plain node
- 1 = sink node
- 2 = source node
`node_x:` - row vector of the x coordinate of the nodes
- default is computed
`node_y:` - row vector of the y coordinate of the nodes
- default is computed
`node_color:` - row vector of the node colors
- the color is an integer from 0 to 16, default is 0 (default foreground):
- 0 = default foreground
- 1 = navyblue
- 2 = blue
- 3 = skyblue
- 4 = aquamarine
- 5 = forestgreen
- 6 = green
- 7 = lightcyan
- 8 = cyan
- 9 = orange
- 10 = red
- 11 = magenta
- 12 = violet

- 13 = yellow
- 14 = gold
- 15 = beige
- 16 = background
`node_diam:` - row vector of the size of the node diameters in pixels
- a node is drawn as a circle
- default is the value of element `default_node_diam`
`node_border:` - row vector of the size of the node borders in pixels
- a node is drawn as a circle
- default is the value of element `default_node_border`
`node_font_size:` - row vector of the size of the font used to draw the name of the node
- you can choose 8, 10, 12, 14, 18 or 24
- default is the value of element `default_font_size`
`node_demand:` - row vector of the node demands
- default is 0
`edge_name:` - row vector of the edge names
- it is better that the names of the edges are different, but this is not an error
- default is the edge numbers as edge names
`edge_color:` - row vector of the edge colors
- the color is an integer from 0 to 16 (see `node_color`)
- default is 0 (default foreground)
`edge_width:` - row vector of the size of the edge widths in pixels
- default is the value of element `default_edge_width`
`edge_hi_width:` - row vector of the size of the highlighted edge widths in pixels
- default is the value of element `default_edge_hi_width`
`edge_font_size:` - row vector of the size of the fonts used to draw the name of the edge
- you can choose 8, 10, 12, 14, 18 or 24
- default is the value of element `default_font_size`
`edge_length:` - row vector of the edge lengths
- default is 0
`edge_cost:` - row vector of the edge costs
- default is 0
`edge_min_cap:` - row vector of the edge minimum capacities
- default is 0
`edge_max_cap:` - row vector of the edge maximum capacities
- default is 0
`edge_q_weight:` - row vector of the edge quadratic weights
- default is 0
`edge_q_orig:` - row vector of the edge quadratic origins
- default is 0
`edge_weight:` - row vector of the edge weights
- default is 0
`default_node_diam:` - default size of the node diameters of the graph
- default is 20 pixels
`default_node_border:` - default size of the node borders of the graph
- default is 2 pixels
`default_edge_width:` - default size of the edge widths of the graph
- default is 1 pixel
`default_edge_hi_width:` - default size of the highlighted edge widths of the graph
- default is 3 pixels
`default_font_size:` - default size of the font used to draw the names of nodes and edges
- default is 12
`node_label:` - row vector of node labels
`edge_label:` - row vector of edge labels

EXAMPLE :

```
g=load_graph(SCI+'/demos/metanet/mesh100');
g('node_color')=int(rand(1:g('node_number'))*16);
g('edge_color')=int(rand(1:edge_number(g))*16);
show_graph(g)
```

SEE ALSO : `arc_number` 468, `check_graph` 472, `edge_number` 478, `glist` 480, `make_graph` 490, `node_number` 503

10.0.823 graph_power — kth power of a directed 1-graph

CALLING SEQUENCE :

```
g1 = graph_power(g,k)
```

PARAMETERS :

`g` : graph list of the graph
`k` : integer
`g1` : graph list 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.

EXAMPLE :

```
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('node_x')=[285  284  335  160  405  189  118  45];
g('node_y')=[266  179   83  176  368  252  64  309];
show_graph(g);
g1=graph_power(g,2);
show_graph(g1,'new');
```

10.0.824 graph_simp — converts a graph to a simple undirected graph

CALLING SEQUENCE :

```
g1 = graph_simp(g)
```

PARAMETERS :

`g` : graph list of the old graph
`g1` : graph list 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.

EXAMPLE :


```
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 15 16
16 17 17];
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 16 1
17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163  63  98 164 162 273 235 267 384 504 493 409 573 601 627 642];
g('node_y')=[ 59 133 223 311 227 299 221 288 384 141 209 299 398 383 187 121 301];
show_graph(g);
g1=graph_simp(g);
show_graph(g1,'new');
```

10.0.825 graph_sum — sum of two graphs

CALLING SEQUENCE :

```
g2 = graph_sum(g,g1)
```

PARAMETERS :

`g` : graph list
`g1` : graph list
`g2` : graph list of the new graph

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.

EXAMPLE :

```
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('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];
g('edge_color')=modulo([1:(edge_number(g))],15)+1;
g('edge_width')=ones(1,(edge_number(g)));
g('node_diam')=[1:(g('node_number'))]+20;
g('node_name')=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q'];
show_graph(g);
ta=[2 3 4 5 11 12 1];
he=[10 5 6 7 15 17 7];
g1=make_graph('foo',1,17,ta,he);
g1('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g1('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
g1('edge_color')=modulo([1:(edge_number(g1))],15)+1;
g1('edge_width')=10*ones(1,(edge_number(g1)));
g1('node_diam')=[1:(g1('node_number'))]+20;
g1('node_name')=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q'];
show_graph(g1,'new');
g2=graph_sum(g,g1);
show_graph(g2,'new');
```

SEE ALSO : `graph_complement` 481, `graph_union` 487

10.0.826 graph_union ______ union of two graphs

CALLING SEQUENCE :

```
g2 = graph_union(g,g1)
```

PARAMETERS :

`g` : graph list
`g1` : graph list
`g2` : graph list of the new graph

DESCRIPTION :
`graph_union` 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`. `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.

EXAMPLE :

```
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];
g('edge_color')=modulo([1:(edge_number(g))],15)+1;
g('node_diam')=[1:(g('node_number'))]+20;
g('node_name')=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q'];
show_graph(g);
l=netwindows(); nw=l(2);
v=[7 8 9 10 11 12 13];
show_nodes(v);
g1=subgraph(v,'nodes',g);
show_graph(g1,'new');
v=[1 2 5 6 7 8 9 10];
netwindow(nw);
show_nodes(v);
g2=subgraph(v,'nodes',g);
show_graph(g2,'new');
g=graph_union(g1,g2);
show_graph(g,'new');
```

SEE ALSO : supernode 516, subgraph 515

10.0.827 hamilton ______ hamiltonian circuit of a graph

CALLING SEQUENCE :

```
cir = hamilton(g)
```

PARAMETERS :

`g` : graph list
`cir` : integer row vector

DESCRIPTION :
`hamilton` finds an hamiltonian circuit (if it exists) of the directed graph `g`.

EXAMPLE :

```
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('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];
g('node_diam')=[1:(g('node_number'))]+20;
show_graph(g);
cir=hamilton(g)
show_arcs(cir);
```

10.0.828 is_connex ______ connectivity test

CALLING SEQUENCE :

```
res = is_connex(g)
```

PARAMETERS :

`g` : graph list
`res` : integer, result of the test

DESCRIPTION :
`is_connex` returns 1 if the graph `g` is connected and 0 otherwise.

EXAMPLE :

```
g=make_graph('foo',1,3,[1,2,3,1],[2,3,1,3]);
is_connex(g)
g=make_graph('foo',1,4,[1,2,3,1],[2,3,1,3]);
is_connex(g)
```

SEE ALSO : con_nodes 474, strong_connex 514

10.0.829 knapsack ______ solves a 0-1 multiple knapsack problem

CALLING SEQUENCE :

```
[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 :
`knapsack` solve a 0-1 multiple knapsack problem with n (n >= 2) items and m knapsacks (m >= 1). `profit` is the vector of the (integer) profits of the n items and `weight` is the vector of the corresponding (integer) weights. `capa` is the vector of the (integer) capacities of the m knapsacks. `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` can be omitted or can have a negative value. `earn` is the value of the criterium for the "optimal" solution and `ind` is a vector giving the optimal location: `ind(i)` gives the number of the knapsack where item i is inserted and this value is 0 if the item i is not in the optimal solution.

We recall that the problem to be solved is the following: `p(i)` and `w` denote respectively the profit and the weight of the item `i` 1=1,...,n; `c(j)` denotes the capacity of the knapsack `j` j=1,...,m; `q(j,i)` denotes the quantity of item `i` in knapsack `j` (in fact 0 or 1).

We want to maximize the global profit `E`: `E=p(1)*[x(1,1)+...+x(m,1)]+...+p(n)*[x(1,n)+...+x(m,n)]` under the constraints: `[w(1)*x(j,1)+...+w(n)*x(j,m)] <= c(j) ; j=1,...,m [x(1,i)+...+x(m,i)] <= 1 ; i=1,...,n x(j,i)= 0 or 1 p(),w(),c()` are positive integers.

EXAMPLE :

```
weight=ones(1,15).*.[1:4];
profit=ones(1,60);
capa=[15 45 30 60];
[earn,ind]=knapsack(profit,weight,capa)
```

SEE ALSO : `qassign` 508

10.0.830 line_graph ______ graph with nodes corresponding to edges

CALLING SEQUENCE :

```
g1 = line_graph(g)
```

PARAMETERS :

`g` : graph list of the old graph
`g1` : graph list 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`.

EXAMPLE :

```
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('node_x')=[281  284  360  185  405  182  118  45];
g('node_y')=[262  179  130  154  368  248  64  309];
show_graph(g);
g1=line_graph(g);
show_graph(g1,'new');
```

SEE ALSO : `arc_graph` 468

10.0.831 load_graph ______ loads a graph

CALLING SEQUENCE :

```
g = load_graph(name)
```

PARAMETERS :

`name` : string, the path of the graph to load
`g` : graph list

DESCRIPTION :
`name` is the name of a graph file which contains the ASCII description of a graph. Such a file must have the `"graph"` extension. `name` can be the name or the pathname of the file; if the `"graph"` extension is missing in `name`, it is assumed. `load_graph` returns the corresponding graph list.

EXAMPLE :

```
g=load_graph(SCI+'/demos/metanet/mesh100.graph');
show_graph(g);
g=load_graph(SCI+'/demos/metanet/colored');
show_graph(g,'new');
```

SEE ALSO : `save_graph` 510

10.0.832 make_graph ______ makes a graph list

CALLING SEQUENCE :

```
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` : graph list

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`.

EXAMPLE :

```
// 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('node_x')=[120  98  87 188 439 698 226 127 342 467 711 779 477];
g('node_y')=[ 21 184 308 426 435 428 129 360 435  55 109 320 321];
show_graph(g)
// creating same graph without isolated node and 14 arcs.
g=make_graph('foo',1,0,ta,he);
g('node_x')=[120  98 226 127 342 467 711 779 477];
g('node_y')=[ 21 184 129 360 435  55 109 320 321];
show_graph(g,'new')
```

SEE ALSO : graph-list 482

10.0.833 mat_2_graph graph from node-arc or node-node incidence matrix

CALLING SEQUENCE :

```
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 : graph list

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.

EXAMPLE :

```
g=load_graph(SCI+'/demos/metanet/colored');
show_graph(g);
a=graph_2_mat(g);
g1=mat_2_graph(a,1);
g1('node_x')=g('node_x'); g1('node_y')=g('node_y');
show_graph(g1,'new');
a=graph_2_mat(g,'node-node');
g1=mat_2_graph(a,1,'node-node');
g1('node_x')=g('node_x'); g1('node_y')=g('node_y');
show_graph(g1,'new');
```

SEE ALSO : adj_lists 467, chain_struct 471, graph_2_mat 480

10.0.834 max_cap_path maximum capacity path

CALLING SEQUENCE :

```
[p,cap] = max_cap_path(i,j,g)
```

PARAMETERS :

i,j : integers, node numbers
g : graph list
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 element edge_max_cap of the graph list. 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).

EXAMPLE :

```
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);
ma=edge_number(g);
g('edge_max_cap')=int(rand(1,ma)*16)+5;
[p,cap]=max_cap_path(1,14,g);
edgecolor=1*ones(1,ma); edgecolor(p)=11*ones(p); g('edge_color')=edgecolor;
x_message(['The maximum capacity is: '+string(cap);
           'Showing the corresponding path']);
show_graph(g); show_arcs(p);
```

10.0.835 max_clique maximum clique of a graph

CALLING SEQUENCE :

```
[size,nodes] = max_clique(g,[ind])
```

PARAMETERS :

g : graph list
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.

EXAMPLE :

```
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('node_x')=[106 199 369 467 470 403 399 347 308 269 184 108 199 268 345 272];
g('node_y')=[341 420 422 321 180 212 286 246 193 244 243 209  59 134  51 348];
g('node_diam')=[1:(g('node_number'))]+20;
show_graph(g);
[ns,no] = max_clique(g);
show_nodes(no);
g1=graph_complement(g);
[ns,no] = max_clique(g1);
show_nodes(no);
```

10.0.836 max_flow ______ maximum flow between two nodes

CALLING SEQUENCE :

```
[v,phi,flag] = max_flow(i,j,g)
```

PARAMETERS :

`i` : integer, number of start node
`j` : integer, number of end node
`g` : graph list
`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 elements `edge_min_cap` and `edge_max_cap` of the graph list. The value of the maximum capacity must be greater than or equal to the value of the minimum capacity. If the value of `edge_min_cap` or `edge_max_cap` is not given (empty row vector `[]`), it is assumed to be equal to 0 on each edge.

EXAMPLE :

```
ta=[1 1 2 2 3 3 4 4 5 5 5 5 6 6 6 7 7 15 15 15 15 15 15];
ta=[ta, 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];
he=[he, 5 6 7 16 16 16 16 16 16 16];
n=16;
g=make_graph('foo',1,n,ta,he);
g('node_x')=[42 615 231 505 145 312 403 233 506 34 400 312 142 614 260 257];
g('node_y')=[143 145 154 154 147 152 157 270 273 279 269 273 273 274 50 376];
ma=edge_number(g);
g('edge_max_cap')=ones(1,ma);
g('edge_min_cap')=zeros(1,ma);
source=15; sink=16;
nodetype=0*ones(1,n); nodetype(source)=2; nodetype(sink)=1;
g('node_type')=nodetype;
nodecolor=0*ones(1,n); nodecolor(source)=11; nodecolor(sink)=11;
```



```
g('node_color')=nodecolor;
show_graph(g);
[v,phi,ierr]=max_flow(source,sink,g);
ii=find(phi<>0); edgecolor=phi; edgecolor(ii)=11*ones(ii);
g('edge_color')=edgecolor;
edgefontsize=8*ones(1,ma); edgefontsize(ii)=18*ones(ii);
g('edge_font_size')=edgefontsize;
g('edge_label')=string(phi);
show_graph(g);
```

10.0.837 mesh2d ______ triangulation of n points in the plane

CALLING SEQUENCE :

```
[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.

EXAMPLE :

```
// 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('node_x')=300*x;
g('node_y')=300*y;
g('default_node_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('node_x')=300*xnew;
g('node_y')=300*ynew;
g('default_node_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('node_x')=300*xnew;
g('node_y')=300*ynew;
g('default_node_diam')=3;
show_graph(g)
```

10.0.838 metanet ______ opens a Metanet window

CALLING SEQUENCE :

```
window = metanet([path,winsize])
```

PARAMETERS :

`path` : string, directory where graph files are searched
`winsize` : row vector defining the size of Metanet window
`window` : integer, window number

DESCRIPTION :

This function is used to open a Metanet window from Scilab.

`path` is an optional argument; it is the directory where graph files are searched. If this path is the pathname of a graph, this graph is displayed in the Metanet window and the directory of this pathname becomes the default directory. By default, `path` is the working directory.

`winsize` is an optional argument; it is a row vector `[width height]` giving the size in pixels of Metanet window. The default is `[1000 1000]`.

Usually, `show_graph` is used and `metanet` is seldom used.

Each time `metanet` is executed, a new window is created and its number is incremented by 1.

SEE ALSO : `netclose` 502, `netwindow` 503, `netwindows` 503, `show_graph` 512

10.0.839 metanet_sync ______ asynchronous or synchronous mode in Metanet

CALLING SEQUENCE :

```
res = metanet_sync([flag])
```

PARAMETERS :

`res` : integer
`flag` : integer

DESCRIPTION :

By default Metanet windows work with Scilab in asynchronous mode, ie Scilab proceeds without waiting for graphics commands sent to Metanet window to terminate: these commands are `show_graph`, `show_arcs` and `show_nodes`. This mode is the most efficient. But when running a lots of such graphical commands, problems can arise.

`metanet_sync(0)` changes to asynchronous mode (default).

`metanet_sync(1)` changes to synchronous mode.

`metanet_sync()` returns the current mode (0 = asynchronous, 1 = synchronous).

10.0.840 min_lcost_cflow _____ minimum linear cost constrained flow

CALLING SEQUENCE :

```
[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` : graph list
`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 elements `edge_min_cap` and `edge_max_cap` of the graph list. The value of the minimum capacity must be equal to zero, and the value of the maximum capacity must be non negative and must be integer numbers. If the value of `edge_min_cap` or `edge_max_cap` is not given (empty row vector `[]`), it is assumed to be equal to 0 on each edge.

The costs on the edges are given by the element `edge_cost` of the graph list. The costs must be non negative. If the value of `edge_cost` is not given (empty row vector `[]`), it is assumed to be equal to 0 on each edge.

The demands, element `node_demand` of the graph list, must be equal to zero.

This function uses the algorithm of Busacker and Goven.

EXAMPLE :

```
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('node_x')=[194 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[56 181 276 278 276 103 174 281 177 86 175 90 290 397 399];
show_graph(g);
g1=g; ma=arc_number(g1); n=g1('node_number');
g1('edge_min_cap')=0*ones(1,ma);
rand('uniform');
g1('edge_max_cap')=round(20*rand(1,ma))+ones(1,ma);
g1('edge_cost')=10*rand(1,ma)+ones(1,ma);
source=15; sink=1; cv=5;
[c,phi,v]=min_lcost_cflow(source,sink,cv,g1);
x_message(['The cost is: '+string(c);
           'Showing the flow on the arcs']);
nodetype=0*ones(1,n); nodetype(source)=2; nodetype(sink)=1;
g1('node_type')=nodetype;
ii=find(phi<>0); edgecolor=phi; edgecolor(ii)=11*ones(ii);
g1('edge_color')=edgecolor;
edgefontsize=8*ones(1,ma); edgefontsize(ii)=18*ones(ii);
```



```
nodecolor=0*ones(1,n); nodecolor(source)=11; nodecolor(sink)=11;
g1('node_color')=nodecolor;
g1('edge_font_size')=edgefontsize;
g1('edge_label')=string(phi);
show_graph(g1);
```

SEE ALSO : `min_lcost_flow1` 498, `min_lcost_flow2` 499, `min_qcost_flow` 500

10.0.841 min_lcost_flow1 _____ minimum linear cost flow

CALLING SEQUENCE :

```
[c,phi,flag] = min_lcost_flow1(g)
```

PARAMETERS :

`g` : graph list
`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 elements `edge_min_cap` and `edge_max_cap` of the graph list. 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 `edge_min_cap` or `edge_max_cap` is not given (empty row vector `[]`), it is assumed to be equal to 0 on each edge.

The costs on the edges are given by the element `edge_cost` of the graph list. The costs must be non negative. If the value of `edge_cost` is not given (empty row vector `[]`), it is assumed to be equal to 0 on each edge.

The demands, element `node_demand` of the graph list, must be equal to zero.

This function uses the out-of-kilter algorithm.

EXAMPLE :

```
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('node_x')=[194 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[56 221 316 318 316 143 214 321 217 126 215 80 330 437 439];
show_graph(g);
g1=g;ma=arc_number(g1);
rand('uniform');
while %T then
  g1('edge_min_cap')=round(20*rand(1,ma));
  g1('edge_max_cap')=round(20*rand(1,ma))+g1('edge_min_cap')+33*ones(1,ma);
  g1('edge_cost')=round(10*rand(1,ma))+ones(1,ma);
  [c,phi,flag]=min_lcost_flow1(g1);
  if flag==1 then break; end;
end;
x_message(['The cost is: '+string(c);
           'Showing the flow on the arcs ']);
```

```
ii=find(phi<>0); edgecolor=phi; edgecolor(ii)=11*ones(ii);
g1('edge_color')=edgecolor;
edgefontsize=8*ones(1,ma); edgefontsize(ii)=18*ones(ii);
g1('edge_font_size')=edgefontsize;
g1('edge_label')=string(phi);
show_graph(g1);
```

SEE ALSO : min_lcost_cflow 497, min_lcost_flow2 499, min_qcost_flow 500

10.0.842 min_lcost_flow2 ________ minimum linear cost flow

CALLING SEQUENCE :

```
[c,phi,flag] = min_lcost_flow2(g)
```

PARAMETERS :

`g` : graph list
`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 elements `edge_min_cap` and `edge_max_cap` of the graph list. 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 `edge_min_cap` or `edge_max_cap` is not given (empty row vector `[]`), it is assumed to be equal to 0 on each edge.

The costs on the edges are given by the element `edge_cost` of the graph list. The costs must be non negative and must be integer numbers. If the value of `edge_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 element `node_demand` of the graph list. 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 `node_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.

EXAMPLE :

```
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('node_x')=[194 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[56 221 316 318 316 143 214 321 217 126 215 80 330 437 439];
show_graph(g);
g1=g; ma=arc_number(g1); n=g1('node_number');
g1('edge_min_cap')=0.*ones(1,ma);
x_message(['Random generation of data';
          'The first(s) generated problem(s) may be unfeasible']);
while %T then
 rand('uniform');
 g1('edge_max_cap')=round(20*rand(1,ma))+20*ones(1,ma);
 g1('edge_cost')=round(10*rand(1,ma)+ones(1,ma));
```



```
 rand('normal');
 dd=20.*rand(1,n)-10*ones(1,n);
 dd=round(dd-sum(dd)/n*ones(1,n));
 dd(n)=dd(n)-sum(dd);
 g1('node_demand')=dd;
 [c,phi,flag]=min_lcost_flow2(g1);
 if flag==1 then break; end;
end;
x_message(['The cost is: '+string(c);
          'Showing the flow on the arcs and the demand on the nodes']);
ii=find(phi<>0); edgecolor=phi; edgecolor(ii)=11*ones(ii);
g1('edge_color')=edgecolor;
edgefontsize=8*ones(1,ma); edgefontsize(ii)=18*ones(ii);
g1('edge_font_size')=edgefontsize;
g1('edge_label')=string(phi);
g1('node_label')=string(g1('node_demand'));
show_graph(g1);
```

SEE ALSO : min_lcost_cflow 497, min_lcost_flow1 498, min_qcost_flow 500

10.0.843 min_qcost_flow ________ minimum quadratic cost flow

CALLING SEQUENCE :

```
[c,phi,flag] = min_qcost_flow(eps,g)
```

PARAMETERS :

`eps` : scalar, precision
`g` : graph list
`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 elements `edge_min_cap` and `edge_max_cap` of the graph list. The value of the maximum capacity must be greater than or equal to the value of the minimum capacity. If the value of `edge_min_cap` or `edge_max_cap` is not given (empty row vector `[]`), it is assumed to be equal to 0 on each edge.

The costs on the edges are given by the elements `edge_q_orig` and `edge_q_weight` of the graph list. The cost on arc `u` is given by:

`(1/2)*edge_q_weight[u](phi[u]-edge_q_orig[u])^2` The costs must be non negative. If the value of `edge_q_orig` or `edge_q_weight` is not given (empty row vector `[]`), it is assumed to be equal to 0 on each edge.

This function uses an algorithm due to M. Minoux.

EXAMPLE :

```
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('node_x')=[194 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
```

```
g('node_y')=[56 221 316 318 316 143 214 321 217 126 215 80 330 437 439];
show_graph(g);
g1=g; ma=arc_number(g1);
rand('uniform');
while %T then
  g1('edge_min_cap')=round(5*rand(1,ma));
  g1('edge_max_cap')=round(20*rand(1,ma))+30*ones(1,ma);
  g1('edge_q_orig')=0*ones(1,ma);
  g1('edge_q_weight')=ones(1,ma);
  [c,phi,flag]=min_qcost_flow(0.001,g1);
 if flag==1 then break; end;
end;
x_message(['The cost is: '+string(c);
          'Showing the flow on the arcs']);
ii=find(phi<>0); edgecolor=phi; edgecolor(ii)=11*ones(ii);
g1('edge_color')=edgecolor;
edgefontsize=8*ones(1,ma); edgefontsize(ii)=18*ones(ii);
g1('edge_font_size')=edgefontsize;
g1('edge_label')=string(phi);
show_graph(g1);
```

SEE ALSO : `min_lcost_cflow` 497, `min_lcost_flow1` 498, `min_lcost_flow2` 499

10.0.844 min_weight_tree ________ minimum weight spanning tree

CALLING SEQUENCE :

```
t = min_weight_tree([i],g)
```

PARAMETERS :

`i` : integer, node number of the root of the tree
`g` : graph list
`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 element `edge_weight` of the graph list. If its value is not given (empty vector `[]`), 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 or the empty vector `[]`.

`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)`

EXAMPLE :

```
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=arc_number(g1); n=g1('node_number');
nodetype=0*ones(1,n); nodetype(1)=2; g1('node_type')=nodetype;
edgecolor=1*ones(1,ma); edgecolor(t)=11*ones(t); g1('edge_color')=edgecolor;
edgewidth=1*ones(1,ma); edgewidth(t)=4*ones(t); g1('edge_width')=edgewidth;
x_message('Minimum weight tree from node 1');
show_graph(g1);
```

10.0.845 neighbors ________ nodes connected to a node

CALLING SEQUENCE :

```
a = neighbors(i,g)
```

PARAMETERS :

`i` : integer
`g` : graph list
`a` : vector of integers

DESCRIPTION :
`neighbors` returns the numbers of the nodes connected with node i for graph `g` (directed or not).

EXAMPLE :

```
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('node_x')=[285  284  335  160  405  189  118  45];
g('node_y')=[266  179   83  176  368  252  64  309];
show_graph(g);
a=neighbors(6,g)
show_nodes(a);
```

SEE ALSO : `predecessors` 508, `successors` 516
"Scilab function"

10.0.846 netclose ________ closes a Metanet window

CALLING SEQUENCE :

```
netclose(window)
```

PARAMETERS :

`window` : integer, window number

DESCRIPTION :
Each Metanet window has a window number returned by the `metanet` and `show_graph` functions. This function is used to close the Metanet window with number `window`.

SEE ALSO : `metanet` 496, `netwindow` 503, `netwindows` 503, `show_graph` 512

10.0.847 netwindow ______ chooses a Metanet window

CALLING SEQUENCE :

```
netwindow(window)
```

PARAMETERS :

`window` : integer, window number

DESCRIPTION :
This function is used to change the Metanet window. Each Metanet window has a window number returned by the `metanet` and `show_graph` functions. To use the Metanet window associated to window number `window`, use `netwindow(window)`. The numbers of existing windows are given by the function `netwindows`.

SEE ALSO : `metanet` 496, `netclose` 502, `netwindows` 503, `show_graph` 512

10.0.848 netwindows ______ gets the numbers of Metanet windows

CALLING SEQUENCE :

```
l = netwindows()
```

PARAMETERS :

`l` : list

DESCRIPTION :
This function returns a list `l`. Its first element is the row vector of all the Metanet windows and the second element is the number of the current Metanet window. This number is equal to 0 if no current Metanet window exists.

SEE ALSO : `metanet` 496, `netclose` 502, `netwindow` 503, `show_graph` 512

10.0.849 node_number ______ number of nodes of a graph

CALLING SEQUENCE :

```
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` 468, `edge_number` 478

10.0.850 nodes_2_path ______ path from a set of nodes

CALLING SEQUENCE :

```
p = nodes_2_path(ns,g)
```

PARAMETERS :

`ns` : row vector of integer numbers of the set of nodes
`g` : graph list
`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.

EXAMPLE :

```
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);
ns=[1 10 15 16 17 14 11 12 13 9 7 8 6];
g1=g; nodecolor=1*ones(g('node_x')); nodecolor(ns)=11*ones(ns);
g1('node_color')=nodecolor;
show_graph(g1); show_nodes(ns);
p=nodes_2_path(ns,g);
g1=g; edgecolor=1*ones(ta); edgecolor(p)=11*ones(p);
g1('edge_color')=edgecolor;
show_graph(g1); show_arcs(p);
show_nodes(ns,'sup');
```

SEE ALSO : `path_2_nodes` 505

10.0.851 nodes_degrees ______ degrees of the nodes of a graph

CALLING SEQUENCE :

```
[outdegree,indegree] = graph_degree(g)
```

PARAMETERS :

`g` : graph list
`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`.

EXAMPLE :

```
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);
[outdegree,indegree]=nodes_degrees(g)
```

SEE ALSO : adj_lists 467

10.0.852 path_2_nodes ______ set of nodes from a path

CALLING SEQUENCE :

```
ns = path_2_nodes(p,g)
```

PARAMETERS :

p : row vector of integer numbers of the arcs of the path
g : graph list
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.

EXAMPLE :

```
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);
p=[2 16 23 25 26 22 17 18 19 13 10 11];
g1=g; edgecolor=1*ones(ta); edgecolor(p)=11*ones(p);
g1('edge_color')=edgecolor;
show_graph(g1); show_arcs(p);
ns=path_2_nodes(p,g);
g1=g; nodecolor=1*ones(g1('node_number')); nodecolor(ns)=11*ones(ns);
g1('node_color')=nodecolor;
show_graph(g1);show_nodes(ns);
show_arcs(p,'sup');
```

SEE ALSO : nodes_2_path 504

10.0.853 perfect_match ______ min-cost perfect matching

CALLING SEQUENCE :

```
[cst,nmatch] = perfect_match(g,arcost)
```

PARAMETERS :

g : graph list

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.

EXAMPLE :

```
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('node_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('node_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);
sp=sparse([ta' he'],[1:size(ta,2)]',[n,n]);
sp1=sparse([[1:n]' nmatch'],ones(1,size(nmatch,2))',[n,n]);
[ij,v,mn]=spget(sp.*sp1);
show_arcs(v');
```

SEE ALSO : best_match 470

10.0.854 pipe_network ______ solves the pipe network problem

CALLING SEQUENCE :

```
[x,pi] = pipe_network(g)
```

PARAMETERS :

g : graph list
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 element 'edge_weigth' of the graph list.

The problem is solved by using sparse matrices LU factorization.

EXAMPLE :

```
ta=[1 1 2 2 3 3 4 4 5 5 5 5 6 6 6 7 7 15 15 15 15 15 15];
ta=[ta, 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];
he=[he, 5 6 7 16 16 16 16 16 16 16];
```

```
n=16;
g=make_graph('foo',1,n,ta,he);
g('node_x')=[42 615 231 505 145 312 403 233 506 34 400 312 142 614 260 257];
g('node_y')=[143 145 154 154 147 152 157 270 273 279 269 273 273 274 50 376];
show_graph(g);
g('node_demand')=[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];
g('edge_weight')=[w, 6 4 3 5 6];
[x,pi] = pipe_network(g)
```

10.0.855 plot_graph _____ general plot of a graph

CALLING SEQUENCE :

```
plot_graph(g,[rep,rep1])
```

PARAMETERS :

`g` : graph list
`rep` : row vector of 13 values for the parameters of the plot
`rep1` : row vector of 4 values defining the plotting rectangle

DESCRIPTION :
`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.

EXAMPLE :

```
// 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('node_type')=[1 1 1 2];g('node_name')=string([1:4]);
g('node_x')=[73 737 381 391]; g('node_y')=[283 337 458 142];
g('node_color')=[3 3 3 11];
g('node_diam')=[30 30 30 60];
```



```
g('edge_color')=[10 0 2 6 11 11 0 0 11];
rep=[2 2 1 1 2 2 2 2 2 2 2 2 2];
rep1=[100 -400 650 300];
xbasc(); plot_graph(g,rep,rep1);
rep=[2 1 1 1 2 2 2 2 2 2 2 2 2];
x_message('plot the graph with different parameters');
xbasc(); plot_graph(g,rep,rep1);
// plotting using dialogs
xbasc(); plot_graph(g);
xset("thickness",4);
xbasc();
plot_graph(g);
xset('default');
```

SEE ALSO : `show_graph` 512

10.0.856 predecessors _____ tail nodes of incoming arcs of a node

CALLING SEQUENCE :

```
a = predecessors(i,g)
```

PARAMETERS :

`i` : integer
`g` : graph list
`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` .

EXAMPLE :

```
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('node_x')=[285  284  335  160  405  189  118  45];
g('node_y')=[266  179   83  176  368  252  64  309];
show_graph(g);
a=predecessors(8,g)
show_nodes(a);
```

SEE ALSO : `neighbors` 502, `successors` 516

10.0.857 qassign _____ solves a quadratic assignment problem

CALLING SEQUENCE :

```
[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(
c, f and d are n x n real arrays; their diagonal entries are zero.

EXAMPLE :

```
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 488

10.0.858 salesman ______ solves the travelling salesman problem

CALLING SEQUENCE :

```
cir = salesman(g,[nstac])
```

PARAMETERS :

g : graph list
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.

EXAMPLE :

```
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('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];
g('node_diam')=[1:(g('node_number'))]+20;
show_graph(g);
g1=make_graph('foo1',1,17,[ta he],[he ta]);
m=arc_number(g1);
g1('edge_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;
show_arcs(cir);
```


10.0.859 save_graph ______ saves a graph

CALLING SEQUENCE :

```
save_graph(g,path)
```

PARAMETERS :

g : graph list
name : 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.

EXAMPLE :

```
g=load_graph(SCI+'/demos/metanet/mesh100');
show_graph(g);
unix('rm mymesh100.graph')
save_graph(g,'mymesh100.graph');
g=load_graph('mymesh100');
show_graph(g,'new');
```

SEE ALSO : load_graph 490

10.0.860 shortest_path ______ shortest path

CALLING SEQUENCE :

```
[p,lp] = shortest_path(i,j,g,[typ])
```

PARAMETERS :

i : integer, number of start node
j : integer, number of end node
g : graph list
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.

EXAMPLE :

```
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('node_x')=[194 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[56 181 276 278 276 103 174 281 177 86 175 90 290 397 399];
show_graph(g);
g1=g;ma=prod(size(g1('head')));
rand('uniform');
g1('edge_length')=int(20*rand(1,ma));
[p,lp]=shortest_path(13,1,g1,'length');
p
x_message(['Showing the arcs of the shortest path ';
           'Choose ""Display arc names"" in the Graph menu to see arc names']);
g1('edge_name')=string(g1('edge_length'));
edgecolor=ones(1:ma);edgecolor(p)=11*ones(p);
g1('edge_color')=edgecolor;
edgefontsize=12*ones(1,ma);edgefontsize(p)=18*ones(p);
g1('edge_font_size')=edgefontsize;
show_graph(g1);
```

SEE ALSO : find_path 478, nodes_2_path 504

10.0.861 show_arcs ______ highlights a set of arcs

CALLING SEQUENCE :

```
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 Metanet window. If the optional argument `sup` is equal to the string 'sup', the highlighting is superposed on the previous one.

By default, this function works in asynchronous mode (see `metanet_sync`).

EXAMPLE :

```
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;
 unix('sleep 2'); show_graph(g1);
 show_arcs(t);
end;
```

SEE ALSO : metanet_sync 496, show_nodes 512

10.0.862 show_graph ______ displays a graph

CALLING SEQUENCE :

```
nw = show_graph(g,[smode,scale])
nw = show_graph(g,'new',[scale,winsize])
```

PARAMETERS :

`g` : graph list
`smode` : string, mode value
`winsize` : row vector defining the size of Metanet window
`scale` : real value, scale factor
`nw` : integer

DESCRIPTION :
`show_graph` displays the graph `g` in the current Metanet window. If there is no current Metanet window, a Metanet window is created. The return value `nw` is the number of the Metanet 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 Metanet window, the new graph is displayed instead.

If the optional argument `smode` is equal to the string 'new', a new Metanet 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 Metanet window. The default is `[1000 1000]`.

The optional argument `scale` is the value of the scale factor when drawing the graph. The default value is 1.

The labels of the nodes and edges, if they exist, are displayed.

By default, this function works in asynchronous mode (see `metanet_sync`).

EXAMPLE :

```
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,2);
show_graph(g,0.5);
show_graph(g,1);
```

SEE ALSO : metanet_sync 496

10.0.863 show_nodes ______ highlights a set of nodes

CALLING SEQUENCE :

```
show_nodes(nodes,[sup])
```

PARAMETERS :

`nodes` : row vector of node numbers
`sup` : string, superposition flag

DESCRIPTION :

show_nodes highlights the set of nodes nodes of the displayed graph in the current Metanet window. If the optional argument sup is equal to the string 'sup', the highlighting is superposed on the previous one.

By default, this function works in asynchronous mode (see metanet_sync).

EXAMPLE :

```
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);
for i=2:3:g('node_number'), show_nodes([i]); end;
for i=1:3:g('node_number'), show_nodes([i],'sup'); end;
```

SEE ALSO : metanet_sync 496, show_arcs 511

10.0.864 split_edge ______ splits an edge by inserting a node

CALLING SEQUENCE :

```
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 : graph list
name : optional name of the added node
g1 : graph list 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.

EXAMPLE :

```
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);
gt=split_edge(1,2,g);
show_graph(gt,'new');
```

SEE ALSO : add_edge 466, add_node 466, delete_arcs 476, delete_nodes 477

10.0.865 strong_connex ______ strong connected components

CALLING SEQUENCE :

```
[nc,ncomp] = strong_connex(g)
```

PARAMETERS :

g : graph list
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.

EXAMPLE :

```
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('node_x')=[197 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[76 181 276 278 276 83 174 281 177 86 175 90 290 397 399];
show_graph(g);
[nc,ncomp]=strong_connex(g);
g1=g; g1('node_color')=8+ncomp; g1('node_diam')=10+5*ncomp;
x_message('Connected components of the graph');
show_graph(g1);
```

SEE ALSO : connex 473, con_nodes 474, strong_con_nodes 514

10.0.866 strong_con_nodes ______ set of nodes of a strong connected component

CALLING SEQUENCE :

```
ns = strong_con_nodes(i,g)
```

PARAMETERS :

i : integer, number of the strong connected component
g : graph list
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.

EXAMPLE :

```
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('node_x')=[197 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
```

```
g('node_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);
n=g('node_number');
nodecolor=0*ones(1,n); nodecolor(ncomp)=11*ones(ncomp);
g('node_color')=nodecolor;
nodediam=20*ones(1,n); nodediam(ncomp)=40*ones(ncomp);
g('node_diam')=nodediam;
x_message('Set of nodes of the strong connected component #3');
show_graph(g);
```

SEE ALSO : connex 473, con_nodes 474, strong_connex 514

10.0.867 subgraph ______ subgraph of a graph

CALLING SEQUENCE :

```
g1 = subgraph(v,ind,g)
```

PARAMETERS :

`v` : row vector, numbers of nodes or edges
`ind` : string, 'nodes' or 'edges'
`g` : graph list
`g1` : graph list 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.

EXAMPLE :

```
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];
g('edge_color')=modulo([1:(edge_number(g))],15)+1;
g('node_diam')=[1:(g('node_number'))]+20;
show_graph(g);
metanet_sync(1);
v=[2 3 4 5 17 13 10];
show_nodes(v);
g1=subgraph(v,'nodes',g);
show_graph(g1);
v=[10 13 12 16 20 19];
show_graph(g);
show_arcs(v);
g1=subgraph(v,'edges',g);
show_graph(g1);
metanet_sync(0);
```

SEE ALSO : add_edge 466, add_node 466, delete_arcs 476, delete_nodes 477, supernode 516

10.0.868 successors ______ head nodes of outgoing arcs of a node

CALLING SEQUENCE :

```
a = successors(i,g)
```

PARAMETERS :

`i` : integer
`g` : graph list
`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` .

EXAMPLE :

```
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('node_x')=[285  284  335  160  405  189  118  45];
g('node_y')=[266  179   83  176  368  252  64  309];
show_graph(g);
a=successors(6,g)
show_nodes(a);
```

SEE ALSO : neighbors 502, predecessors 508

10.0.869 supernode ______ replaces a group of nodes with a single node

CALLING SEQUENCE :

```
g1 = supernode(v,g)
```

PARAMETERS :

`v` : row vector, nodes numbers
`g` : graph list
`g1` : graph list 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.

EXAMPLE :

```
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];
g('edge_color')=modulo([1:(edge_number(g))],15)+1;
g('node_diam')=[1:(g('node_number'))]+20;
```

```
show_graph(g);
v=[7 10 13 9];
show_nodes(v);
g1=supernode(v,g);
show_graph(g1,'new');
```

SEE ALSO : add_edge 466, add_node 466, delete_arcs 476, delete_nodes 477

10.0.870 trans_closure ______ transitive closure

CALLING SEQUENCE :

```
g1 = trans_closure(g)
```

PARAMETERS :

g : graph list
g1 : graph list

DESCRIPTION :
trans_closure returns as a new graph list 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.

EXAMPLE :

```
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('node_x')=[129 200 283 281 128 366 122 333];
g('node_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('tail'); he1=g1('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('edge_color')=edgecolor;
x_message('Transitive closure of the graph');
show_graph(g1);
```

Chapter 11

Scicos

11.1 Scicos editor

11.1.1 scicos Block diagram editor and GUI for the hybrid simulator scicosim

CALLING SEQUENCE :

```
sys=scicos()
sys=scicos(sys,[menus])
sys=scicos(file,[menus])
```

PARAMETERS :

`sys` : a Scicos data structure

`file` : a character string. The path of a file containing the image of a Scicos data structure. These files may have `.cos` or `.cosf` extensions.

`menus` : a vector of character strings. It allows to select some of the Scicos menus. If `menus==[]` Scicos draws the diagram and the contents of each super blocks in separate windows without menu bar. This option is useful to print diagrams.

DESCRIPTION :

Scicos is a visual editor for constructing models of hybrid dynamical systems. Invoking Scicos with no argument opens up an empty Scicos window. Models can then be assembled, loaded, saved, compiled, simulated, using GUI of Scicos. The input and ouput arguments are only useful for debugging purposes. Scicos serves as an interface to the various block diagram compilers and the hybrid simulator scicosim.

SEE ALSO : `scicosim` 551, `scicos_main` 545, `scicos_menus` 520

11.1.2 scicos_menus Scicos menus description

DESCRIPTION :

Here is a list of operations available in Scicos:

`Main menus` :

`Edit` : Opens the diagram/palette edition menu.

`Simulate` : Opens the compilation/execution menu.

`Diagram` : Opens the diagram/file management menu.

`Block` : Opens the block management menu.

`Misc` : Opens miscellaneous menu items.

`Diagram/palette edition menu.` : This menu allows to edit diagram and palettes

`Palettes` : opens up a selection dialog where user may select a desired palette among all defined palettes.

`Context` : opens up a dialog where user may enter and modify Scilab instructions to be executed when diagram is loaded (`Edit../Load` menu) or evaluated (`Simulate../Eval` menu) (of course instructions are also evaluated when dialog returns).These instructions may be used to define Scilab variables whose names are used in the block parameters definition expressions.

`Move` : To move a block in main Scicos window, select first the Move menu item, then click on the selected block, drag the mouse to the desired block position and click again to fix the position.

`Copy` : To copy a block in main Scicos window, select first the Copy menu item, then click left on the to-be-copied block (in Scicos window or in a palette) , and finally click where you want the copy to be placed in Scicos window. This menu item remains active until user choose an other one

`Copy Region`: To copy a region in main Scicos window, select first the Copy menu item, then click right on a corner of the desired region (in Scicos window or in a palette) , drag to select the desired region, click to fix the selected region and finally click where you want the copy to be placed in Scicos window. If source diagram is a big region, selection may take a while.

`Replace` : To replace a block in the active editor Scicos window select first the Replace menu item, then select the replacement block (in Scicos window or in a palette) , and finally click on the to-be-replaced block. It is not possible to replace a connected block with another block with different port locations.

`Align` : To obtain nice diagrams, you can align ports of different blocks, vertically and horizontally. select first the Align menu item, then on the first port and finally on the second port. The block corresponding to the second port is moved. Connected blocks cannot be aligned.

`AddNew` :To add a newly defined block to the current palette or diagram select first this menu item, a dialog box will pop up asking for the name of the GUI function associated with the block. If this function is not already loaded it is searched in the current directory. The user may then click at the desired position of the block in the palette or diagram .

`Link` : This menu item is defined only in diagram edition mode. To connect an output port to an input port, select first the Link menu item, then on the intermediate points, if necessary, and finally on the input port. Scicos tries to draw horizontal and vertical lines to form links.

To split a link, select first the Link menu item, then on the link where the split should be placed, and finally on an input port. Only one link can go from and to a port. Link color can be changed directly by clicking on the link.

This menu item remains active until user choose an other one

`Delete` : To delete a block or a link, select first the Delete menu item, then click left on the selected object. If you delete a block all links connected to it are deleted as well. This menu item remains active until user choose an other one.

`Delete Region` :To delete a region in main Scicos window select first the Delete Region menu item, then click right on a corner of the desired region (in Scicos window or in a palette), drag to select de desired region, click to fix the selected region. If source diagram is a big region, selection may take a while.

`Flip` : To reverse the positions of the (regular) inputs and outputs of a block placed on its sides, click on the Flip menu item first and then on the selected block. This does not affect the order, nor the position of the input and output event ports which are numbered from left to right. A connected block cannot be flipped.

`Undo` : Click on the Undo menu item to undo the last edit operation.

`Simulation menu` :

`Setup` : In the main Scicos window, clicking on the Setup menu item invokes a dialog box that allows you to change integration parameters: absolute and relative error tolerances for the ode solver, the time tolerance (the smallest time interval for which the ode solver is used to update continuous states), and the maximum time increase realized by a single call to the ode solver.

`Compile` : This menu item need never be used since compilation is performed automatically, if necessary, before the beginning of every simulation (`Run` menu item).

Normally, a new compilation is not needed if only system parameters and internal states are modified. In some cases however modifications are not correctly updated and a manual compilation may be needed before a Restart or a Continue. Click on this menu item to compile the block diagram. Please report if you encounter such a case.

`Eval` : blocks dialogs answers can be defined using Scilab expressions. These expressions are evaluated immediately and they are also stored as character strings. Click on the Eval menu item to have them re-evaluated according to the new values of underlying Scilab variables defined by context for example.

`Run` : To start the simulation. If the system has already been simulated, a dialog box appears where you can choose to Continue, Restart or End the simulation. You may interrupt the simulation by clicking on the "stop" button, change any of the block parameters and continue or restart the simulation with the new values.

`Diagram menu` :

`Replot` : Scicos window stores the complete history of the editing session. Click on the Replot menu item to erase the history and replot the diagram or palette. Replot diagram before printing or exporting Scicos diagrams.

`New` : Clicking on the New menu item creates an empty diagram in the main Scicos window. If the previous content of the window is not saved, it will be lost.

`Purge` : Suppress deleted blocks out of Scicos data structure. This menu changes block indexing and implies compilation of the diagram before compilation.

`Rename` : Click on this menu item to change the diagram or palette's name. A dialog window will pop up.

`Make block` : Click on this menu item to save the Super Block as a new Scicos block. A Scilab function is generated and saved in <window_name>.sci file in the desired directory. <window_name> is the name of the Super Block appearing on top of the window. A dialog allows choosing the directory. This block may be added to a palette using `Edit/AddNew` menu item.

`Save` : Saves the block diagram in the current binary file selected by a previous call to `SaveAs` or `Load` menu item. If no current binary file, diagram is saved in the current directory as `<window_name>.cos`.

`Save As` : Saves the block diagram in a binary file. A file selection dialog will pop up.

`FSave` : Save the diagram in a formatted ascii file. A dialog box allows choosing the file name which must have a .cosf extension.

Formatted save is slower than regular save but has the advantage that the generated file is system independent (usefull for exchanging data on different computers.

`Load` : Loads an ascii or binary file containing a saved block diagram. A file selection dialog will pop up.

`Save as Palette` : select the Save as Palette menu item to save the block diagram as a palette in a binary file. A dialog box allows choosing the file which must have a .cos extension. The palette takes the name of the file (without the extension).

`.scilab` user file is updated.

`FSave as Palette` : select the FSave as Palette menu item to save the block diagram as a palette in an ascii formatted file. A dialog box allows choosing the file which must have a .cosf extension. The palette takes the name of the file (without the extension).

`Load as Palette` :select the Load menu item to load an ascii or binary file containing a saved block diagram as a palette. A dialog box allows user choosing the file.

`Exit` : Click on the Exit menu item to close current diagram. If current diagram is not a Super block Exit menu item leave Scicos and return to Scilab session. Save your diagram or palette before leaving.

`Object menu` :

`Set` :To change the parameters of a regular block or link, to open a super block, select first this menu item, click next on the desired object. A dialog or edition window appear that allows you to modify object

`Resize` : To change the size of a block , select first this menu item, click next on the desired block. A dialog appears that allows you to change the width and/or height of the block shape.

`Icon` : To change the icon of a block drawn by `standard_draw`, select first this menu item, click next on the desired block. A dialog appears that allows you to enter Scilab instructions used to draw the icon. These instructions may refer to `orig` and `sz` variables and more generaly to the block data structure named `o` in this context (see `scicos_block`). If Icon description selects colors for drawing, it is necessary to get it through `scs_color` function to have `Color` menu item work properly.

`Color` : To change the background of a block drawn by `standard_draw`, or color of a link select first this menu item, click next on the selected object. A color palette appears where user may select the block background color.

`Label` : To change or define the blocks label, select first this menu item, click next on the desired block. A dialog appears that allows you to enter the desired label. Labels may be used within blocks computational functions as an identification (see `getlabel` function).

`Miscellaneous menu` :

`Window` : Clicking on the Window menu item invokes a dialog box that allows you to change the editor window dimensions.

`Shift` :To shift the diagram to left, right, up or down, select this menu item, then click on the point you want to appear in the middle of the graphics window.

`Zoom in` : When you select this menu item the diagram is zoomed in by a factor of 10%

`Zoom out` : When you select this menu item the diagram is zoomed out by a factor of 10%

`Options` : Select this menu item to set display options.

`Help` : To get help on an object or menu menu items, select first Help menu item and then on the selected object or menu item.

`Calc:` : When you click on this menu item you switch Scilab to the pause mode (see the help on pause). In the Scilab main window and you may enter Scilab instructions to compute whatever you want. to go back to Scicos you need to enter ""return"" or "[...]=return(...)" Scilab instruction. ' If you use "[...]=return(...)" Scilab instruction take care not to modify Scicos variables such as "scs_m","scs_gc", "menus","datam",... ' If you have modified Scicos graphic window you may restore it using the Scicos "Replot" menu.

SEE ALSO : `scicos` 520

11.2 Blocks

11.2.1 ABSBLK_f — Scicos abs block

DIALOGUE PARAMETERS :

None.

DESCRIPTION :

This block realizes element-wise vector absolute value operation. This block has a single input and a single output port. Port dimension is determined by the context.

11.2.2 AFFICH_f — Scicos numerical display

DIALOGUE PARAMETERS :

`font` : integer, the selected font number (see xset)

`fontsize` : integer, the selected font size (set xset)

`color` : integer, the selected color for the text (see xset)

`Total numer of digits` : an integer greater than 3, the maximum number of digits used to represent the number (sign, integer part and rational part)

`rational part number of digits` : an integer greater than or equal 0, the number of digits used to represent the rational part

DESCRIPTION :

This block displays the value of its unique input inside the block (in the diagram) during simulation. The block must be located in the main Scicos window.

Warning: each time the block is moved user must click on it to set its parameters. The display position is then automatically updated.

SEE ALSO : `SCOPE_f` 540

11.2.3 ANDLOG_f — Scicos logical AND block

DIALOGUE PARAMETERS :

None.

DESCRIPTION :

This block, with two event inputs and a regular output, outputs +1 or -1 on its regular ouput depending on input events.

`+1` : When events are synchronously present on both event input ports

`-1` : When only one event is present.

SEE ALSO : `IFTHEL_f` 533

11.2.4 ANIMXY_f — Scicos 2D animated visualization block

DESCRIPTION :

This block realizes the visualization of the evolution of the two regular input signals by drawing the second input as a function of the first at instants of events on the event input port.

DIALOGUE PARAMETERS :

`Curve colors` : an integer. It is the color number (>=0) or marker type (<0) used to draw the evolution of the input port signal. See `xset()` for color (dash type) definitions.

`Line or mark size` : an integer.

`Output window number` : The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don't have the same window numbers (unless superposition of the curves is desired).

`Output window position` : a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

`Output window size` : a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.

`Xmin, Xmax` : Minimum and maximum values of the first input; used to set up the X-axis of the plot in the graphics window.

`Ymin, Ymax` : Minimum and maximum values of the second input; used to set up the Y-axis of the plot in the graphics window.

`Buffer size` : an integer. In order to minimize the number of graphics outputs, data may buffered.

REMARKS :

Output window number, Output window size, Output window position are only taken into account at the initialisation time of the simulation.

SEE ALSO : `SCOPE_f` 540, `EVENTSCOPE_f` 530, `SCOPXY_f` 540

11.2.5 BIGSOM_f — Scicos addition block

DIALOGUE PARAMETERS :

`Input signs` : a vector `sgn` of weights (generaly +1 or -1). The number of input signs fix the number of input ports.

DESCRIPTION :

This block realize weighted sum of the input vectors. The output is vector `k`th component is the sum of the `k`th components of each input ports weighted by `sgn(k)`.

SEE ALSO : `GAIN_f` 531, `SOM_f` 541

11.2.6 CLINDUMMY_f _ Scicos dummy continuous system with state

DESCRIPTION :
This block should be placed in any block diagram that contains a zero-crossing block but no continuous system with state. The reason for that is that it is the ode solver that find zero crossing surfaces.

SEE ALSO : `ZCROSS_f` 545

11.2.7 CLKIN_f _ Scicos Super Block event input port

DESCRIPTION :
This block must only be used inside Scicos Super Blocks to represent an event input port.
In a Super Block, the event input ports must be numbered from 1 to the number of event input ports.

DIALOGUE PARAMETERS :

`Port number` : an integer defining the port number.

SEE ALSO : `IN_f` 533, `OUT_f` 536, `CLKOUT_f` 525

11.2.8 CLKINV_f _ Scicos Super Block event input port

DESCRIPTION :
This block must only be used inside Scicos Super Blocks to represent an event input port.
In a Super Block, the event input ports must be numbered from 1 to the number of event input ports.

DIALOGUE PARAMETERS :

`Port number` : an integer defining the port number.

SEE ALSO : `IN_f` 533, `OUT_f` 536, `CLKOUTV_f` 526

11.2.9 CLKOUT_f _ Scicos Super Block event output port

DESCRIPTION :
This block must only be used inside Scicos Super Blocks to represent an event output port.
In a Super_Block, the event output ports must be numbered from 1 to the number of event output ports.

DIALOGUE PARAMETERS :

`Port number` : an integer giving the port number.

SEE ALSO : `IN_f` 533, `OUT_f` 536, `CLKIN_f` 525

11.2.10 CLKOUTV_f _ Scicos Super Block event output port

DESCRIPTION :
This block must only be used inside Scicos Super Blocks to represent an event output port.
In a Super_Block, the event output ports must be numbered from 1 to the number of event output ports.

DIALOGUE PARAMETERS :

`Port number` : an integer giving the port number.

SEE ALSO : `IN_f` 533, `OUT_f` 536, `CLKINV_f` 525

11.2.11 CLKSOM_f _ Scicos event addition block

DIALOGUE PARAMETERS :
None.

DESCRIPTION :
This block is an event addition block with up to three inputs. The output reproduces the events on all the input ports. Strictly speaking, CLKSOM is not a Scicos block because it is discarded at the compilation phase. The inputs and output of CLKSOM are synchronized.

11.2.12 CLKSOMV_f _ Scicos event addition block

DIALOGUE PARAMETERS :
None.

DESCRIPTION :
This block is an event addition block with up to three inputs. The output reproduces the events on all the input ports. Strictly speaking, CLKSOMV is not a Scicos block because it is discarded at the compilation phase. The inputs and output of CLKSOMV are synchronized.

11.2.13 CLKSPLIT_f _ Scicos event split block

DIALOGUE PARAMETERS :
None.

DESCRIPTION :
This block is an event split block with an input and two outputs. The outputs reproduces the event the input port on each output ports. Strictly speaking, CLKSPLIT is not a Scicos block because it is discarded at the compilation phase. This block is automatically created when creating a new link issued from a link.

The inputs and output of CLKSPLIT are synchronized.

11.2.14 CLOCK_f _ Scicos periodic event generator

DESCRIPTION :
This block is a Super Block constructed by feeding back the output of an event delay block into its input event port. The unique output of this block generates a regular train of events.

DIALOGUE PARAMETERS :

`Period` : scalar. One over the frequency of the clock. Period is the time that separates two output events.
`Init time` : scalar. Starting date. if negative the clock never starts.

SEE ALSO : `EVTDLY_f` 530

11.2.15 CLR_f ___ Scicos continuous-time linear system (SISO transfer function)

DIALOGUE PARAMETERS :

`Numerator` : a polynomial in s.
`Denominator` : a polynomial in s.

DESCRIPTION :
This block realizes a SISO linear system represented by its rational transfer function `Numerator/Denominator`. The rational function must be proper.

SEE ALSO : `CLSS_f` 527, `INTEGRAL_f` 533

11.2.16 CLSS_f ___ Scicos continuous-time linear state-space system

DESCRIPTION :
This block realizes a continuous-time linear state-space system.

```
xdot=A*x+B*u
y   =C*x+D*u
```

The system is defined by the (A,B,C,D) matrices and the initial state x0. The dimensions must be compatible.

DIALOGUE PARAMETERS :

`A` : square matrix. The A matrix
`B` : the `B` matrix, `[]` if system has no input
`C` : the `C` matrix , `[]` if system has no output
`D` : the `D` matrix, `[]` if system has no `D` term.
`x0` : vector. The initial state of the system.

SEE ALSO : `CLR_f` 527, `INTEGRAL_f` 533

11.2.17 CONST_f ___ Scicos constant value(s) generator

DIALOGUE PARAMETERS :

`constants` : a real vector. The vector size gives the size of the output port. The value `constants(i)` is assigned to the ith component of the output.

DESCRIPTION :
This block is a constant value(s) generator.

11.2.18 COSBLK_f ___ Scicos cosine block

DIALOGUE PARAMETERS :
None.
DESCRIPTION :
This block realizes vector cosine operation. `y(i)=cos(u(i))`. The port input and output port sizes are equal and determined by the context.
SEE ALSO : `SINBLK_f` 541, `GENSIN_f` 532

11.2.19 CURV_f ___ Scicos block, tabulated function of time

DIALOGUE PARAMETERS :
Tabulated function is entered using a graphics curve editor (see edit_curv in Scilab documentation)
DESCRIPTION :
This block defines a tabulated function of time. Between mesh points block performs a linear interpolation. Outside tabulation block outputs last tabulated value.
User may define the tabulation of the function using a curve editor.

11.2.20 DELAY_f ___ Scicos delay block

DIALOGUE PARAMETERS :

`Discretization time step` : positive scalar, delay discretization time step
`Register initial state` : register initial state vector. Dimension must be greater than or equal to 2

DESCRIPTION :
This block implements as a discretized delay. It is in fact a Scicos super block formed by a shift register and a clock.
value of the delay is given by the discretization time step multiplied by the number of states of the register minus one
SEE ALSO : `DELAYV_f` 528, `EVTDLY_f` 530, `REGISTER_f` 538

11.2.21 DELAYV_f ___ Scicos time varying delay block

DIALOGUE PARAMETERS :

`Number inputs` : size of the delayed vector (-1 not allowed)
`Register initial state` : register initial state vector. Dimension must be greater than or equal to 2
`Max delay` : Maximum delay that can be produced by this block

DESCRIPTION :
This block implements a time varying discretized delay. The value of the delay is given by the second input port. The delayed signal enters the first input port and leaves the unique output prot.
The first event output port must be connected to unique input event port if auto clocking is desired. But the input event port can also be driven by outside clock. In that case, the max delay is size of initial condition times the period of the incoming clock.
The second output event port generates an event if the second input goes above the maximum delay specified. This signal can be ignored. In that case the output will be delayed by max delay.
SEE ALSO : `DELAY_f` 528, `EVTDLY_f` 530, `REGISTER_f` 538

11.2.22 DEMUX_f ______ Scicos demultiplexer block

DIALOGUE PARAMETERS :

`number of output ports` : positive integer less than or equal to 8.

DESCRIPTION :
Given a vector valued input this block splits inputs over vector valued outputs. So `u=[y1;y2....;yn]`, where `yi` are numbered from top to bottom. Input and Output port sizes are determined by the context.

SEE ALSO : `MUX_f` 536

11.2.23 DLRADAPT_f ___ Scicos discrete-time linear adaptive system

DIALOGUE PARAMETERS :

`Vector of p mesh points` : a vector which defines `u2` mesh points. Numerator roots : a matrix, each line gives the roots of the numerator at the corresponding mesh point.
`Denominator roots` : a matrix, each line gives the roots of the denominator at the corresponding mesh point.
`gain` : a vector, each vector entry gives the transfer gain at the corresponding mesh point.
`past inputs` : a vector of initial value of past `degree(Numerator)` inputs
`past outputs` : a vector of initial value of past `degree(Denominator)` outputs

DESCRIPTION :
This block realizes a SISO linear system represented by its rational transfer function whose numerator and denominator roots are tabulated functions of the second block input. The rational function must be proper.

Roots are interpolated linearly between mesh points.

SEE ALSO : `DLSS_f` 529, `DLR_f` 529

11.2.24 DLR_f _ Scicos discrete-time linear system (transfer function)

DIALOGUE PARAMETERS :

`Numerator` : a polynomial in z.
`Denominator` : a polynomial in z.

DESCRIPTION :
This block realizes a SISO linear system represented by its rational transfer function (in the symbolic variable z). The rational function must be proper.

SEE ALSO : `DLSS_f` 529, `DLRADAPT_f` 529

11.2.25 DLSS_f ____ Scicos discrete-time linear state-space system

DESCRIPTION :
This block realizes a discrete-time linear state-space system. The system is defined by the (A,B,C,D) matrices and the initial state x0. The dimensions must be compatible. At the arrival of an input event on the unique input event port, the state is updated.

DIALOGUE PARAMETERS :

`A` : square matrix. The A matrix
`B` : the B matrix
`C` : the C matrix
`x0` : vector. The initial state of the system.

SEE ALSO : `DLR_f` 529, `INTEGRAL_f` 533, `CLSS_f` 527, `DLSS_f` 529

11.2.26 EVENTSCOPE_f ______ Scicos event visualization block

DESCRIPTION :
This block realizes the visualization of the input event signals.

DIALOGUE PARAMETERS :

`Number of event inputs` : an integer giving the number of event input ports colors : a vector of integers. The i-th element is the color number (>=0) or dash type (<0) used to draw the evolution of the i-th input port signal. See `xset` for color (dash type) definitions.
`Output window number` : The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don't have the same window numbers (unless superposition of the curves is desired). Output window position : a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.
`Output window size` : a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.
`Refresh period` : Maximum value on the X-axis (time). The plot is redrawn when time reaches a multiple of this value.

REMARKS :
Output window number, Output window size, Output window position are only taken into account at the initialisation time of the simulation.

SEE ALSO : `SCOPXY_f` 540, `SCOPE_f` 540, `ANIMXY_f` 524

11.2.27 EVTDLY_f ______ Scicos event delay block

DESCRIPTION :
One event is generated `Delay` after an event enters the unique input event port. Block may also generate an initial output event.

DIALOGUE PARAMETERS :

`Delay` : scalar. Time delay between input and output event.
`Auto-exec` : scalar. If `Auto-exec>=0` block initially generates an output event at date `Auto-exec`.

SEE ALSO : `CLOCK_f` 526

11.2.28 EVTGEN_f ______ Scicos event firing block

DESCRIPTION :
One event is generated on the unique output event port if `Event time` is larger than equal to zero, if not, no event is generated.

DIALOGUE PARAMETERS :

`Event time` : scalar. date of the initial event

SEE ALSO : `CLOCK_f` 526, `EVTDLY_f` 530

11.2.29 EXPBLK_f ______ Scicos a^u block

DIALOGUE PARAMETERS :

`a` : real positive scalar

DESCRIPTION :
This block realizes `y(i)=a^u(i)`. The input and output port sizes are determined by the compiler.

11.2.30 GAINBLK_f ______ Scicos gain block

DIALOGUE PARAMETERS :

`Gain` : a real matrix.

DESCRIPTION :
This block is a gain block. The output is the Gain times the regular input (vector). The dimensions of Gain determines the input (number of columns) and output (number of rows) port sizes.

11.2.31 GAIN_f ______ Scicos gain block

DIALOGUE PARAMETERS :

`Gain` : a real matrix.

DESCRIPTION :
This block is a gain block. The output is the Gain times the regular input (vector). The dimensions of Gain determines the input (number of columns) and output (number of rows) port sizes.

This block is obsolete. Use GAINBLK_f block instead of it

11.2.32 GENERAL_f ______ Scicos general zero crossing detector

DESCRIPTION :
Depending on the sign (just before the crossing) of the inputs and the input numbers of the inputs that have crossed zero, an event is programmed (or not) with a given delay, for each output. The number of combinations grows so fast that this becomes unusable for blocks having more than 2 or 3 inputs. For the moment this block is not documented.

DIALOGUE PARAMETERS :

`Size of regular input` : integer.
`Number of output events` : integer.
`the routing matrix` : matrix. number of rows is the number of output events. The columns correspond to each possible combination of signs and zero crossings of the inputs. The entries of the matrix give the delay for generating the output event (<0 no event is generated).

SEE ALSO : `NEGTOPOS_f` 536, `POSTONEG_f` 536, `ZCROSS_f` 545

11.2.33 GENERIC_f ______ Scicos generic interfacing function

DESCRIPTION :
This block can realize any type of block. The computational function must already be defined in Scilab, Fortran or C code.

DIALOGUE PARAMETERS :

`simulation function` : a character string, the name of the computational function
`function type` : a non negative integer, the type of the computational function
`input port sizes` : a vector of integers, size of regular input ports.
`output port sizes` : a vector of integers, size of regular output ports.
`input event port sizes` : a vector of ones, size of event input ports. The size of the vector gives the number of event input ports.
`output event port sizes` : a vector of ones, size of event output ports. The size of the vector gives the number of of event output ports.
`Initial continuous state` : a column vector.
`Initial discrete state` : a column vector.
`System type` : a string: c,d, z or l (CBB, DBB, zero crossing or synchro).
`Real parameter vector` : column vector. Any parameters used in the block can be defined here as a column vector.
`Integer parameter vector` : column vector. Any integer parameters used in the block can be defined here as a column vector.
`initial firing` : vector. Size of this vector corresponds to the number of event outputs. The value of the i-th entry specifies the time of the preprogrammed event firing on the i-th output event port. If less than zero, no event is preprogrammed.
`direct feedthrough` : character "y" or "n", specifies if block has a direct input to output feedthrough.
`Time dependance` : character "y" or "n", specifies if block output depends explicitly on time.

SEE ALSO : `scifunc_block` 545

11.2.34 GENSIN_f ______ Scicos sinusoid generator

DESCRIPTION :
This block is a sine wave generator: M*sin(F*t+P)

DIALOGUE PARAMETERS :

`Magnitude` : a scalar. The magnitude M.
`Frequency` : a scalar. The frequency F.
`Phase` : a scalar. The phase P.

SEE ALSO : `GENSQR_f` 532, `RAND_f` 537, `SAWTOOTH_f` 539

11.2.35 GENSQR_f ______ Scicos square wave generator

DESCRIPTION :
This block is a square wave generator: output takes values `-M` and `M`. Every time an event is received on the input event port, the output switches from `-M` to `M`, or `M to -M`.

DIALOGUE PARAMETERS :

`Amplitude` : a scalar `M`.

SEE ALSO : `GENSIN_f` 532, `SAWTOOTH_f` 539, `RAND_f` 537

11.2.36 HALT_f — Scicos Stop block

DIALOGUE PARAMETERS :

`State on halt` : scalar. A value to be placed in the state of the block. For debugging purposes this allows to distinguish between different halts.

DESCRIPTION :
This block has a unique input event port. Upon the arrival of an event, the simulation is stopped and the main Scicos window is activated. Simulation can be restarted or continued (Run button).

11.2.37 IFTHEL_f — Scicos if then else block

DIALOGUE PARAMETERS :
None.

DESCRIPTION :
One event is generated on one of the output event ports when an input event arrives. Depending on the sign of the regular input, the event is generated on the first or second output.

This is a synchro block, i.e., input and output event are synchronized.

11.2.38 IN_f — Scicos Super Block regular input port

DESCRIPTION :
This block must only be used inside Scicos Super Blocks to represent a regular input port. The input size is determined by the context.
In a Super Block, regular input ports must be numbered from 1 to the number of regular input ports.

DIALOGUE PARAMETERS :

`Port number` : an integer giving the port number.

SEE ALSO : `CLKIN_f` 525, `OUT_f` 536, `CLKOUT_f` 525

11.2.39 INTEGRAL_f — Scicos simple integrator

DESCRIPTION :
This block is an integrator. The output is the integral of the input.

DIALOGUE PARAMETERS :

`Initial state` : a scalar. The initial condition of the integrator.

SEE ALSO : `CLSS_f` 527, `CLR_f` 527

11.2.40 INTRP2BLK_f — Scicos 2D linear interpolation block

DIALOGUE PARAMETERS :

`X coord.` : an n-vector (strictly increasing)
`Y coord.` : an m-vector (strictly increasing)
`Z values` : an mxn matrix

DESCRIPTION :
The output of this block is a function of the inputs obtained by bilinear interpolation. This block has two scalar inputs and a single scalar output. The X(i) and Y(i) give respectively the X coordinate and the Y coordinate of the i-th data point to be interpolated and Z(Y(i),X(i)) its value.

11.2.41 INTRPLBLK_f — Scicos linear interpolation block

DIALOGUE PARAMETERS :

`X coord.` : a vector (strictly increasing)
`Y coord.` : a vector (same size as X coord)

DESCRIPTION :
The output of this block is a function of the input obtained by linear interpolation. This block has a single scalar input and a single scalar output port. The X coord. and Y coord. give respectively the X coordinate and the Y coordinate of the data points to be interpolated. X coord must be strictly increasing.

11.2.42 INVBLK_f — Scicos inversion block

DIALOGUE PARAMETERS :
None.

DESCRIPTION :
This block computes `y(i)=1/u(i)`. The input (output) size is determined by the context

11.2.43 LOGBLK_f — Scicos logarithm block

DIALOGUE PARAMETERS :

`a : real scalar greater than 1`

DESCRIPTION :
This block realizes `y(i)=log(u(i))/log(a)`. The input and output port sizes are determined by the context.

11.2.44 LOOKUP_f — Scicos Lookup table with graphical editor

DESCRIPTION :
This block realizes a non-linear function defined using a graphical editor.

11.2.45 MAX_f — Scicos max block

DIALOGUE PARAMETERS :
None.

DESCRIPTION :
The block outputs the maximum of the input vector: `y=max(u1,...un)`. The input vector size is determined by the compiler according to the connected blocks port sizes.

SEE ALSO : `MIN_f` 535

11.2.46 MCLOCK_f — Scicos 2 frequency event clock

DESCRIPTION :
This block is a Super Block constructed by feeding back the outputs of an MFCLCK block into its input event port. The two outputs of this block generate regular train of events, the frequency of the first input being equal to that of the second output divided by an integer n. The two outputs are synchronized (this is impossible for standard blocks; this is a Super Block).

DIALOGUE PARAMETERS :

`Basic period` : scalar. equals 1/f, f being the highest frequency.
`n` : an intger >1. the frequency of the first output event is f/n.

SEE ALSO : `MFCLCK_f` 535, `CLOCK_f` 526

11.2.47 MFCLCK_f Scicos basic block for frequency division of event clock

DESCRIPTION :
This block is used in the Super Block MCLOCK. The input event is directed once every n times to output 1 and the rest of the time to output 2. There is a delay of "Basic period" in the transmission of the event. If this period >0 then the second output is initially fired. It is not if this period=0. In the latter case, the input is driven by an event clock and in the former case, feedback can be used.

DIALOGUE PARAMETERS :

`Basic period` : positive scalar.
`n` : an integer greater than 1.

SEE ALSO : `MCLOCK_f` 535, `CLOCK_f` 526

11.2.48 MIN_f — Scicos min block

DIALOGUE PARAMETERS :
None.

DESCRIPTION :
The block outputs the minimum of the input vector: `y=min(u1,...un)`. The input vector size is determined by the compiler according to the connected blocks port sizes.

SEE ALSO : `MAX_f` 535

11.2.49 MUX_f — Scicos multiplexer block

DIALOGUE PARAMETERS :

`number of output ports` : integer greater than or equal to 1 and less than 8

DESCRIPTION :
Given `n` vector valued inputs this block merges inputs in an single output vector. So `y=[u1;u2....;un]`, where `ui` are numbered from top to bottom. Input and Output port sizes are determined by the context.

SEE ALSO : `MUX_f` 536

11.2.50 NEGTOPOS_f — Scicos negative to positive detector

DESCRIPTION :
An output event is generated when the unique input crosses zero with a positive slope.

SEE ALSO : `POSTONEG_f` 536, `ZCROSS_f` 545, `GENERAL_f` 531

11.2.51 OUT_f — Scicos Super Block regular output port

DIALOGUE PARAMETERS :

`Port number` : an integer giving the port number.

DESCRIPTION :
This block must only be used inside Scicos Super Blocks to represent a regular output port.
In a Super Block, regular output ports must be numbered from 1 to the number of regular output ports.
size of the output is determined by the compiler according to the connected blocks port sizes.

SEE ALSO : `CLKIN_f` 525, `IN_f` 533, `CLKOUT_f` 525

11.2.52 POSTONEG_f — Scicos positive to negative detector

DESCRIPTION :
An output event is generated when the unique input crosses zero with a negative slope.

SEE ALSO : `NEGTOPOS_f` 536, `ZCROSS_f` 545, `GENERAL_f` 531

11.2.53 POWBLK_f — Scicos u^a block

DIALOGUE PARAMETERS :

`a` : real scalar

DESCRIPTION :
This block realizes `y(i)=u(i)^a`. The input and output port sizes are determined by the compiler according to the connected blocks port sizes.

11.2.54 PROD_f ______ Scicos element wise product block

DESCRIPTION :
The output is the element wize product of the inputs.

11.2.55 QUANT_f ______ Scicos Quantization block

DIALOGUE PARAMETERS :

`Step` : scalar, Quantization step
`Quantization method` : scalar with possible values 1,2,3 or 4
`1` : Round method
`2` : Truncation method
`3` : Floor method
`4` : Ceil method

DESCRIPTION :
This block outputs the quantization of the input according to a choice of methods
for `Round` method
`y(i)=Step*(int(u(i)/Step+0.5)-0.5)` if `u(i)<0`.
`y(i)=Step*(int(u(i)/Step-0.5)+0.5)`. if `u(i)>=0`.
For truncation method
`y(i)=Step*(int(u(i)/Step+0.5))` if `u(i)<0`.
`y(i)=Step*(int(u(i)/Step-0.5))` if `u(i)>=0`.
For floor method
`y(i)=Step*(int(u(i)/Step+0.5))` .
For ceil method
`y(i)=Step*(int(u(i)/Step-0.5))` .

11.2.56 RAND_f ______ Scicos random wave generator

DESCRIPTION :
This block is a random wave generator: each output component takes piecewise constant random values. Every time an event is received on the input event port, the outputs take new independent random values.
output port size is given by the size of `A` and `B` vectors
DIALOGUE PARAMETERS :

`flag` : 0 or 1. 0 for uniform distribution on `[A,A+B]` and 1 for normal distribution `N(A,B*B)`.
`A` : scalar
`B` : scalar

SEE ALSO : `GENSIN_f` 532, `SAWTOOTH_f` 539, `GENSQR_f` 532

11.2.57 READC_f ______ Scicos "read from C binary file" block

DIALOGUE PARAMETERS :

`Time record Selection` : an empty matrix or a positive integer. If an integer `i` is given the `i`th element of the read record is assumed to be the date of the output event. If empty no output event exists.
`Output record selection` : a vector of positive integer. `[k1,..,kn]`,The `ki`th element of the read record gives the value of `i`th output.
`Input file name` : a character string defining the path of the file
`Input Format` : a character string defining the format to use
`"l","s","ul","us","d","f","c","uc"` : for reading respectively long, ashort, unsigned long, unsigned short, double, float, char and unsigned char. If required by the `swap mode`, the bytes which are read are automatically swapped if necessary (by checking little-endian status) in order to produce machine independent binary files (in little-endian mode).
`"ull","uls","ubl","ubs",` : can be used for reading respectively unsigned little-endian long or short and unsigned big-endian long or short.
`"dx","fx","lx","sx"` : with `x=b` or `x=l` can be used for reading double, float, long or short as big or little endian.
`Record size` : The file is supposed to be formed by a sequence of data with same format. these data are organized in a sequence of record each of them containing `Record size` data.
`Buffer size` : To improve efficiency it is possible to buffer the input data. read on the file is only done after each `Buffer size` call to the block.
`Initial record index` : a scalar. This fixes the first record of the file to use.
`Swap mode` : With `Swap mode=1` 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. If `Swap mode=0` then automatic bytes swap is disabled.

DESCRIPTION :
This block allows user to read datas in a C file. `Output record selection` and `Time record Selection` allows the user to select data among file records.
Each call to the block advance one record in the file.

SEE ALSO : `RFILE_f` 539, `mget` 64

11.2.58 REGISTER_f ______ Scicos shift register block

DESCRIPTION :
This block realizes a shift register. At every input event, the register is shifted one step.
DIALOGUE PARAMETERS :

`Initial condition` : a column vector. It contains the initial state of the register.

SEE ALSO : `DELAY_f` 528, `DELAYV_f` 528, `EVTDLY_f` 530

11.2.59 RELAY_f ______ Scicos relay block

DIALOGUE PARAMETERS :

`number of inputs` : a scalar. Number of regular and event inputs.
`initial connected input` : an integer. It must be between 1 and the number of inputs.

DESCRIPTION :
This block routes one of the regular inputs to the unique regular output. the choice of which input is to be routed is done, initially by the "initial connected input" parameter. Then, every time an input event arrives on the i-th input event port, the i-th regular input port is routed to the regular output.

11.2.60 RFILE_f Scicos "read from file" block

DIALOGUE PARAMETERS :

`Time record Selection` : an empty matrix or a positive integer. If an integer `i` is given the `i`th element of the read record is assumed to be the date of the output event. If empty no output event exists.

`Output record selection` : a vector of positive integer. `[k1,..,kn]`,The `ki`th element of the read record gives the value of `i`th output.

`Input file name` : a character string defining the path of the file

`Input Format` : a character string defining the Fortran format to use or nothing for an unformatted (binary) write

`Buffer size` : To improve efficiency it is possible to buffer the input data. read on the file is only done after each `Buffer size` call to the block.

`size of output` : a scalar. This fixes the number of "value" read.

DESCRIPTION :

This block allows user to read datas in a file, in formatted or binary mode. `Output record selection` and `Time record Selection` allows the user to select data among file records.

Each call to the block advance one record in the file.

SEE ALSO : `WFILE_f` 544

11.2.61 SAMPLEHOLD_f Scicos Sample and hold block

DIALOGUE PARAMETERS :

None.

DESCRIPTION :

Each time an input event is received block copy its input on the output and hold it until input event. For periodic Sample and hold, event input must be generated by a `Clock`.

SEE ALSO : `DELAY_f` 528, `CLOCK_f` 526

11.2.62 SAT_f Scicos Saturation block

DESCRIPTION :

This block realizes the non-linear function: saturation.

DIALOGUE PARAMETERS :

`Min` : a scalar. Lower saturation bound

`Max` : a scalar. Upper saturation bound

`Slope` : a scalar. The slope of the line going through the origin and describing the behaviour of the function around zero.

SEE ALSO : `LOOKUP_f` 534

11.2.63 SAWTOOTH_f Scicos sawtooth wave generator

DESCRIPTION :

This block is a sawtooth wave generator: output is (t-t_i) from ti to t_(i+1) where t_i and t_(i+1) denote the times of two successive input events.

DIALOGUE PARAMETERS :

None.

SEE ALSO : `GENSIN_f` 532, `GENSQR_f` 532, `RAND_f` 537

11.2.64 SCOPE_f Scicos visualization block

DESCRIPTION :

This block realizes the visualization of the evolution of the signals on the standard input port(s) at instants of events on the event input port.

DIALOGUE PARAMETERS :

`Curve colors` : a vector of integers. The i-th element is the color number (>0) or dash type (<0) used to draw the evolution of the i-th input port signal. See `plot2d` for color (dash type) definitions.

`Output window number` : The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don't have the same window numbers (unless superposition of the curves is desired).

`Output window position` : a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

`Output window size` : a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.

`Ymin, Ymax` : Minimum and maximum values of the input; used to set up the Y-axis of the plot in the graphics window.

`Refresh period` : Maximum value on the X-axis (time). The plot is redrawn when time reaches a multiple of this value.

`Buffer size` : To improve efficiency it is possible to buffer the input data. The drawing is only done after each `Buffer size` call to the block.

`Accept herited events` : if 0 `SCOPE_f` draws a new point only when an event occurs on its event input port. if 1 `SCOPE_f` draws a new point when an event occurs on its event input port and when it's regular input changes due to an event on an other upstrem block (herited events).

REMARKS :

Output window number, Output window size, Output window position are only taken into account at the initialisation time of the simulation.

SEE ALSO : `SCOPXY_f` 540, `EVENTSCOPE_f` 530, `ANIMXY_f` 524

11.2.65 SCOPXY_f Scicos visualization block

DESCRIPTION :

This block realizes the visualization of the evolution of the two regular input signals by drawing the second input as a function of the first at instants of events on the event input port.

DIALOGUE PARAMETERS :

`Curve colors` : an integer. It is the color number (>0) or dash type (<0) used to draw the evolution of the input port signal. See `plot2d` for color (dash type) definitions.

`Line or mark size` : an integer.

`Output window number` : The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don't have the same window numbers (unless superposition of the curves is desired).
`Output window position` : a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.
`Output window size` : a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.
`Xmin, Xmax` : Minimum and maximum values of the first input; used to set up the X-axis of the plot in the graphics window.
`Ymin, Ymax` : Minimum and maximum values of the second input; used to set up the Y-axis of the plot in the graphics window.
`Buffer size` : To improve efficiency it is possible to buffer the input data. The drawing is only done after each `Buffer size` call to the block.

REMARKS :
Output window number, Output window size, Output window position are only taken into account at the initialisation time of the simulation.

SEE ALSO : `SCOPE_f` 540, `EVENTSCOPE_f` 530, `ANIMXY_f` 524

11.2.66 SELECT_f — Scicos selector block

DIALOGUE PARAMETERS :

`number of inputs` : a scalar. Number of regular and event inputs.
`initial connected input` : an integer. It must be between 1 and the number of inputs.

DESCRIPTION :
This block routes one of the regular inputs to the unique regular output. the choice of which input is to be routed is done, initially by the "initial connected input" parameter. Then, every time the block is activated through its i-th input activation port, the i-th regular input value port is put to the regular output.

11.2.67 SINBLK_f — Scicos sine block

DIALOGUE PARAMETERS :
None.

DESCRIPTION :
This block realizes vector sine operation. `y(i)=sin(u(i))`. The input and output port sizes are equal and determined by the context.

11.2.68 SOM_f — Scicos addition block

DIALOGUE PARAMETERS :

`Input signs` : a (1x3) vector of +1 and -1. If -1, the corresponding input is multiplied by -1 before addition.

DESCRIPTION :
This block is a sum. The output is the element-wise sum of the inputs.

Input ports are located at up, left or right and down position. You must specify 3 gain numbers but if only two links are connected only the first values are used, ports are numbered anti-clock wise.

SEE ALSO : `GAIN_f` 531

11.2.69 SPLIT_f — Scicos regular split block

DIALOGUE PARAMETERS :
None.

DESCRIPTION :
This block is a regular split block with an input and two outputs. The outputs reproduces the input port on each output ports. Strictly speaking, SPLIT is not a Scicos block because it is discarded at the compilation phase. This block is automatically created when creating a new link issued from a link.

Port sizes are determined by the context.

11.2.70 STOP_f — Scicos Stop block

DIALOGUE PARAMETERS :

`State on halt` : scalar. A value to be placed in the state of the block. For debugging purposes this allows to distinguish between different halts.

DESCRIPTION :
This block has a unique input event port. Upon the arrival of an event, the simulation is stopped and the main Scicos window is activated. Simulation can be restarted or continued (Run button).

11.2.71 SUPER_f — Scicos Super block

DESCRIPTION :
This block opens up a new Scicos window for editing a new block diagram. This diagram describes the internal functions of the super block.
Super block inputs and outputs (regular or event) are designated by special (input or output) blocks.
Regular input blocks must be numbered from 1 to the number of regular input ports. Regular input ports of the super block are numbered from the top of the block shape to the bottom.

Regular output portss must be numbered from 1 to the number of regular output ports. Regular output ports of the super block are numbered from the top of the block shape to the bottom.

Event input blocks must be numbered from 1 to the number of event input ports. Event input ports of the super block are numbered from the left of the block shape to the right.
Event output ports must be numbered from 1 to the number of event output ports. Event output ports of the super block are numbered from the left of the block shape to the right.

SEE ALSO : `CLKIN_f` 525, `OUT_f` 536, `CLKOUT_f` 525, `IN_f` 533

11.2.72 TANBLK_f ______ Scicos tan block

DIALOGUE PARAMETERS :
None.

DESCRIPTION :
This block realizes vector tangent operation. input (output) port size is determined by the compiler.

SEE ALSO : `SINBLK_f` 541

11.2.73 TCLSS_f ______ Scicos jump continuous-time linear state-space system

DESCRIPTION :
This block realizes a continuous-time linear state-space system with the possibility of jumps in the state. The number of inputs to this block is two. The first input is the regular input of the linear system, the second carries the new value of the state which is copied into the state when an event arrives at the unique event input port of this block. That means the state of the system jumps to the value present on the second input (of size equal to that of the state). The system is defined by the (A,B,C,D) matrices and the initial state x0. The dimensions must be compatible. The sizes of inputs and outputs are adjusted automatically.

DIALOGUE PARAMETERS :

`A` : square matrix. The A matrix
`B` : the B matrix
`C` : the C matrix
`D` : the D matrix
`x0` : vector. The initial state of the system.

SEE ALSO : `CLSS_f` 527, `CLR_f` 527

11.2.74 TEXT_f ______ Scicos text drawing block

DIALOGUE PARAMETERS :

`txt` : a character string, Text to be displayed
`font` : a positive integer less than 6, number of selected font (see xset)
`siz` : a positive integer, selected font size (see xset)

DESCRIPTION :
This special block is only use to add text at any point of the diagram window. It has no effect on the simulation.

11.2.75 TIME_f ______ Scicos time generator

DIALOGUE PARAMETERS :
None.

DESCRIPTION :
This block is a time generator. The unique regular output is the current time.

11.2.76 TRASH_f ______ Scicos Trash block

DIALOGUE PARAMETERS :
None

DESCRIPTION :
This block does nothing. It simply allows to safely connect the outputs of other blocks which should be ignored. Useful for sinking outputs of no interest. The input size is determined by the compiler.

11.2.77 WFILE_f ______ Scicos "write to file" block

DIALOGUE PARAMETERS :

`input size` : a scalar. This fixes the input size
`Output file name` : a character string defining the path of the file
`Output Format` : a character string defining the Fortran format to use or nothing for an unformatted (binary) write
`Buffer size` : To improve efficiency it is possible to buffer the input data. write on the file is only done after each `Buffer size` calls to the block.

DESCRIPTION :
This block allows user to save data in a file, in formatted and binary mode. Each call to the block corresponds to a record in the file. Each record has the following form: `[t,V1,...,Vn]` where `t` is the value of time when block is called and `Vi` is the ith input value

SEE ALSO : `RFILE_f` 539

11.2.78 WRITEC_f ______ Scicos "write to C binary file" block

DIALOGUE PARAMETERS :

`Input size` : a scalar, the size of the input
`Output file name` : a character string defining the path of the file
`Output Format` : a character string defining the format to use
`"l","s","ul","us","d","f","c","uc"` : for reading respectively long, ashort, unsigned long, unsigned short, double, float, char and unsigned char. If required by the `swap mode`, the bytes which are read are automatically swapped if necessary (by checking little-endian status) in order to produce machine independent binary files (in little-endian mode).
`"ull","uls","ubl","ubs",` : can be used for reading respectively unsigned little-endian long or short and unsigned big-endian long or short.
`"dx","fx","lx","sx"` : with `x=b` or `x=l` can be used for reading double, float, long or short as big or little endian.
`Buffer size` : To improve efficiency it is possible to buffer the input data. read on the file is only done after each `Buffer size` call to the block.
`Swap mode` : With `Swap mode=1` 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. If `Swap mode=0` then automatic bytes swap is disabled.

DESCRIPTION :
This block allows user to write datas in a C binary file.

SEE ALSO : `READC_f` 537, `mput` 69

11.2.79 ZCROSS_f — Scicos zero crossing detector

DESCRIPTION :
An output event is generated when all inputs (if more than one) cross zero simultaneously.
DIALOGUE PARAMETERS :

`Number of inputs` : a positive integer.

SEE ALSO : `POSTONEG_f` 536, `GENERAL_f` 531

11.2.80 scifunc_block — Scicos block defined interactively

DESCRIPTION :
This block can realize any type of Scicos block. The function of the block is defined interactively using dialogue boxes and in Scilab language. During simulation, these instructions are interpreted by Scilab; the simulation of diagrams that include these types of blocks is slower. For more information see Scicos reference manual.
DIALOGUE PARAMETERS :

`number of inputs` : a scalar. Number of regular input ports
`number of outputs` : a scalar. Number of regular output ports
`number of input events` : a scalar. Number of input event ports
`number of output events` : a scalar. Number of output event ports
`Initial continuous state` : a column vector.
`Initial discrete state` : a column vector.
`System type` : a string: c or d (CBB or DBB, other types are not supported).
`System parameter` : column vector. Any parameters used in the block can be defined here a column vector.
`initial firing` : vector. Size of this vector corresponds to the number of event outputs. The value of the i-th entry specifies the time of the preprogrammed event firing on the i-th output event port. If less than zero, no event is preprogrammed.
`Instructions` : other dialogues are opened consecutively where used may input Scilab code associated with the computations needed (block initialization, outputs, continuous and discrete state, output events date, block ending),

SEE ALSO : `GENERIC_f` 532

11.3 Data Structures

11.3.1 scicos_main — Scicos editor main data structure

DEFINITION :

```
scs_m=list(params,o_1,....,o_n)
```

PARAMETERS :

`params` : Scilab list, `params=list(wpar,title,tol,tf,context,void,options,void,void,doc))`
`wpar` : viewing parameters: `[w,h,Xshift,Yshift]`

`w` : real scalar,Scicos editor window width
`h` : real scalar,Scicos editor window height
`Xshift` : real scalar, diagram drawing x offset within Scicos editor window
`Yshift` : real scalar, diagram drawing y offset within Scicos editor window
`title` : character string, diagram title and default name of save file name
`tol` : 1 x 4 vector `[atol,rtol,ttol,maxt]`, where `atol`, `rtol` are respectively absolute and relative tolerances for the ode solver, `ttol` is the minimal distance between to different events time and `maxt` is maximum integration time interval for a single call to the ode solver.
`tf` : real scalar, final time for simulation.
`context` : vector of character strings, Scilab instructions used to define Scilab variables used in block definitions as symbolic parameters.
`void` : unused fields
`options` : `list(With3D,Color3D)`
`With3D` : boolean, true for 3D shape blocks
`Color3D` : vector with three entries `[R,G,B]`. defines the color of 3D shape
`doc` : user defined diagram documentation structure, default value is `list()`
`o_i` : block or link or deleted object data structure.

See `scicos_block` and `scicos_link`).

Deleted object data structure is marked `list('Deleted')`.
`scs_m` : main Scicos structure

DESCRIPTION :
Scicos editor uses and modifies the Scicos editor main data structure to keep all information relative to the edited diagram. Scicos compiler uses it as a input.

SEE ALSO : `scicos` 520, `scicos_block` 546, `scicos_link` 548

11.3.2 scicos_block — Scicos block data structure

DEFINITION :

```
blk=list('Block',graphics,model,void,gui)
```

PARAMETERS :

`"Block"` : keyword used to define list as a Scicos block representation
`graphics` : Scilab list, graphic properties data structure
`model` : Scilab list, system properties data structure.
`void` : unused, reserved for future use.
`gui` : character string, the name of the graphic user interface function (generally written in Scilab) associated with the block.
`blk` : Scilab list, Scicos block data structure

DESCRIPTION :
Scicos editor creates and uses for each block a data structure containing all information relative to the graphic interface and simulation part of the block. Each of them are stored in the Scicos editor main data structure. Index of these in Scicos editor main data structure is given by the creation order.

For Super blocks `model(8)` contains a data structure similar to the `scicos_main` data structure.

SEE ALSO : `scicos_graphics` 547, `scicos_model` 547

11.3.3 scicos_graphics — Scicos block graphics data structure

DEFINITION :

`graphics=list(orig,sz,flip,exprs,pin,pout,pein,peout,gr_i)`

PARAMETERS :

`orig` : 2 x 1 vector, the coordinate of down-left point of the block shape.
`sz` : vector `[w,h]`, where `w` is the width and `h` the height of the block shape.
`flip` : boolean, the block orientation. if true the input ports are on the left of the box and output ports are on the right. if false the input ports are on the right of the box and output ports are on the left.
`exprs` : column vector of strings, contains expressions answered by the user at block set time.
`pin` : column vector of integers. If `pin(k)<>0` then `k`th input port is connected to the `pin(k)<>0` block, else the port is unconnected. If no input port exist `pin==[]`.
`pout` : column vector of integers. If `pout(k)<>0` then `k`th output port is connected to the `pout(k)<>0` block, else the port is unconnected. If no output port exist `pout==[]`.
`pein` : column vector of ones. If `pein(k)<>0` then `k`th event input port is connected to the `pein(k)<>0` block, else the port is unconnected. If no event input port exist `pein==[]`.
`peout` : column vector of integers. If `peout(k)<>0` then `k`th event output port is connected to the `epout(k)<>0` block, else the port is unconnected. If no event output port exist `peout==[]`.
`gr_i` : column vector of strings, contains Scilab instructions used to customize the block graphical aspect. This field may be set with `"Icon"` sub_menu.
`graphics` : Scilab list, Scicos block graphics data structure.

DESCRIPTION :
Scicos block graphics data structure contains all information relative to graphical display of the block and to user dialogue. Fields may be fixed by block definition or set as a result of user dialogue or connections.

SEE ALSO : `scicos` 520, `scicos_model` 547, `scicos_main` 545

11.3.4 scicos_model — Scicos block functionality data structure

DEFINITION :

```
model=list(sim,in,out,evtin,evtout,state,dstate,..
      rpar,ipar,blocktype,firing,dep_ut,label,import,ID)
```

PARAMETERS :

`sim` : list(fun,typ) or fun. In the latest case typ is supposed to be 0.
`fun` : character string, the name of the block simulation function (a linked C or Fortran procedure or a Scilab function).
`typ` : integer, calling sequence type of simulation function (see documentation for more precision).
`in` : column vector of integers, input port sizes indexed from top to bottom of the block. If no input port exist `in==[]`.
`out` : column vector of integers, output port sizes indexed from top to bottom of the block. If no output port exist `in==[]`.
`evtin` : column vector of ones, the size of `evtin` gives the number of event input ports. If no event input port exists `evtin` must be equal to `[]`.
`evtout` : column vector of ones, the size of `evtout` gives the number of event output ports. If no event output port exists `evtout` must be equal to `[]`.

`state` : column vector, the initial continuous state of the block. Must be `[]` if no continuous state.
`dstate` : column vector, the initial discrete state of the block. Must be `[]` if no discrete state.
`rpar` : column vector, the vector of floating point block parameters. Must be `[]` if no floating point parameters.
`ipar` : column vector, the vector of integer block parameters. Must be `[]` if no integer parameters.
`blocktype` : a character with possible values:
: 'c' block output depend continuously of the time.
: 'd' block output changes only on input events.
: 'z' zero crossing block
: 'l' logical block
`firing` : a vector whose size is equal to the size of `evtout`> It contains output initial event dates (Events generated before any input event arises). Negative values stands for no initial event on the corresponding port.
`dep_ut` : 1x 2 vector of boolean `[dep_u, dep_t]`, `dep_u` must be true if output depends continuously of the input, `dep_t` must be true if output depends continuously of the time.
`label` : a character string, used as a label
`import` : Unused.
`ID` : a character string, used as an identifier.
`model` : Scilab list, Scicos block model data structure.

DESCRIPTION :
Scicos block model data structure contains all information relative to the simulation functionality of the block. Fields may be fixed by block definition or set.

If block is a super block, the fields `state`,`dstate`,`ipar`,`blocktype`,`firing`, `dep_ut`, are unused.
The `rpar` field contains a data structure similar to the `scicos_main` data structure.

SEE ALSO : `scicos` 520, `scicos_model` 547, `scicos_main` 545

11.3.5 scicos_link — Scicos link data structure

DEFINITION :

`lnk=list('Link',xx,yy,'drawlink',id,[0,0],ct,from,to)`

PARAMETERS :

`"Link"` : keyword used to define list as a Scicos link representation
`xx` : vector of x coordinates of the link path.
`yy` : vector of y coordinates of the link path.
`id` : Character string, the link id
`ct` : 2 x 1 vector, `[color,typ]` where `color` defines the color used for the link drawing and `typ` defines its type (0 for regular link ,1 for event link).
`from` : 2 x 1 vector, `[block,port]` where `block` is the index of the block at the origin of the link and `port` is the index of the port.
`to` : 2 x 1 vector, `[block,port]` where `block` is the index of the block at the end of the link and `port` is the index of the port.

DESCRIPTION :
Scicos editor creates and uses for each link a data structure containing all information relative to the graphic interface and interconnection information. Each of them are stored in the Scicos editor main data structure. Index of these in Scicos editor main data structure is given by the creation order.

SEE ALSO : `scicos` 520, `scicos_main` 545, `scicos_graphics` 547, `scicos_model` 547

11.3.6 scicos_cpr — Scicos compiled diagram data structure

DEFINITION :

`cpr=list(state,sim,cor,corinv)`

PARAMETERS :

`state` : Scilab `tlist` contains initial state.
`state('x')` : continuous state vector.
`state('z')` : discrete state vector.
`state('tevts')` : vector of event dates
`state('evtspt')` : vector of event pointers
`state('pointi')` : pointer to next event state('npoint') : not used yet state('outtb') : vector of inputs/outputs initial values.
`sim` : Scilab `tlist`. Usually generated by Scicos `Compile` menu. Some useful entries are:
`sim('rpar')` : vector of blocks' floating point parameters
`sim('rpptr')` : (nblk+1) x 1 vector of integers,

`sim('rpar')(rpptr(i):(rpptr(i+1)-1))` is the vector of floating point parameters of the `i`th block.

`sim('ipar')` : vector of blocks' integer parameters
`sim('ipptr')` : (nblk+1) x 1 vector of integers,

`sim('ipar')(ipptr(i):(ipptr(i+1)-1))` is the vector of integer parameters of the `i`th block.

`sim('funs')` : vector of strings containing the names of each block simulation function
`sim('xptr')` : (nblk+1) x 1 vector of integers,

`state('x')(xptr(i):(xptr(i+1)-1))` is the continuous state vector of the `i`th block.

`sim('zptr')` : (nblk+1) x 1 vector of integers,

`state('z')(zptr(i):(zptr(i+1)-1))` is the discrete state vector of the `i`th block.

`sim('inpptr')` : (nblk+1) x 1 vector of integers,

`inpptr(i+1)-inpptr(i)` gives the number of input ports. `inpptr(i)`th points to the beginning of `i`th block inputs within the indirection table `inplnk`.

`sim('inplnk')` : nblink x 1 vector of integers,

`inplnk(inpptr(i)-1+j)` is the index of the link connected to the `j`th input port of the `i`th block. where `j` goes from `1` to `inpptr(i+1)-inpptr(i))`.

`sim('outptr')` : (nblk+1) x 1 vector of integers,

`outptr(i+1)-outptr(i)` gives the number of output ports. `outptr(i)`th points to the beginning of `i`th block outputs within the indirection table `outlnk`.

`sim('outlnk')` : nblink x 1 vector of integers,

`outlnk(outptr(i)-1+j)` is the index of the link connected to the `j`th output port of the `i`th block. where `j` goes from `1` to `outptr(i+1)-outptr(i))`.

`sim('lnkptr')` : (nblink+1) x 1 vector of integers,

`k`th entry points to the beginning of region within `outtb` dedicated to link indexed `k`.

`sim('funs')` : vector of strings containing the names of each block simulation function
`sim('funtyp')` : vector of block block types.
`cor` : is a list with same recursive structure as scs_m each leaf contains the index of associated block in `cpr` data structure.
`corinv` : corinv(i) is the path of `i` th block defined in `cpr` data structure in the `scs_m` data structure.

DESCRIPTION :

Scicos compiled diagram data structure contains all information needed to simulate the system (see `scicosim`).

SEE ALSO : `scicos` 520, `scicos_model` 547, `scicos_main` 545, `scicosim` 551

11.4 Useful Functions

11.4.1 standard_define — Scicos block initial definition function

CALLING SEQUENCE :

`o=standard_define(sz,model,dlg,gr_i)`

PARAMETERS :

`o` : Scicos block data structure (see scicos_block)
`sz` : 2 vector, giving the initial block width and height
`model` : initial model data structure definition (see scicos_model)
`dlg` : vector of character strings,initial parameters expressions
`gr_i` : vector of character strings, initial icon definition instructions

DESCRIPTION :

This function creates the initial block data structure given the initial size `sz`, this initial model definition `model`, the initial parameters expressions `dlg` and initial icon definition instructions `gr_i`

SEE ALSO : `scicos_model` 547

11.4.2 standard_draw — Scicos block drawing function

CALLING SEQUENCE :

`standard_draw(o)`

PARAMETERS :

`o` : Scicos block data structure (see scicos_block)

DESCRIPTION :

`standard_draw` is the Scilab function used to display standard blocks in interfacing functions.
It draws a block with a rectangular shape with any number of regular or event input respectively on the left and right faces of the block (if not flipped), event input or output respectively on the top and bottom faces of the block. Number of ports, size, origin, orientation, background color, icon of the block are taken from the block data structure `o`.

SEE ALSO : `scicos_block` 546

11.4.3 standard_input — get Scicos block input port positions

CALLING SEQUENCE :

`[x,y,typ]=standard_input(o)`

PARAMETERS :

`o` : Scicos block data structure (see scicos_block)
`x` : vector of x coordinates of the block regular and event input ports
`y` : vector of y coordinates of the block regular and event output ports
`typ` : vector of input ports types (+1 : regular port; -1:event port)

DESCRIPTION :

`standard_input` is the Scilab function used to get standard blocks input port position and types in interfacing functions.
Port positions are computed, each time they are required, as a function of block dimensions.

SEE ALSO : `scicos_block` 546

11.4.4 standard_origin — Scicos block origin function

CALLING SEQUENCE :

```
[x,y]=standard_draw(o)
```

PARAMETERS :

- o : Scicos block data structure (see scicos_block)
- x : x coordinate of the block origin (bottom left corner)
- y : y coordinate of the block origin (bottom left corner)

DESCRIPTION :

standard_origin is the Scilab function used to get standard blocks position in interfacing functions.

SEE ALSO : scicos_block 546

11.4.5 standard_output — get Scicos block output port positions

CALLING SEQUENCE :

```
[x,y,typ]=standard_output(o)
```

PARAMETERS :

- o : Scicos block data structure (see scicos_block)
- x : vector of x coordinates of the block regular and event output ports
- y : vector of y coordinates of the block regular and event output ports
- typ : vector of output ports types (+1 : regular port; -1:event port)

DESCRIPTION :

standard_output is the Scilab function used to get standard blocks output port position and types in interfacing functions.
Port positions are computed, each time they are required, as a function of block dimensions.

SEE ALSO : scicos_block 546

11.4.6 scicosim — Scicos simulation function

CALLING SEQUENCE :

```
[state,t]=scicosim(state,0,tf,sim,'start' [,tol])
[state,t]=scicosim(state,tcur,tf,sim,'run' [,tol])
[state,t]=scicosim(state,tcur,tf,sim,'finish' [,tol])
```

PARAMETERS :

- state : Scilab tlist contains scicosim initial state. Usually generated by Scicos Compile or Run menus (see scicos_cpr for more details).
- tcur : initial simulation time
- tf : final simulation time (Unused with options 'start' and 'finish'
- sim : Scilab tlist. Usually generated by Scicos Compile menu (see scicos_cpr for more details).

- tol : 4 vector [atol,rtol,ttol,deltat] where atol, rtol are respectively the absolute and relative tolerances for ode solver (see ode), ttol is the precision on event dates. deltat is maximum integration interval for each call to ode solver.
- t : final reached time

DESCRIPTION :

Simulator for Scicos compiled diagram. Usually scicosim is called by scicos to perform simulation of a diagram.

But scicosim may also be called outside Scicos. Typical usage in such a case may be:

1. Use Scicos to define a block diagram, compile it.
2. Save the compiled diagram using Save,SaveAs Scicos menus .
3. In Scilab, load saved file using load function. You get variables scicos_ver, scs_m, cpr

scs_m is the diagram Scicos main data structure.
cpr is the data structure list(state,sim,cor,corinv) if the diagram had been compiled before saved, else cpr=list()

4. Extract state, sim out of cpr
5. Execute [state,t]=scicosim(state,0,tf,sim,'start' [,tolerances]) for initialisation.
6. Execute [state,t]=scicosim(state,0,tf,sim,'run' [,tolerances]) for simulation from 0 to tf. Many successives such calls may be performed changing initial and final time.
7. Execute [state,t]=scicosim(state,0,tf,sim,'finish' [,tolerances]) at the very end of the simulation to close files,...

For advanced user it is possible to "manually" change some parameters or state values

SEE ALSO : scicos 520, scicos_cpr 549

11.4.7 curblock — get current block index in a Scicos simulation function

CALLING SEQUENCE :

```
k=curblock()
```

PARAMETERS :

- k : integer, index of the block corresponding to the Scilab simulation function where this function is called.

DESCRIPTION :

During simulation it may be interesting to get the index of the current block to trace execution, to get its label, to animate the block icon according to simulation...
For block with a computational function written in Scilab, Scilab primitive function curblock() allows to get the index of the current block in the compiled data structure.
To obtain path to the block in the Scicos main structure user may uses the corinv table (see scicos_cpr).
For block with a computational function written in C user may uses the C function k=C2F(getcurblock)(). Where C2F is the C compilation macro defined in <SCIDIR>/routines/machine.h
For block with a computational function written in Fortran user may uses the integer function k=getcurblock().

SEE ALSO : getblocklabel 553, getscicosvars 553, setscicosvars 554, scicos_cpr 549, scicos_main 545

11.4.8 getblocklabel — get label of a Scicos block at running time

CALLING SEQUENCE :

```
label=getblocklabel()
label=getblocklabel(k)
```

PARAMETERS :

`k` : integer, index of the block. if `k` is omitted `k`is supposed to be equal to `curblock()`.
`label` : a character string, The label of `k`th block (see `Label` button in `Block` menu.

DESCRIPTION :

For display or debug purpose it may be usefull to give label to particular blocks of a diagram. This may be done using Scicos editor (`Label` button in `Block` menu). During simulation, value of these labels may be obtained in any Scilab block with `getblocklabel` Scilab primitive function. For C or fortran computational functions, user may use `C2F(getlabel)` to get a block label. See `routines/scicos/import.c` file for more details

Block indexes are those relative to the compile structure`cpr`.

SEE ALSO : `curblock` 552, `getscicosvars` 553, `setscicosvars` 554

11.4.9 getscicosvars — get Scicos data structure while running

CALLING SEQUENCE :

v=getscicosvars(name)

PARAMETERS :

`name` : a character string, the name of the required structure
`v` : vector of the structure value

DESCRIPTION :

This function may be used in a Scilab block to get value of some particular global data while running. It allows to write diagram monitoring blocks.
for example the instruction `disp(getscicosvars('x'))` displays the entire continuous state of the diagram.

```
x=getscicosvars('x');
xptr=getscicosvars('xptr');
disp(x(xptr(k):xptr(k+1)-1))
```

displays the continuous state of the `k` block

name	data structure definition
'x'	continuous state
'xptr'	continuous state splitting vector
'z'	discrete state
'zptr'	discrete state splitting vector
'rpar'	real parameters vector
'rpptr'	rpar splitting vector
'ipar'	integer parameters vector
'ipptr'	ipar splitting vector
'outtb'	vector of all input/outputs values
'inpptr'	inplnk splitting vector
'outptr'	outlnk splitting vector
'inplnk'	vector of input port values address in lnkptr
'outlnk'	vector of output port values address in lnpkpr
'lnkptr'	outtb splitting vector

See `scicos_cpr` for more detail on these data structures.
For C or fortran computational function the C procedure `C2F(getscicosvars)` may used. See `routines/scicos/import.c` file for more details.

SEE ALSO : `setscicosvars` 554, `scicosim` 551, `curblock` 552, `scicos_cpr` 549, `getblocklabel` 553

11.4.10 setscicosvars — set Scicos data structure while running

CALLING SEQUENCE :

setscicosvars(name,v)

PARAMETERS :

`name` : a character string, the name of the required structure
`v` : vector of the new structure value

DESCRIPTION :

This function may be used in a Scilab block to set value of some particular global data while running. It allows to write diagram supervisor blocks.
for example the instructions

```
x=getscicosvars('x');
xptr=getscicosvars('xptr');
x(xptr(k):xptr(k+1)-1)=xk
setscicosvars('x',x)
```

Changes the continuous state of the `k` block to `xk`.

name	data structure definition
'x'	continuous state
'xptr'	continuous state splitting vector
'z'	discrete state
'zptr'	discrete state splitting vector
'rpar'	real parameters vector
'rpptr'	rpar splitting vector
'ipar'	integer parameters vector
'ipptr'	ipar splitting vector
'outtb'	vector of all input/outputs values
'inpptr'	inplnk splitting vector
'outptr'	outlnk splitting vector
'inplnk'	vector of input port values address in lnkptr
'outlnk'	vector of output port values address in lnpkpr
'lnkptr'	outtb splitting vector

See `scicos_cpr` for more detail on these data structures.
For C or fortran computational function the C procedure `C2F(setscicosvars)` may used. See `routines/scicos/import.c` file for more details.
Warning: The use of this function requires a deep knowledge on how scicosim works, it must be used very carefully. Unpredicted parameters, state, link values changes may produce erroneous simulations.

SEE ALSO : `getscicosvars` 553, `scicosim` 551, `curblock` 552, `scicos_cpr` 549, `getblocklabel` 553

Chapter 12

Sound

12.0.11 analyze ________ frequency plot of a sound signal

CALLING SEQUENCE :

analyze (w [,fmin,fmax,rate,points])

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`.

EXAMPLE :

```
// 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);
```

12.0.12 loadwave ________ load a sound <<wav>> file into scilab

CALLING SEQUENCE :

```
x=loadwave('file-name');
```

PARAMETERS :

x : vector

DESCRIPTION :

Read a wav sound file into Scilab. you can transform other sound files into wav file with the `sox` program.

SEE ALSO : `savewave` 557, `analyze` 556, `mapsound` 556

12.0.13 mapsound ________ Plots a sound map

CALLING SEQUENCE :

```
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.

EXAMPLE :

```
// 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);
```

12.0.14 savewave save data into a sound <<wav>> file.

CALLING SEQUENCE :

```
savewave('file-name',x [, rate ]);
```

PARAMETERS :

`x` : vector
`rate` : a scalar. 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.

EXAMPLE :

```
// 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));
savewave(TMPDIR+'/foo.wav',s);
```

SEE ALSO : `loadwave` 556, `analyze` 556, `mapsound` 556

Chapter 13

Cumulative Distribution Functions, Inverses, Random variables

13.0.15 cdfbet — cumulative distribution function Beta distribution

CALLING SEQUENCE :

```
[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.

13.0.16 cdfbin — cumulative distribution function Binomial distribution

CALLING SEQUENCE :

```
[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.

13.0.17 cdfchi ______ cumulative distribution function chi-square distribution

CALLING SEQUENCE :

```
[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 chisqure 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.

13.0.18 cdfchn ______ cumulative distribution function non-central chi-square distribution

CALLING SEQUENCE :

```
[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.

13.0.19 cdff ______ cumulative distribution function F distribution

CALLING SEQUENCE :

```
[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.

13.0.20 cdffnc _____ cumulative distribution function non-central f-distribution

CALLING SEQUENCE :

```
[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.

13.0.21 cdfgam _ cumulative distribution function gamma distribution

CALLING SEQUENCE :

```
[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.

13.0.22 cdfnbn __ cumulative distribution function negative binomial distribution

CALLING SEQUENCE :

```
[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.

13.0.23 cdfnor _ cumulative distribution function normal distribution

CALLING SEQUENCE :

```
[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.

13.0.24 cdfpoi _ cumulative distribution function poisson distribution

CALLING SEQUENCE :

```
[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.

13.0.25 cdft _ cumulative distribution function Student's T distribution

CALLING SEQUENCE :

```
[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.

13.0.26 grand _ Random number generator

CALLING SEQUENCE :

```
Y=grand(m,n,'option' [,arg1,..,argn])
Y=grand(x,'option' [,arg1,....,argn])
Y=grand('option')
Y=grand('option'  [,arg1,....,argn])
```

PARAMETERS :

`grand('advnst',K)` : Advances the state of the current generator by 2^K values and resets the initial seed to that value.
`Y=grand(m,n,'bet',A,B), Y=grand(x,'bet',A,B)` : Returns random deviates from the beta distribution with parameters A and B. The density of the beta is `x^(a-1) * (1-x)^(b-1) / B(a,b)` for `0 < x < 1` Method: R. C. H. Cheng Generating Beta Variables with Nonintegral Shape Parameters Communications of the ACM, 21:317-322 (1978) (Algorithms BB and BC)

`Y=grand(m,n,'bin',N,P), Y=grand(x,'bin',N,P)` : Generates random deviates from a binomial distribution whose number of trials is N and whose probability of an event in each trial is P. N is the number of trials in the binomial distribution from which a random deviate is to be generated. P is the probability of an event in each trial of the binomial distribution from which a random deviate is to be generated. (0.0 <= P <= 1.0)

Method: This is algorithm BTPE from: Kachitvichyanukul, V. and Schmeiser, B. W. Binomial Random Variate Generation. Communications of the ACM, 31, 2 (February, 1988) 216.

`Y=grand(m,n,'chi',Df), Y=grand(x,'chi',Df)` : Generates random deviates from the distribution of a chisquare with DF degrees of freedom random variable. Uses relation between chisquare and gamma.

`Y=grand(m,n,'def'), Y=grand(x,'def')` : Returns random floating point numbers from a uniform distribution over 0 - 1 (endpoints of this interval are not returned) using the current generator

`Y=grand(m,n,'exp',Av), Y=grand(x,'exp',Av)` : Generates random deviates from an exponential distribution with mean AV. For details see: Ahrens, J.H. and Dieter, U. Computer Methods for Sampling From the Exponential and Normal Distributions. Comm. ACM, 15,10 (Oct. 1972), 873 - 882.

`Y=grand(m,n,'f',Dfn,Dfd), Y=grand(x,'f',Dfn,Dfd)` : Generates random deviates from the F (variance ratio) distribution with DFN degrees of freedom in the numerator and DFD degrees of freedom in the denominator. Method: Directly generates ratio of chisquare variates

`Y=grand(m,n,'gam',Shape,Scale), Y=grand(x,'gam',Shape,Scale)` : Generates random deviates from the gamma distribution whose density is (Scale**Shape)/Gamma(Shape) * X**(Shape-1) * Exp(-Scale*X) For details see:

`(Case R >= 1.0)` : Ahrens, J.H. and Dieter, U. Generating Gamma Variates by a Modified Rejection Technique. Comm. ACM, 25,1 (Jan. 1982), 47 - 54. Algorithm GD

`(Case 0.0 < R < 1.0)` : Ahrens, J.H. and Dieter, U. Computer Methods for Sampling from Gamma, Beta, Poisson and Binomial Distributions. Computing, 12 (1974), 223-246/ Adapted algorithm GS.

`G=grand('getcgn')` : Returns in G the number of the current random number generator (1..32)

`Sd=grand('getsd')` : Returns the value of two integer seeds of the current generator Sd=[sd1,sd2]

`grand('initgn',I)` : Reinitializes the state of the current generator

`I = -1` : sets the state to its initial seed

`I = 0` : sets the state to its last (previous) seed

`I = 1` : sets the state to a new seed 2^w values from its last seed

`Y=grand(m,n,'lgi'),Y=grand(x,'lgi')` : Returns random integers following a uniform distribution over (1, 2147483562) using the current generator.

`Y=grand(M,'mn',Mean,Cov)` :Generate M Multivariate Normal random deviates Mean must be a Nx1 matrix and Cov a NxN positive definite matrix Y is a NxM matrix

`Y=grand(n,'markov',P,x0)` Generates `n` successive states of a Markov chain described by the transition matrix `P`. Initial state is given by `x0`

`Y=grand(M,'mul',N,P)` Generate M observation from the Multinomial distribution. N is the Number of events that will be classified into one of the categories 1..NCAT P is the vector of probabilities. P(i) is the probability that an event will be classified into category i. Thus, P(i) must be [0,1]. P(i) is of size NCAT-1 (P(NCAT) is 1.0 minus the sum of the first NCAT-1 P(i). Y(:,i) is an observation from multinomial distribution. All Y(:,i) will be nonnegative and their sum will be N. Y is of size NcatxM

Algorithm from page 559 of Devroye, Luc. Non-Uniform Random Variate Generation. Springer-Verlag, New York, 1986.

`Y=grand(m,n,'nbn',N,P),Y=grand(x,'nbn',N,P)` : Generates random deviates from a negative binomial distribution. N is the required number of events (N > 0). P is The probability of an event during a Bernoulli trial (0.0 < P < 1.0).

Method: Algorithm from page 480 of Devroye, Luc. Non-Uniform Random Variate Generation. Springer-Verlag, New York, 1986.

`Y=grand(m,n,'nch',Df,Xnon), Y=grand(x,'nch',Df,Xnon)` : Generates random deviates from the distribution of a noncentral chisquare with DF degrees of freedom and noncentrality parameter XNONC. DF is he degrees of freedom of the chisquare (Must be >= 1.0) XNON the Noncentrality parameter of the chisquare (Must be >= 0.0) Uses fact that noncentral chisquare is the sum of a chisquare deviate with DF-1 degrees of freedom plus the square of a normal deviate with mean XNONand standard deviation 1.

`Y=grand(m,n,'nf',Dfn,Dfd,Xnon), Y=grand(x,'nf',Dfn,Dfd,Xnon)` : Generates random deviates from the noncentral F (variance ratio) distribution with DFN degrees of freedom in the numerator, and DFD degrees of freedom in the denominator, and noncentrality parameter XNONC. DFN is the numerator degrees of freedom (Must be >= 1.0) DFD is the Denominator degrees of freedom (Must be positive) XNON is the Noncentrality parameter (Must be nonnegative) Method: Directly generates ratio of noncentral numerator chisquare variate to central denominator chisquare variate.

`Y=grand(m,n,'nor',Av,Sd), Y=grand(x,'nor',Av,Sd)` : Generates random deviates from a normal distribution with mean, AV, and standard deviation, SD. AV is the mean of the normal distribution. SD is the standard deviation of the normal distribution. For details see: Ahrens, J.H. and Dieter, U. Extensions of Forsythe's Method for Random Sampling from the Normal Distribution. Math. Comput., 27,124 (Oct. 1973), 927 - 937.

`Sd=grand('phr2sd','string')` : Uses a phrase (character string) to generate two seeds for the RGN random number generator. Sd is an integer vector of size 2 `Sd=[Sd1,Sd2]`

`Y=grand(m,n,'poi',mu), Y=grand(x,'poi',mu)` : Generates random deviates from a Poisson distribution with mean MU. MU is the mean of the Poisson distribution from which random deviates are to be generated (MU >= 0.0). For details see: Ahrens, J.H. and Dieter, U. Computer Generation of Poisson Deviates From Modified Normal Distributions. ACM Trans. Math. Software, 8, 2 (June 1982),163-179

`Mat=grand(M,'prm',vect)` : Generate M random permutation of column vector `vect`. Mat is of size `(size(vect)xM)`

`grand('setall',ISEED1,ISEED2)` : Sets the initial seed of generator 1 to ISEED1 and ISEED2. The initial seeds of the other generators are set accordingly, and all generators states are set to these seeds.

`grand('setcgn',G)` : Sets the current generator to G. All references to a generator are to the current generator.

`grand('setsd',ISEED1,ISEED2)` : Resets the initial seed and state of generator g to ISEED1 and ISEED2. The seeds and states of the other generators remain unchanged.

`Y=grand(m,n,'uin',Low,High), Y=grand(x,'uin',Low,High)` : Generates integers uniformly distributed between LOW and HIGH. Low is the low bound (inclusive) on integer value to be generated. High is the high bound (inclusive) on integer value to be generated. If (HIGH-LOW) > 2,147,483,561 prints error message

`Y=grand(m,n,'unf',Low,High),Y=grand(x,'unf',Low,High)` : Generates reals uniformly distributed between LOW and HIGH. Low is the low bound (exclusive) on real value to be generated High is the high bound (exclusive) on real value to be generated

DESCRIPTION :

Interface fo Library of Fortran Routines for Random Number Generation (Barry W. Brown and James Lovato, Department of Biomathematics, The University of Texas, Houston)

This set of programs contains 32 virtual random number generators. Each generator can provide 1,048,576 blocks of numbers, and each block is of length 1,073,741,824. Any generator can be set to the beginning or end of the current block or to its starting value. The methods are from the paper cited immediately below, and most of the code is a transliteration from the Pascal of the paper into Fortran.

P. L'Ecuyer and S. Cote. Implementing a Random Number Package with Splitting Facilities. ACM Transactions on Mathematical Software 17:1, pp 98-111.

Most users won't need the sophisticated capabilities of this package, and will desire a single generator. This single generator (which will have a non-repeating length of 2.3 X 10^18 numbers) is the default. In order to accommodate this use, the concept of the current generator is added to those of the cited paper; references to a generator are always to the current generator. The current generator is initially generator number 1; it can be changed by 'setcgn', and the ordinal number of the current generator can be obtained from 'getcgn'.

The user of the default can set the initial values of the two integer seeds with 'setall'. If the user does not set the seeds, the random number generation will use the default values, 1234567890 and 123456789. The values of the current seeds can be achieved by a call to 'getsd'. Random number may be obtained as integers ranging from 1 to a large integer by reference to option 'lgi' or as a floating point number between 0 and 1 by a reference to option 'def'. These are the only routines needed by a user desiring a single stream of random numbers.

CONCEPTS :

A stream of pseudo-random numbers is a sequence, each member of which can be obtained either as an integer in the range 1..2,147,483,563 or as a floating point number in the range [0..1]. The user is in charge of which representation is desired.

The method contains an algorithm for generating a stream with a very long period, 2.3 X 10^18. This stream in partitioned into G (=32) virtual generators. Each virtual generator contains 2^20 (=1,048,576) blocks of non-overlapping random numbers. Each block is 2^30 (=1,073,741,824) in length.

The state of a generator is determined by two integers called seeds. The seeds can be initialized by the user; the initial values of the first must lie between 1 and 2,147,483,562, that of the second between 1 and 2,147,483,398. Each time a number is generated, the values of the seeds change. Three values of seeds are remembered by the generators at all times: the value with which the generator was initialized, the value at the beginning of the current block, and the value at the beginning of the next block. The seeds of any generator can be set to any of these three values at any time.

Of the 32 virtual generators, exactly one will be the current generator, i.e., that one will be used to generate values for 'lgi' and 'def'. Initially, the current generator is set to number one. The current generator may be changed by calling 'setcgn', and the number of the current generator can be obtained using 'getcgn'.

TEST EXAMPLE :

An example of the need for these capabilities 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 alalyzed. The second method will choose a data set size between 100 and 200, generate the data and alalyze 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 the package, it is a snap. Use generator 1 to obtain the sample size for method one and generator 2 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.

INTERFACE :

A random number is obtained either as a random integer between 1 and 2,147,483,562 by using option 'lgi' (large integer) or as a random floating point number between 0 and 1 by using option 'def'.

The seed of the first generator can be set by using option 'setall'; the values of the seeds of the other 31 generators are calculated from this value.

The number of the current generator can be set by using option 'setcgn' The number of the current generator can be obtained by using option 'getcgn'.

Chapter 14

Tools for dynamical systems

14.0.27 artest ________ arnold dynamical system

CALLING SEQUENCE :

```
artest(f_l,[odem,xdim,npts])
arnold(t,x)
iarf([a])
```

PARAMETERS :

- `f_l` : can be "arnold" or arnold. It is the name of the external for the arnold system. If `f_l` is set to "arnold" a Fortran coded version of the arnold system where a(1:6)=1 will be used and if `f_l` is set to arnold a Scilab coded version will be used . `arnold` is a Scilab macro coding the Arnold system. This macro is loaded when calling `artest`.
- `iarf` : is used to fix the values of `a` in the Scilab coded case. `a` is a vector of size 6.
- `odem,xdim,npts` : are optional arguments. Their meaning and syntax can be found in the `portr3d` help

DESCRIPTION :

A call to the function `artest()` will interactively display a phase portrait of a the following dynamical system :

```
ydot(1)=a(1)*cos(y(2)) +a(2)*sin(y(3))
ydot(2)=a(3)*cos(y(3)) +a(4)*sin(y(1))
ydot(3)=a(5)*cos(y(1)) +a(6)*sin(y(2))
```

SEE ALSO : `portr3d` 577, `ode` 350, `chaintest` 573, `lotest` 575

14.0.28 bifish ______ shows a bifurcation diagram in a fish population discrete time model

CALLING SEQUENCE :

```
bifish([f_ch])
```

PARAMETERS :

- `f_ch` : can be one of `fish`, `fishr` and `fishr2`. This option selects the population model.

DESCRIPTION :

The dynamical system `fish` is the following :

```
y=b*exp(-0.1*(x(k)_1+x(k)_2));
x(k+1)=[ y 2*y ; s 0.0]*x(k);
```

and the parameters `s` evolves to show the bifurcation diagram. `fishr` and `fishr2` are constructed as above but with added white noises.

```
fishr
y=b*exp(-0.1*(xk(1)+xk(2)))
xkp1=[ y 2*y ; s*(1+0.1*(rand()-0.5)) 0.0]*xk

fishr2
z=exp(-0.1*(xk(1)+xk(2)))
xkp1=[ b*z**(1+0.1*(rand()-0.5)) 2*b*z**(1+0.1*(rand()-0.5)) ; s 0.0]*xk
```

The three macros `fish, fishr, fishr2` are loaded in Scilab when calling `bifish`.

SEE ALSO : `ode` 350

14.0.29 boucle ___ phase portrait of a dynamical system with observer

CALLING SEQUENCE :

```
[]=boucle(fch,[abruit,xdim,npts,farrow])
```

PARAMETERS :

`fch` : Scilab macro. `fch` is supposed to be an observed-controlled system with noisy output of state dimension 4 ([x;xchap] is of dimension 4). `fch` can be created with the macro `obscont1` or can be set to one of the two following string which gives pre computed examples
`"bcomp"` : for a non-linear competition model.
`"lcomp"` : for a linear example.
`abruit` : give the noise variance.
`xdim,npts,farrow` : See portrait

DESCRIPTION :
Phase portrait of dynamical systems.

SEE ALSO : `portrait` 578, `ode` 350, `obscont1` 577

14.0.30 chaintest ___ a three-species food chain model

CALLING SEQUENCE :

```
chaintest([f_l,b1,odem,xdim,npts])
[xdot]=chain(t,x)
[z1]=ch_f1(u)
[z2]=ch_f2(u)
```

PARAMETERS :

`f_l` : the name of the macro which code the three-species food chain model (default value `chain`).
`odem,xdim,npts` : are optional arguments. Their meaning and syntax can be found in the `portr3d` help

DESCRIPTION :
A call to the function `chaintest()` will interactively display a phase portrait of a three-species food chain model given by:

```
ff1= f1(x(1))
ff2= f2(x(2))

xdot1= x(1)*(1-x(1)) - ff1*x(2)
xdot2= ff1*x(2) -  ff2*x(3) - 0.4*x(2)
xdot3= ff2*x(3) - 0.01*x(3)
```

and

```
f1(u)=5*u/(1+b1*u)
f2(u)z2=0.1*u/(1+2*u)
```

The default value for `b1` is 3.0.
The Scilab macros `chain(t,x)`,`f1(u)`,`f2(u)` code the dynamical system

SEE ALSO : `portr3d` 577, `ode` 350

14.0.31 gpeche ___ a fishing program

CALLING SEQUENCE :

```
[xk,ukp1]=gpeche(uk,pasg)
[ut]=peche(t)
[pdot]=pechep(t,p)
```

DESCRIPTION :
`gpeche` Iterates a gradient method on a fishing problem Computes the trajectory associated to the command law `uk` prints the cost value and computes a new control.

14.0.32 fusee ___ a set of Scilab macro for a landing rocket problem

FUSEE :

```
[xdot]=fusee(t,x)
```

Dynamical motion equation for the rocket

FINIT :

```
finit()
```

Initialises the following parameters for rocket landing.

`k` : The acceleration of the rocket engines
`gamma` : The moon gravity acceleration.
`umax` : the gaz ejection flow out.
`mcap` : the mass of the space capsule.
`cpen` : penalisation in the cost function of the final state.

FUSEEGRAD :

```
[ukp1]=fuseegrad(niter,ukp1,pasg)
```

`niter` : number of gradient iteration steps.
`ukp1` : initial control value (vector of sie 135)
`pasg` : the gradient step value.

DESCRIPTION :
Iterate a gradient method and returns the computed control.

FUSEEP :

```
[pdot]=fuseep(t,p)
```

DESCRIPTION :
adjoint equation for the landing rocket problem.

POUSSE :

```
[ut]=pousse(t)
```

return the value of a piece wise constant control build on the discrete control `uk`

UBANG :

```
[uk]=ubang(tf,tcom)
```

returns a bang-bang control, 0 form time 0 to tcom and 1 form tcom to tf.

FCOUT :

```
[c,xk,pk,ukp1]=fcout(tf,uk,pasg)
```

DESCRITION :

optimise the following cost function by gradient iterations.

```
c = -m(tf) + C*( h(tf)**2 + v(tf)**2)
```

SFUSEE :

```
[]=sfusee(tau,h0,v0,m0,Tf)
```

DESCRIPTION :

computes the rocket trajectory when a bang-bang control is used `tau` is the commutation time.

- `h0` : The initial position (high)
- `v0` : The initial speed (negative if the rocket is landing)
- `m0` : The total initial mass (capsule and fuel).
- `Tf` : Time horizon.

EQUAD :

DESCRIPTION :

```
[xk,pk]=equad(tf,uk)
```

Computes the state and adjoint state of the rocket system for a given control `ur`.

TRAJ :

```
[xt]=traj(t)
```

returns a piece wise value of the mass evolution.

14.0.33 lotest ______ demo of the Lorenz attractor

CALLING SEQUENCE :

```
[]=lotest([f_l,odem,xdim,npts,pinit])
[y]=lorenz(t,x)
[]=ilo(sig,ro,beta)
[]=ilof(sig,ro,beta)
```

PARAMETERS :

- `f_l` : can be `"loren"` or `lorenz`. it is the name of the external for the Lorenz system. `"loren"` will use a Fortran coded version of the lorenz system and arnold will and `loren` will use a Scilab coded version.`lorentz` is the Scilab macro which code the lorenz system. This macro is loaded when calling `lotest`.
- `ilof, ilo` :are used to fix the parameters of the Fortran and Scilab code version of the Lorenz system.
- `odem,xdim,npts` : are optional arguments. Their meaning and syntax can be found in the portr3d help

DESCRIPTION :

A call to the function `lotest()` will interactively display a phase portrait of a the following dynamical system

```
y(1)=sig*(x(2)-x(1));
y(2)=ro*x(1) -x(2)-x(1)*x(3);
y(3)=-beta*x(3)+x(1)*x(2);
```

SEE ALSO : `portr3d` 577, `ode` 350, `chaintest` 573, `lotest` 575

14.0.34 mine ______ a mining problem

CALLING SEQUENCE :

```
[cout,feed]=mine(n1,n2,uvect)
```

PARAMETERS :

- `n1` : Number of discrete point for the state.
- `n2` : Number of time step
- `uvect` : a row vector which gives the possible control value (integer values). for example `u=[-1,0,1]` means that at each step we come down one step or stay at the same level or rise one step).
- `cout(n1,n2)` : The Bellman values.
- `feed(n1,n2)` : The feedback Law.

DESCRIPTION :

Dynamic programming applied to an optimal extraction of ore in an opencast mine. The extraction is done as follows : the steam shovel move forward for (k=1,2,...,n2) at each step it takes the ore, then move up or down (or stay at the same level) according to the control value to reach another level at next step. The extraction process must maximise the following cost :

```
-- n2-1
\\
/        f(x(k),k) + V_F(x,n2)
-- k=1
```

with `x(k+1)=x(k) + u`. `x(k)` is the trajectory depth at step `k` (`x=1` is the ground level).

The instantaneous cost f(i,k) `stands for the benefit of digging at depth i` at position `k`. It must be given as a Scilab macro `ff_o`

```
[y]=ff_o(x,k)
```

and for efficiency `ff_o` must accept and return column vectors for `x` and `y`.

`V_F(i,n2)` is a final cost which is set so as to impose the steam shovel to be at ground level at position `n2`

FF_O :

SHOWCOST :

CALLING SEQUENCE :

```
[]=showcost(n1,n2,teta,alpha)
```

DESCRIPTION :

Shows a 3D representation of the instantaneous cost.

14.0.35 obscont1 — a controlled-observed system

CALLING SEQUENCE :

```
[macr]=obscont1(sysn)
```

PARAMETERS :

`sysn` : A Scilab string which gives the name of the controlled system.
`gaincom,gainobs` : column vectors giving the requested gains
`macr` : a new Scilab function which simulates the controlled observed system.

```
[x1dot]=macr(t,x1,abruit,pas,n)
x1=[x;xchap],
```

DESCRIPTION :
This macros return a new function which computes the controlled observed version of a linearised system around the `(xe,ue)` point.
before calling this function, a noise vector `br` should be created. the equilibrium point `(xe,ue)` should be given as a global Scilab. the linearised system f,g,h and the two gain matrices `l,k` are returned as global Scilab data.

14.0.36 portr3d — 3 dimensional phase portrait.

CALLING SEQUENCE :

```
[]=portr3d(f,[odem,xdim,npts,pinit])
```

PARAMETERS :

`f` : a Scilab external which gives the field of the dynamical system. Hence it can be a macro name which computes the field at time t and point x [y]=f(t,x,[u]) or a list list(f1,u1) where f1 is a macro of type [y]=f1(t,x,u) or a character string
`rest` : The other parameters are optional. If omitted they will be asked interactively
`odem` : gives the integration method to use. The value "default" can be used, otherwise see ode for a complete set of possibilities
`npts` : a vector of size (2,10) [number-of-points,step] gives the step for integration and the number of requested points. The solution will be calculated and drawn for time=0:step:(step*[number-of-points])
`xdim` : [xmin,xmax,ymin,ymax,zmin,zmax] the boundaries of the graphic frame.
`pinit` : initial values for integration. A set of initial points can be given in a matrix

```
pinit = [x0(1), x1(1),...., xn(1)
         x0(2), x1(2),...., xn(2)
         x0(3), x1(3),...., xn(3)].
```

DESCRIPTION :
Interactive integration and display of a 3 dimensional phase portrait of a dynamical system dx/dt=f(t,x,[u]) (where u is an optional parameter)

SEE ALSO : `ode` 350

14.0.37 portrait — 2 dimensional phase portrait.

CALLING SEQUENCE :

```
[]=portrait(f,[odem,xdim,npts,pinit])
```

PARAMETERS :

`f` : a Scilab external which gives the field of the dynamical system. Hence it can be a macro name which computes the field at time t and point x [y]=f(t,x,[u]) or a list list(f1,u1) where f1 is a macro of type [y]=f1(t,x,u) or a character string. The macro can be used to simulate a continuous or discrete system and in case of discrete system the second parameter must be set to 'discrete'
`rest` : The other parameters are optional. If omitted they will be asked interactively
`odem` : gives the integration method to use. The value "default" can be used, otherwise see ode for a complete set of possibilities
`npts` : a vector of size (2,10) [number-of-points,step] gives the step for integration and the number of requested points. The solution will be calculated and drawn for time=0:step:(step*[number-of-points])
`xdim` : [xmin,xmax,ymin,ymax,zmin,zmax] the boundaries of the graphic frame.
`pinit` : initial values for integration. A set of initial points can be given in a matrix

```
pinit = [x0(1), x1(1),...., xn(1)
         x0(2), x1(2),...., xn(2)
         x0(3), x1(3),...., xn(3)].
```

DESCRIPTION :
Interactive integration and display of a 2 dimensional phase portrait of a dynamical system dx/dt=f(t,x,[u]) (where u is an optional parameter)

EXAMPLE :

```
a=rand(2,2)
deff('[ydot]=l_s(t,y)','ydot=a*y')
portrait(l_s)
```

SEE ALSO : `ode` 350

14.0.38 recur — a bilinear recurrent equation

CALLING SEQUENCE :

```
[y]=recur(x0,var,k,n)
[integr]=logr(k,var)
```

DESCRIPTION :
computes solutions of a bilinear recurrent equation

```
x(i+1)=-x(i)*(k + sqrt(var)*br(i))
```

with initial value `x0` and driven by a white noise of variance `var`.
Trajectories are drawn and the empirical Lyapunov exponent is returned (`x(i)` is not to much different from `exp(y*i)`)
A theoretical computation of the Lyapunov exponent is given by

```
[integr]=logr(k,var)
```

14.0.39 systems — a collection of dynamical system

CALLING SEQUENCE :

```
[]=systems()
```

DESCRIPTION :
A call to this function will load into Scilab a set of macros which describes dynamical systems. Their parameters can be initiated by calling the routine tdinit().

BIOREACT :

```
[ydot]=biorecat(t,x)
```

a bioreactor model,

`x(1)` is the biomass concentration
`x(2)` is the sugar concentration

```
xdot(1)=mu_td(x(2))*x(1)- debit*x(1);
xdot(2)=-k*mu_td(x(2))*x(1)-debit*x(2)+debit*x2in;
```

where mu_td is given by

```
mu_td(x)=x/(1+x);
```

COMPET :

```
[xdot]=compet(t,x [,u])
```

a competition model. `x(1),x(2)` stands for two populations which grows on a same resource. `1/u` is the level of that resource (1 is the default value).

```
xdot=0*ones(2,1);
xdot(1) = ppr*x(1)*(1-x(1)/ppk) - u*ppa*x(1)*x(2) ,
xdot(2) = pps*x(2)*(1-x(2)/ppl) - u*ppb*x(1)*x(2) ,
```

The macro `[xe]=equilcom(ue)` computes an equilibrium point of the competition model and a fixed level of the resource ue (default value is 1)
The macro `[f,g,h,linsy]=lincomp([ue])` gives the linearisation of the competition model (with output y=x) around the equilibrium point xe=equilcom(ue). This macro returns [f,g,h] the three matrices of the linearised system. and linsy which is a Scilab macro [ydot]=linsy(t,x) which computes the dynamics of the linearised system

CYCLLIM :

```
[xdot]=cycllim(t,x)
```

a model with a limit cycle

```
xdot=a*x+qeps(1-||x||**2)x
```

LINEAR :

```
[xdot]=linear(t,x)
```

a linear system
BLINPER :

```
[xdot]=linper(t,x)
```


a linear system with quadratic perturbations.
POP :

```
[xdot]=pop(t,x)
```

a fish population model

```
xdot= 10*x*(1-x/K)- peche(t)*x
```

PROIE :
a Predator prey model with external insecticide.

```
[xdot]=p_p(t,x,[u]
```

`x(1)` The prey (that we want to kill)
`x(2)` the predator (that we want to preserve)
`u` mortality rate due to insecticide which destroys both prey and predator (default value u=0)

```
xdot(1) = ppr*x(1)*(1-x(1)/ppk) - ppa*x(1)*x(2) - u*x(1);
xdot(2) = -ppm*x(2)             + ppb*x(1)*x(2) - u*x(2);
```

The macro `[xe]=equilpp([ue])` computes the equilibrium point of the `p_p` system for the value `ue` of the command. The default value for `ue` is 0.

```
xe(1) =  (ppm+ue)/ppb;
xe(2) =  (ppr*(1-xe(1)/ppk)-ue)/ppa;
```

LINCOM :

```
[xdot]=lincom(t,x,k)
```

linear system with a feedback

```
xdot= a*x +b*(-k*x)
```

SEE ALSO : tdinit 581

14.0.40 tangent — linearization of a dynamical system at an equilibrium point

CALLING SEQUENCE :

```
[f,g,newm]=tangent(ff,xe,[ue])
```

PARAMETERS :

`ff` : a string which gives the name of the Scilab macro which codes the system
`xe` : column vector which gives the equilibrium point for the value ue of the parameter
`ue` : real value.
`f, g` : two matrices for the linearised system `dxdot=f.dx + g.du`
`newm` : A new macro of type [y]=newm(t,x,u) which computes the field of the linearised system (newm(t,xe,ue)=0)

DESCRIPTION :
linearises around the equilibrium point `(xe,ue)` the vector field of the dynamical system given by a Scilab macro `ff`, `xdot=ff(t,x,[u])`. The dynamical system is supposed to be autonomous.

14.0.41 tdinit interactive initialisation of the tdcs dynamical systems

CALLING SEQUENCE :

```
tdinit()
```

DESCRIPTION :
This macro can be used to interactively define the parameters needed by the dynamical systems described in systems

bioreactor model
competition model
system with limit cycle
linear system
quadratic model
linear system with a feedback

SEE ALSO : `portrait` 578, `systems` 579

Chapter 15

Tools for fractal analysis

15.0.42 alphagifs ______ Holder function estimation using IFS

prey predatory model

Author: Khalid Daoudi

Estimates the pointwise Holder exponents of a 1-D real signal using the GIFS method.

USAGE :
[Alpha, Ci]=wave2gifs(sig, limtype)

INPUT PARAMETERS :

- sig : Real vector [1,n] or [n,1] Contains the signal to be analysed.
- limtype : Character string Specifies the type of limit you want to use. You have the choice between 'slope' and 'cesaro'.

OUTPUT PARAMETERS :

- Alpha : Real vector Contains the estimated Holder function of the signal.
- Ci : Real matrix Contains the GIFS coefficients obtained using the Schauder basis.

DESCRIPTION :

PARAMETERS :

- sig is a real vector [1,n] or [n,1] which contains the signal to be analysed.
- limtype is a character string Specifies the type of limit you want to use. You have the choice between 'slope' and 'cesaro'.
- Alpha is a real vector which contains the estimated Holder function of the signal i.e the estimated pointwise Holder exponent a each point of the given signal.
- Ci is a real matrix which contains the GIFS coefficients obtained as the ration between (synchrounous) Schauder basis coefficients at succesive scales.

ALGORITHM DETAILS :
The algorithm uses the GIFS method to estimate the Holder exponent at each point of a given signal. The first step of this method consists in computing the coefficients of the GIFS whose attractor is the given signal. In the second step, we replace each coefficient which absolute value is greater than 1 (resp. smaller than 1/2) by 1 (resp. by 1/2). We then perform the computation of the limit that yields the estimated Holder function using the chosen type of limit.

SEE ALSO: :
gifs and prescalpha

EXAMPLE: :
Synthesis of an fbm with exponent H=0.7 (of size 1024 samples) : x = fmblevinson(1024,0.7) ; Estimation of The Holder function : Alpha = alphagifs(x,'slope'); plot(Alpha)

15.0.43 AtanH ______ Arctangent variation

Author: Paulo Goncalves

Generates an arc-tangent trajectory

USAGE :

```
Ht=AtanH(N,h1,h2,shape);
```

INPUT PARAMETERS :

- N : Positive integer Sample size of the trajectory
- h1 : Real scalar First value of the arc-tangent trajectory
- h2 : Real scalar Last value of the arc-tangent trajectory
- shape : real in [0,1] smoothness of the trajectory shape = 0 : constant piecewise (step function) shape = 1 : linear

OUTPUT PARAMETERS :

- Ht : real vector [1,N] Time samples of the arc-tangent trajectory

SEE ALSO: :

EXAMPLE: :
[Ht] = AtanH(1024,0,1,0.9,0.01) ;

15.0.44 bbch ______ beneath-beyond concave hull

Author: Christophe Canus

This C_LAB routine determines the concave hull of a function graph using the beneath-beyond algorithm.

USAGE :
[rx,ru_x]=bbch(x,u_x)
INPUT PARAMETERS :

- x : real vector [1,N] or [N,1] Contains the abscissa.
- u_x : real vector [1,N] or [N,1] Contains the function to be regularized.

OUTPUT PARAMETERS :

- rx : real vector [1,M] Contains the abscissa of the regularized function.
- ru_x : real vector [1,M] Contains the regularized function.

DESCRIPTION :

PARAMETERS :
The abscissa x and the function u_x to be regularized must be of the same size [1,N] or [N,1]. The abscissa rx and the concave regularized function ru_x are of the same size [1,M] with M<=N.
ALGORITHM DETAILS :
Standard beneath-beyond algorithm.
EXAMPLES :

MATLAB :

```
h=.3;beta=3;
N=1000;
% chirp singularity (h,beta)
x=linspace(0.,1.,N);
u_x=abs(x).^h.*sin(abs(x).^(-beta));
plot(x,u_x);
hold on;
[rx,ru_x]=bbch(x,u_x);
plot(rx,ru_x,'rd');
plot(x,abs(x).^h,'k');
```

SCILAB :

```
//
```

REFERENCES :
None.SH See Also linearlt (C_LAB routine).

15.0.45 binom ______ binomial measure synthesis

Author: Christophe Canus

This C_LAB routine synthesizes a large range of pre-multifractal measures related to the binomial measure paradigm (deterministic, shuffled, pertubated, and mixing of two binomials: lumping and sum) and computes linked theoretical functions (partition sum function, Reyni exponents function, generalized dimensions, multifractal spectrum).

USAGE :
[varargout,[optvarargout]]=binom(p0,str,varargin,[optvarargin])
INPUT PARAMETERS :

- p0 : strictly positive real scalar Contains the weight of the binomial.
- str : string Contains the type of ouput.

- varargin : variable input argument Contains the variable input argument.
- optvarargin : optional variable input arguments Contains optional variable input arguments.

OUTPUT PARAMETERS :

- varargout : variable output argument Contains the variable output argument.
- optvarargout : optional variable output argument Contains an optional variable output argument.

DESCRIPTION :

PARAMETERS :

The binomial measure is completly characterized by its weight p0. This first parameter must be >0. and <1. (the case of p0=.5 corresponds to the Lebesgue measure). The second parameter str is a variable string used to determine the desired type of output. There are six suffix strings ('meas' for measure, 'cdf' for cumulative distribution function, , 'pdf' for probability density function, 'part' for partition sum function, 'Reyni' for Reyni exponent function , 'spec' for multifractal spectrum) for the deterministic binomial measure and a lot of possibly composed prefix strings for related measures ('shuf' for shuffled, 'pert' for pertubated, 'lump' for lumping , 'sum' for sum, 'sumpert' for sum of pertubated, and so on) which can be added to the first ones to form composed strings. For example, 'lumppertmeas' is for the synthesis of the lumping of 2 pertubated binomial pre-multifractal measures and 'sumspec' is for the computation of the multifractal spectrum of the sum of two binomials. Note that all combinaisons of strings are not implemented yet. When a string containing suffix string 'meas' is given as second input, a pre-multifractal measure mu_n (first output argument) is synthesized on the dyadic intervals I_n (second optional output argument) of the unit interval. In that case, the third input argument is a strictly positive real (integer) scalar n which contains the resolution of the pre-multifractal measure. The size of the output real vectors mu_n (and I_n if used) is equal to 2n (so be aware the stack size ;-)). This option is implemented for the deterministic ('meas'), shuffled ('shufmeas') and pertubated ('pertmeas') binomial, and also for the mixing (lumping or sum) of two deterministic ('lumpmeas' and 'summeas') or pertubated ('lumppertmeas' and 'sumpertmeas') binomial measures. When a string containing prefix 'shuf' is given as second input, the synthesis is made for a shuffled binomial measure. At each level of the multiplicative cascade and for all nodes of the corresponding binary tree, the weight is chosen uniformly among p0 and 1-p0. This option is implemented only for the binomial measure ('shufmeas'). When a string containing prefix 'pert' is given as second input, the synthesis is made for a pertubated binomial measure. In that case, the fourth input argument is a strictly positive real scalar epsilon which contains the pertubation around weights. The weight is an independant random variable identically distributed between p0-epsilon and p0+epsilon which must be >0., <1. This option is implemented only for the binomial measure ('pertmeas') and the mixing (lumping and sum) of two binomial measures ('lumppertmeas' and 'sumpertmeas'). When replacing suffix string 'meas' with suffix string 'cdf', respectively suffix string 'pdf', the cumulative distribution function F_n, respectively the probability density function p_n, related to this pre-multifractal measure is computed (first output argument). When string 'part' is given as second input, the partition sum function znq of multifractal measure is computed as sole output argument. In that case, the third input argument is a strictly positive real (integer) vector vn which contains the resolutions, and the fourth input argument is a real vector q which contains the measure exponents. The size of the output real matrix znq is equal to size(q)*size(vn). This option is implemented only for the binomial measure. When string 'Reyni' is given as second input, the Reyni exponents function tq (and the generalized dimensions Dq if used) of the multifractal measure is computed as first output argument (and second optional output argument if used). In that case, the third input argument is a real vector q which contains the measure's exponents. The size of the output real vector tq is equal to size(q)). This option is implemented only for the binomial measure. When a string containing suffix string 'spec' is given as second input, the multifractal spectrum f_alpha

(second output argument) is synthesized on the Hoelder exponents alpha (first output argument). In that case, the third input argument is a strictly positive real (integer) scalar N which contains the number of Hoelder exponents. The size of both output real vectors alpha and f_alpha is equal to N. This option is implemented only for the binomial measure ('spec') and the mixing (lumping and sum) of two binomial measures ('lumpspec' and sumspec').

ALGORITHM DETAILS :

For the deterministic binomial, the pre-multifractal measure synthesis algorithm is implemented is a iterative way (supposed to run faster than a recursive one). For the shuffled or the pertubated binomial, the synthesis algorithm is implemented is a recursive way (to be able to pick up a i.i.d. r.v. at each level of the multiplicative cascade and for all nodes of the corresponding binary tree w.r.t. the given law). Note that the shuffled binomial measure is not conservative.

EXAMPLES :

MATLAB :

```
p0=.2;
n=10;
% synthesizes a pre-multifractal binomial measure
[mu_n,I_n]=binom(p0,'meas',n);
plot(I_n,mu_n);
% synthesizes the cdf of a pre-multifractal shuffled binomial measure
F_n=binom(p0,'shufcdf',n);
plot(I_n,F_n);
e=.19;
% synthesizes the pdf of a pre-multifractal pertubated binomial measure
p_n=binom(p0,'pertpdf',n,e);
plot(I_n,p_n);
vn=[1:1:8];
q=[-5:.1:+5];
% computes the partition sum function of a binomial measure
znq=binom(p0,'part',vn,q);
plot(-vn*log(2),log(znq));
% computes the Reyni exponents function of a binomial measure
tq=binom(p0,'Reyni',q);
plot(q,tq);
N=200;
q0=.4;
% computes the multifractal spectrum of the lumping of two binomial measures
[alpha,f_alpha]=binom(p0,'lumpspec',N,q0);
plot(alpha,f_alpha);
```

SCILAB :

```
p0=.2;
n=10;
// synthesizes a pre-multifractal binomial measure
[mu_n,I_n]=binom(p0,'meas',n);
plot(I_n,mu_n);
// synthesizes the cdf of a pre-multifractal shuffled binomial measure
```

```
F_n=binom(p0,'shufcdf',n);
plot(I_n,F_n);
e=.19;
// synthesizes the pdf of a pre-multifractal pertubated binomial measure
p_n=binom(p0,'pertpdf',n,e);
plot(I_n,p_n);
xbasc();
vn=[1:1:8];
q=[-5:.1:+5];
// computes the partition sum function of a binomial measure
znq=binom(p0,'part',vn,q);
mn=zeros(max(size(q)),max(size(vn)));
for i=1:max(size(q))
   mn(i,:)=-vn*log(2);
end
plot2d(mn',log(znq'));
// computes the Reyni exponents function of a binomial measure
tq=binom(p0,'Reyni',q);
plot(q,tq);
N=200;
q0=.4;
// computes the multifractal spectrum of the lumping of two binomial measures
[alpha,f_alpha]=binom(p0,'lumpspec',N,q0);
plot(alpha,f_alpha);
```

REFERENCES :

"Multifractal Measures", Carl J. G. Evertsz and Benoit B. MandelBrot. In Chaos and Fractals, New Frontiers of Science, Appendix B. Edited by Peitgen, Juergens and Saupe, Springer Verlag, 1992 pages 921-953. "A class of Multinomial Multifractal Measures with negative (latent) values for the "Dimension" f(alpha)", Benoit B. MandelBrot. In Fractals' Physical Origins and Properties, Proceeding of the Erice Meeting, 1988. Edited by L. Pietronero, Plenum Press, New York, 1989 pages 3-29. .SH See also sbinom, multim1d, multim2d, smultim1d, smultim2d (C_LAB routines). MFAS_measures, MFAS_dimensions, MFAS_spectra (Matlab and/or Scilab demo scripts).

15.0.46 contwt ______ Continuous L2 wavelet transform

Author: Paulo Goncalves

Computes a continuous wavelet transform of a 1-D signal (real or complex). The scale operator is unitary with respect to the L2 norm. Two closed form wavelets are available: the Mexican Hat or the Morlet Wavelet (real or analytic). For arbitrary analyzing wavelets, numerical approximation is achieved using a Fast Mellin Transform.

USAGE :

[wt,scale,f,scalo,wavescaled]=contwt(x,[fmin,fmax,N,wvlt_length])

INPUT PARAMETERS :

- x : Real or complex vector [1,nt] or [nt,1] Time samples of the signal to be analyzed.
- fmin : real scalar in [0,0.5] Lower frequency bound of the analysis. When not specified, this parameter forces the program to interactive mode.

- fmax : real scalar [0,0.5] and fmax > Upper frequency bound of the analysis. When not specified, this parameter forces the program to interactive mode.
- N : positive integer. number of analyzing voices. When not specified, this parameter forces the program to interactive mode.
- wvlt_length : scalar or vector specifies the analyzing wavelet: 0: Mexican hat wavelet (real) Positive real integer: real Morlet wavelet of size 2*wvlt_length+1) at finest scale 1 Positive imaginary integer: analytic Morlet wavelet of size 2*wvlt_length+1) at finest scale 1 Real valued vector: waveform samples of an arbitrary bandpass function.

OUTPUT PARAMETERS :

- wt : Real or complex matrix [N,nt] coefficients of the wavelet transform.
- scale : real vector [1,N] analyzed scales
- f : real vector [1,N] analyzed frequencies
- scalo : real positive valued matrix [N,nt] Scalogram coefficients (squared magnitude of the wavelet coefficients wt)
- wavescaled : Scalar or real valued matrix [length(wavelet at coarser scale)+1,N]

  Dilated versions of the analyzing wavelet

DESCRIPTION :

PARAMETERS :

- x : signal to be analyzed. Real or complex vector
- fmin : lower frequency bound of the analysis. fmin is real scalar comprised in [0,0.5]
- fmax : upper frequency bound of the analysis. fmax is a real scalar comprised in [0,0.5] and fmax > fmin
- N : number of analyzing voices geometrically sampled between minimum scale fmax/fmax and maximum scale fmax/fmin.
- wvlt_length : specifies the analyzing wavelet: 0: Mexican hat wavelet (real). The size of the wavelet is automatically fixed by the analyzing frequency Positive real integer: real Morlet wavelet of size 2*wvlt_length+1) at finest scale (1) Positive imaginary integer: analytic Morlet wavelet of size 2*|wvlt_length|+1) at finest scale 1. The corresponding wavelet transform is then complex. May be usefull for event detection purposes. Real valued vector: corresponds to the time samples waveform of any arbitrary bandpass function viewed as the analyzing wavelet at any given scale. Then, an approximation of the scaled wavelet versions is achieved using the Fast Mellin Transform (see dmt and dilate).
- wt : coefficients of the wavelet transform. X-coordinated corresponds to time (uniformly sampled), Y-coordinates correspond to frequency (or scale) voices (geometrically sampled between fmax (resp. 1) and fmin (resp. fmax / fmin). First row of wt corresponds to the highest analyzed frequency (finest scale).
- scale : analyzed scales (geometrically sampled between 1 and fmax /fmin
- f : analyzed frequencies (geometrically sampled between fmax and fmin . f corresponds to fmax/scale
- scalo : Scalogram coefficients (squared magnitude of the wavelet coefficients wt)
- wavescaled : If wvlt_length is a real or Imaginary pure scalar, then wavescaled equal wvlt_length . If wvlt_length is a vector (containing the waveform samples of an arbitrary analyzing wavelet), then wavescaled contains columnwise all scaled version of wvlt_length used for the analysis. In this latter case, first element of each column gives the effective time support of the analyzing wavelet at the corresponding scale. wavescaled can be used for reconstructing the signal (see icontwt)

ALGORITHM DETAILS :

The wavelet transform of x is computed via convolutions of dilated and translated versions of a single function called the "mother wavelet". The scales are given by the dilatation factor. As the scales are not absolute, the scale factor is determined through the specification of the minimum and maximum frequency of the decomposition considered as a time/frequency transform. The maximum frequency might not be greater than the Nyquist Frequency i.e. 0.5 as the wavelet at this scale would be undersampled (and therefore would create aliasing). The number of scales tells how many convolutions are computed. The bigger it is, the slower the transform is. The frequency (or scale) axis is geometrically sampled. The resulting scales and frequencies values can be obtained as output parameters. The meaning of the wavelet length parameter is manyfold. When non zero integer, it tells the routine to use a real Morlet Wavelet and gives its length at scale 1 (maximum frequency). When it is a positive imaginary integer, the analytic Morlet wavelet is used. If wvlt_length = 0, the Mexican Hat is used. The resulting wavelet transform is then real but has a quite poor frequency resolution. If wvlt_length is a real vector, it corresponds to the analyzing wavelet waveform in time at any arbitrary scale. Dilated and compressed version of it (according to the range [fmin , fmax] are computed directly from wvlt_length using a Fast Mellin Transform. For all choices of wavelet, approximative reconstruction of the decomposed signal is possible (see icontwt).

SEE ALSO: :
icontwt, contwtmir and cwt

EXAMPLE: :
Signal synthesis

```
x = morlet(0.1,128) ;
```

A Morlet (of size 2*8+1 samples) wavelet transform

```
[wtMorlet,scale,f,scaloMorlet] = contwt(x,0.01,0.5,128,8) ;
viewmat(scalomor,1:257,1:128);
```

Compared with a Mexican hat wavelet transform

```
[wtMex,scale,f,scaloMex] = contwt(x,0.01,0.5,128,0) ;
viewmat(scaloMex,1:257,1:128);
```

15.0.47 contwtmir _ Continuous L2 wavelet transform with mirroring

Author: Paulo Goncalves

Computes a continuous wavelet transform of a mirrored 1-D signal (real or complex). The scale operator is unitary with respect to the L2 norm. Two closed form wavelets are available: the Mexican Hat or the Morlet Wavelet (real or analytic). For arbitrary analyzing wavelets, numerical approximation is achieved using a Fast Mellin Transform.

USAGE :
[wt,scale,f,scalo,wavescaled]=contwtmir(x,[fmin,fmax,N,wvlt_length])

INPUT PARAMETERS :

- x : Real or complex vector [1,nt] or [nt,1] Time samples of the signal to be analyzed.
- fmin : real scalar in [0,0.5] Lower frequency bound of the analysis. When not specified, this parameter forces the program to interactive mode.
- fmax : real scalar [0,0.5] and fmax > Upper frequency bound of the analysis. When not specified, this parameter forces the program to interactive mode.
- N : positive integer. number of analyzing voices. When not specified, this parameter forces the program to interactive mode.
- wvlt_length : scalar or vector specifies the analyzing wavelet: 0: Mexican hat wavelet (real) Positive real integer: real Morlet wavelet of size 2*wvlt_length+1) at finest scale 1 Positive imaginary integer: analytic Morlet wavelet of size 2*wvlt_length+1) at finest scale 1 Real valued vector: waveform samples of an arbitrary bandpass function.

OUTPUT PARAMETERS :

- wt : Real or complex matrix [N,nt] coefficient of the wavelet transform.
- scale : real vector [1,N] analyzed scales
- f : real vector [1,N] analyzed frequencies
- scalo : real positive valued matrix [N,nt] Scalogram coefficients (squared magnitude of the wavelet coefficients wt)
- wavescaled : Scalar or real valued matrix [length(wavelet at coarser scale)+1,N]

  Dilated versions of the analyzing wavelet

DESCRIPTION :

PARAMETERS :

- x : signal to be analyzed. Real or complex vector
- fmin : lower frequency bound of the analysis. fmin is real scalar comprised in [0,0.5]
- fmax : upper frequency bound of the analysis. fmax is a real scalar comprised in [0,0.5] and fmax > fmin
- N : number of analyzing voices geometrically sampled between minimum scale fmax/fmax and maximum scale fmax/fmin.
- wvlt_length : specifies the analyzing wavelet: 0: Mexican hat wavelet (real). The size of the wavelet is automatically fixed by the analyzing frequency Positive real integer: real Morlet wavelet of size 2*wvlt_length+1) at finest scale (1) Positive imaginary integer: analytic Morlet wavelet of size 2*|wvlt_length|+1) at finest scale 1. The corresponding wavelet transform is then complex. May be usefull for event detection purposes. Real valued vector: corresponds to the time samples waveform of any arbitrary bandpass function viewed as the analyzing wavelet at any given scale. Then, an approximation of the scaled wavelet versions is achieved using the Fast Mellin Transform (see dmt and dilate).
- wt : coefficient of the wavelet transform. X-coordinated corresponds to time (uniformly sampled), Y-coordinates correspond to frequency (or scale) voices (geometrically sampled between fmax (resp. 1) and fmin (resp. fmax / fmin). First row of wt corresponds to the highest analyzed frequency (finest scale).
- scale : analyzed scales (geometrically sampled between 1 and fmax /fmin
- f : analyzed frequencies (geometrically sampled between fmax and fmin . f corresponds to fmax/scale
- scalo : Scalogram coefficients (squared magnitude of the wavelet coefficients wt)
- wavescaled : If wvlt_length is a real or Imaginary pure scalar, then wavescaled equal wvlt_length . If wvlt_length is a vector (containing the waveform samples of an arbitrary analyzing wavelet), then wavescaled contains columnwise all scaled version of wvlt_length used for the analysis. In this latter case, first element of each column gives the effective time support of the analyzing wavelet at the corresponding scale. wavescaled can be used for reconstructing the signal (see icontwt)

ALGORITHM DETAILS :
The overall details of the algorithm are similar to those of contwt . The difference stems from the mirror operation applied to the signal before computing the wavelet transform to minimize border effects. At each scale j the analyzed signal is mirrored at its both extremities. The number of added samples at both sides is equal to scale(j)* wvlt_length (the half length of the analyzing wavelet at this particular scale). After convolution of the mirrored signal with the analyzing wavelet, the result is truncated to the actual size of the initial signal.

SEE ALSO: :
contwt, icontwt and cwt

EXAMPLE: :
Signal synthesis x = fbmlevinson(1024,0.8) ; Regular Wavelet transform [wt_nomirror,scale,f] = contwt(x,2(-6),2(-1),128,8) ; viewmat(abs(wt_nomirror),[1 1 24]) ; Compared with a mirrored wavelet transform [wt_mirror,scale,f] = contwtmir(x,2(-6),2(-1),128,0) ; viewmat(abs(wt_mirror),[1 1 24]) ;

15.0.48 contwtspec Continuous L2 wavelet based Legendre spectrum

Author: Paulo Goncalves

Estimates the multifractal Legendre spectrum of a 1-D signal from the wavelet coefficients of a L2 continuous decomposition

USAGE :

```
[alpha,f_alpha,logpart,tau] = contwtspec(wt,scale,Q[,FindMax,ChooseReg])
```

INPUT PARAMETERS :

- wt : Real or complex matrix [N_scale,N] Wavelet coefficients of a continuous wavelet transform (output of contwt or contwtmir))
- scale : real vector [1,N_scale] Analyzed scale vector
- Q : real vector [1,N_Q] Exponents of the partition function
- FindMax : 0/1 flag. FindMax = 0 : estimates the Legendre spectrum from all coefficients FindMax = 1 : estimates the Legendre spectrum from the local Maxima coefficients of the wavelet transform Default value is FindMax = 1
- ChooseReg : 0/1 flag or integer vector [1,N_reg], (N_reg <= N_scale) ChooseReg = 0 : full scale range regression ChooseReg = 1 : asks online the scale indices setting the range for the linear regression of the partition function. ChooseReg = [n1 ... nN_reg] : scale indices for the linear regression of the partition function.

OUTPUT PARAMETERS :

- alpha : Real vector [1,N_alpha], N_alpha <= N_Q Singularity support of the multifractal Legendre spectrum
- f_alpha : real vector [1,N_alpha] Multifractal Legendre spectrum
- logpart : real matrix [N_scale,N_Q] Log-partition function

- tau : real vector [1,N_Q] Regression function

SEE ALSO: :
contwt, cwtspec, cwt, dwtspec, FWT

EXAMPLE: :

```
N = 2048 ; H = 0.7 ; Q = linspace(-4,4,11) ;
[x] = fbmlevinson(N,H) ;
[wt,scale] = contwtmir(x,2^(-8),2^(-1),16,8) ;
[alpha,f_alpha,logpart,tau] = contwtspec(wt,scale,Q,1,1) ;
plot(alpha,f_alpha),
```

15.0.49 cwt Continuous Wavelet Transform

Author: Bertrand Guiheneuf

This routine computes the continuous wavelet transform of a real signal. Two wavelets are available: the Mexican Hat or the Morlet Wavelet.

USAGE :
[wt,scales,freqs]=cwt(sig,fmin,fmax,nbscales,[wvlt_length])

INPUT PARAMETERS :

- sig : real vector [1,n] or [n,1] Contains the signal to be decomposed.
- fmin : real positive scalar Lowest frequency of the wavelet analysis
- fmax : real positive scalar Highest frequency of the wavelet analysis
- nbscales : integer positive scalar Number of scales to compute between the lowest and the highest frequencies.
- wvlt_length : real positive scalar (optionnal) If equal to 0 or not specified, the wavelet is the Mexican Hat and its length is automaticaly choosen. Otherwise, Morlet's wavelet is used and it's length at scale 1 is given by wvlt_length

OUTPUT PARAMETERS :

- wt : complex matrix [nbscales,n] Wavelet transform. The first line is the finer scale (scale 1). It is real if the Mexican Hat has been used, complex otherwise.
- scales : real vector [1,nbscales] Scale corresponding to each line of the wavelet transform.
- freqs : real vector [1,nbscales] Frequency corresponding to each line of the wavelet transform.

DESCRIPTION :

PARAMETERS :
The wavelet transform of sig is computed via convolutions of dilated and translated versions of a single function called the "wavelet". The scales are given by the dilatation factor. As the scales are not absolute, the scale factor is determined through the specification of the minimum and maximum frequency of the decomposition considered as a time/frequency transform. The maximum frequency

might not be greater than the Nyquist Frequency i.e. 0.5 as the wavelet at this scale would be undersampled. The number of scales tells how many convolutions are computed. The bigger it is, the slower the transform is. The frequency (or scale) axis is logarithmicaly sampled. The resulting scales and frequencies values can be obtained as output parameters. The meaning of the wavelet length parameter is twofold. If non zero, it tells the routine to use a Morlet Wavelet and gives its length at scale 1 (maximum frequency). Otherwise (zero or not specified), the Mexican Hat is used. The resulting wavelet transform is then real but has a quite poor frequency resolution.

ALGORITHM DETAILS :

Convolutions are computed through discrete linear convolutions in time domain. No FFT is used. The linear filters are obtained by a sampling of the wavelet after dilatation. The signal is mirrored at its boundaries.

15.0.50 cwtspec — Continuous L1 wavelet based Legendre spectrum

Author: Paulo Goncalves

Estimates the multifractal Legendre spectrum of a 1-D signal from the wavelet coefficients of a L1 continuous decomposition

USAGE :

```
[alpha,f_alpha,logpart] = cwtspec(wt,scale,Q[,FindMax,ChooseReg])
```

INPUT PARAMETERS :

- wt : Real or complex matrix [N_scale,N] Wavelet coefficients of a continuous wavelet transform (output of cwt)
- scale : real vector [1,N_scale] Analyzed scale vector
- Q : real vector [1,N_Q] Exponents of the partition function
- FindMax : 0/1 flag. FindMax = 0 : estimates the Legendre spectrum from all coefficients FindMax = 1 : estimates the Legendre spectrum from the local Maxima coefficients of the wavelet transform Default value is FindMax = 1
- ChooseReg : 0/1 flag or integer vector [1,N_reg], (N_reg <= N_scale) ChooseReg = 0 : full scale range regression ChooseReg = 1 : asks online the scale indices setting the range for the linear regression of the partition function. ChooseReg = [n1 ... nN_reg] : scale indices for the linear regression of the partition function.

OUTPUT PARAMETERS :

- alpha : Real vector [1,N_alpha], N_alpha <= N_Q Singularity support of the multifractal Legendre spectrum
- f_alpha : real vector [1,N_alpha] Multifractal Legendre spectrum
- logpart : real matrix [N_scale,N_Q] Log-partition function
- tau : real vector [1,N_Q] Regression function

SEE ALSO: :

cwt, contwtspec, contwt, dwtspec

EXAMPLE: :

```
N = 2048 ; H = 0.7 ; Q = linspace(-4,4,11) ;
[x] = fbmlevinson(N,H) ;
[wt,scale] = cwt(x,2^(-8),2^(-1),16,8) ;
[alpha,f_alpha,logpart,tau] = cwtspec(wt,scale,Q,1,1) ;
plot(alpha,f_alpha),
```

15.0.51 cwttrack_all — Continuous L2 wavelet based Holder function estimation

Author: Paulo Goncalves

Estimates the Holder function of a signal from its continuous wavelet transform (L2 contwt). cwttrack_all merely runs cwttrack as many times as there are time samples to be analyzed

USAGE :

```
[HofT,whichT] = cwttrack_all(wt,scale,FindMax,ChooseReg,radius,DeepScale,dT)
```

INPUT PARAMETERS :

- wt : Real or complex matrix [N_scale,N] Wavelet coefficients of a continuous wavelet transform (output of contwt)
- scale : real vector [1,N_scale] Analyzed scale vector
- whichT : Integer whichT, when non zero specifies the time position on the signal where to estimate the local Holder exponent. When whichT is zero, the global scaling exponent (or LRD exponent) is estimated.
- FindMax : 0/1 flag. FindMax = 0 : estimates the Holder exponents (local or global) from all coefficients of the wavelet transform FindMax = 1 : estimates the Holder exponents (local or global) from the local Maxima coefficients of the wavelet transform Default value is FindMax = 1
- ChooseReg : 0/1 flag or integer vector [1,N_reg], (N_reg <= N_scale) ChooseReg = 0 : full scale range regression ChooseReg = 1 : scale range is choosed by the user, clicking with the mouse on a regression graph. ChooseReg = [n1 ... nN_reg] : imposes the scale indices for the linear regression of the wavelet coefficients versus scale in a log-log plot Default value is ChooseReg = 0
- radius : Positive integer. The local maxima line search is restricted to some neighbourhood of the analyzed point. Basically, this region is defined by the cone of influence of the wavelet. radius allows to modulate the width of the cone. Default value is cone = 8 .
- DeepScale : strictly positive integer. DeepScale tells the maxima line procedure how depth in scale to scan from step to step. Default value is DeepScale = 1

- dT 01 Integer. Sampling period for the Holder function estimate

OUTPUT PARAMETERS :

- HofT : Real scalar. Local or global Holder exponent estimated
- whichT Integer vector Time sampling vector

SEE ALSO: :
cwttrack
EXAMPLE: :

```
N = 2048 ;
[x] = GeneWei(N,linspace(0,1,N),1.2,1,1) ;
[wt,scale] = contwtmir(x,2^(-6),2^(-1),64,8*i) ;
[HofT,whichT] = cwttrack_all(wt,scale,1,0,8,1,(N/64)) ;
```

15.0.52 cwttrack — Continuous L2 wavelet based Holder exponent estimation

Author: Paulo Goncalves

Estimates the local or global Holder exponent of a 1-D signal from its L2 continuous wavelet transform (output of contwt(mir)). In some cases, the global Holder exponent can also be refered to as the long range dependance parameter

USAGE :

```
[HofT] = cwttrack(wt,scale,whichT,FindMax,ChooseReg,radius,DeepScale,Show)
```

INPUT PARAMETERS :

- wt : Real or complex matrix [N_scale,N] Wavelet coefficients of a continuous wavelet transform (output of contwt)
- scale : real vector [1,N_scale] Analyzed scale vector
- whichT : Integer whichT, when non zero specifies the time position on the signal where to estimate the local Holder exponent. When whichT is zero, the global scaling exponent (or LRD exponent) is estimated.
- FindMax : 0/1 flag. FindMax = 0 : estimates the Holder exponents (local or global) from all coefficients of the wavelet transform FindMax = 1 : estimates the Holder exponents (local or global) from the local Maxima coefficients of the wavelet transform Default value is FindMax = 1

- ChooseReg : 0/1 flag or integer vector [1,N_reg], (N_reg <= N_scale) ChooseReg = 0 : full scale range regression ChooseReg = 1 : scale range is choosed by the user, clicking with the mouse on a regression graph. ChooseReg = [n1 ... nN_reg] : imposes the scale indices for the linear regression of the wavelet coefficients versus scale in a log-log plot Default value is ChooseReg = 0
- radius : Positive integer. The local maxima line search is restricted to some neighbourhood of the analyzed point. Basically, this region is defined by the cone of influence of the wavelet. radius allows to modulate the width of the cone. Default value is cone = 8 .
- DeepScale : strictly positive integer. DeepScale tells the maxima line procedure how depth in scale to scan from step to step. Default value is DeepScale = 1
- Show 0/1 flag. Show = 1 : display the maxima line trajectory and the log-log regression graph Show = 0 : no display

OUTPUT PARAMETERS :

- HofT : Real scalar. Local or global Holder exponent estimated

ALGORITHM DETAILS :
The maxima line search follows the two steps:

- all local maxima are found using a standard gradient technique
- local maxima are connected along scales by finding the minimum Lobatchevsky distance between two consecutive maxima lying beneath the cone of influence.

SEE ALSO: :
cwttrack_all, contwtspec, contwt, dwtspec
EXAMPLE: :

```
N = 1024 ;
[x] = GeneWei(N,[ones(1,N/2)*0.2 ones(1,N/2)*0.8],2,1,1) ;
[wt,scale] = contwtmir(x,2^(-8),2^(-1),64,8*i) ;
HofT_1 = cwttrack(wt,scale,N/4,1,1)
HofT_2 = cwttrack(wt,scale,3*N/4,1,1)
```

15.0.53 dilate — Dilation of a signal

Author: Paulo Goncalves

Computes dilated/compressed version of a signal using Fast Mellin transform.

USAGE :
[sscaled,mellin,beta] = dilate(s,a,[fmin,fmax,N])
INPUT PARAMETERS :

- s : real vector [1,nt] or [nt,1] Time samples of the signal to be scaled.

- a : real strictly positive vector [1,N_scale] Dilation/compression factors. a < 1 corresponds to compression in time
- fmin : real scalar in [0,0.5] Lower frequency bound of the signal (necessary for the intermediate computation of the Mellin transform)
- fmax : real scalar [0,0.5] and fmax > Upper frequency bound of the signal (necessary for the intermediate computation of the Mellin transform)
- N : positive integer. number of Mellin samples.

OUTPUT PARAMETERS :

- sscaled : Real matrix with N_scale columns Each column j (for j = 1 .. N_scale) contains the dilated/compressed version of s by scale a(j). First element of each column gives the effective time support for each scaled version of s.
- mellin : complex vector [1,N] Mellin transform of s.
- beta : real vector [1,N] Variable of the Mellin transform mellin.

DESCRIPTION :

PARAMETERS :

- s : signal to be analyzed. Real or complex vector. Size of s should be odd. If even, a zero sample is appended at the end of the signal
- a scale factor. Maximum allowed scale is determined by the spectral extent of the signal to be compressed: the spectral extent of the compressed signal can not go beyond the Nyquist frequency (1/2). There is no theoretical limit for the minimum allowed scale, other than the computational cost.
- fmin : lower frequency bound of the analysis. fmin is real scalar comprised in [0,0.5]
- fmax : upper frequency bound of the analysis. fmax is a real scalar comprised in [0,0.5] and fmax > fmin
- N : number of Mellin samples. This number must be greater than some ammount determined by the spectral extent of the signal, to avoid aliasing in the Mellin domain.

ALGORITHM DETAILS :
This algorith uses a Fast Mellin Transform (dmt) to diagonalize the Scale operator. The algorithm runs as follows

- compute the Fourier-Mellin transform of the signal
- Multiply the result by a(-i.beta) (beta is the Mellin variable), for each values of scale a
- compute the inverse Fourier-Mellin transform to get the a-dilated version of s

SEE ALSO: :
dmt, idmt

EXAMPLE: :
Signal synthesis

```
x = morlet(0.1,32) ;
plot(x)
```

Dilation by a factor 2

```
[sscaled,mellin,beta] = dilate(x,2,0.01,0.5,256) ;
plot(sscaled(2:sscaled(1)))
```


Compression by a factor 2

```
[sscaled,mellin,beta] = dilate(x,1/2,0.01,0.5,256) ;
plot(sscaled(2:sscaled(1)))
```

15.0.54 dimR2d — Regularization dimension of the surface of a 2d function

Author: Francois Roueff

Computes the regularization dimension of the surface of a 2d function. Two kernels are available: the Gaussian or the Rectangle.

USAGE :
[dim, handlefig]=dimR(x,sigma,voices,Nmin,Nmax,kernel,mirror,reg,graphs)

INPUT PARAMETERS :

- x : Real or complex matrix [nt,pt] Space samples of the signal to be analyzed.
- sigma : Real positive number Standard Deviation of the noise. Its default value is null (noise-free)
- voices : Positive integer. number of analyzing voices. When not specified, this parameter is set to 128.
- Nmin : Integer in [2,nt/3] Lower scale bound (lower width) of the analysing kernel. When not specified, this parameter is set to around nt/12.
- Nmax : Integer in [Nmin,2nt/3] Upper scale bound (upper width) of the analysing kernel. When not specified, this parameter is set to nt/3.
- kernel : String specifies the analyzing kernel: "gauss": Gaussian kernel (default) "rect": Rectangle kernel
- mirror : Boolean

  specifies wether the signal is to be mirrorized for the analyse (default: 0).
- reg : Boolean

  specifies wether the regression is to be done by the user or automatically (default: 0).
- graphs : Boolean:

  specifies wether the regularized graphs have to be displayed (default: 0).

OUTPUT PARAMETERS :

- dim : Real Estimated regularization dimension.
- handlefig : Integer vector Handles of the figures opened during the procedure.

DESCRIPTION :
This function is the same as dimR but adapted to 2d signals. For a more complete explanation of the regularization dimension, one can refer to: "A regularization approach to fractionnal dimension estimation", F. Roueff, J. Levy-Vehel, submitted to Fractal 98 conference. The regularized graphs of x are computed via convolutions of x with dilated versions of the kernel at different scales. The lengths of the regularized graphs are computed via convolutions of x with the derivatives of the

dilated versions of the kernel. The regularization dimension is computed either via an automatic range regression or via a regression by hand on the loglog plot of the lengths versus scales. If sigma is strictly positive, an estimation of the lengths without noise is used for the regression. These lengths are displayed in red while those of the noisy signal are in black. They should seperate at fine scales. When one specifies the range regression, the loglog plot of the lengths versus scales appears. Above are either increments (when sigma is null) or a loglog plot of the noise prevalence in the lengths. One selects the scale range of the regression. In the case of noisefree signals, select a scale region with stable increments. In the case of a strictly positive sigma, select a scale region where the noise prevalence is not too close to 1 (0 in log10): it should correspond to an approximately linear region for the red estimations. The number of scales (voices) tells how many convolutions are computed. The bigger it is, the slower the computation is. The scale axis is geometrically sampled (i.e. its log is arithmetically sampled). The gaussian kernel should give a better result but the rectangle is faster. As a general rule, be careful of the size of the input signal and of the maximal size of the kernel (Nmax x Nmax) to avoid too long computing times.

SEE ALSO: :
cwttrack, cwtspec.

EXAMPLE: :
Signal synthesis

```
x = GeneWei(100,0.6,2,1.0,0);
y = GeneWei(100,0.4,3,1.0,1);
w = x'*y;
mesh(w);
```

Dimension of the graph with a regression by hand

```
[dim,H] = dimR2d(w,0,25,5,30,'gauss',0,1,0);
```

Close the figures

```
close(H)
```

15.0.55 dmt ______ Discrete Mellin transform of a vector

Author: Paulo Goncalves

Computes the Fast Mellin transform of a signal.

USAGE :
[mellin,beta] = dmt(s,[fmin,fmax,N])

INPUT PARAMETERS :

- s : real vector [1,nt] or [nt,1] Time samples of the signal to be transformed.
- fmin : real scalar in [0,0.5] Lower frequency bound of the signal

- fmax : real scalar [0,0.5] and fmax > Upper frequency bound of the signal
- N : positive integer. number of Mellin samples.

OUTPUT PARAMETERS :

- mellin : complex vector [1,N] Mellin transform of s.
- beta : real vector [1,N] Variable of the Mellin transform mellin.

DESCRIPTION :

PARAMETERS :

- s : signal to be transformed. Real or complex vector.
- fmin : lower frequency bound of the analysis. fmin is real scalar comprised in [0,0.5]
- fmax : upper frequency bound of the analysis. fmax is a real scalar comprised in [0,0.5] and fmax > fmin
- N : number of Mellin samples. This number must be greater than some ammount determined by the spectral extent of the signal, to avoid aliasing in the Mellin domain.

ALGORITHM DETAILS :
The fast Mellin transform can be simply interpreted as a FFT applied to a geometrically sampled vector.

SEE ALSO: :
idmt, dilate

EXAMPLE: :
Signal synthesis

```
x = morlet(0.1,32) ;
plot(x)
```

Computation of the Mellin transform

```
[mellin,beta] = dmt(x,0.01,0.5,128) ;
plot(beta,abs(mellin))
```

15.0.56 dwtspec ______ Discrete wavelet based Legendre spectrum

Author: Paulo Goncalves

Estimates the multifractal Legendre spectrum of a 1-D signal from the wavelet coefficients of a discrete decomposition

USAGE :

```
[alpha,f_alpha,logpart] = dwtspec(wt,Q[,ChooseReg])
```

INPUT PARAMETERS :

- wt : Real vector [1,N] Wavelet coefficients of a discrete wavelet transform (output of FWT)
- Q : real vector [1,N_Q] Exponents of the partition function
- ChooseReg : 0/1 flag or integer vector [1,N_reg], (N_reg <= N_scale) ChooseReg = 0 : full scale range regression ChooseReg = 1 : asks online the scale indices setting the range for the linear regression of the partition function. ChooseReg = [n1 ... nN_reg] : scale indices for the linear regression of the partition function.

OUTPUT PARAMETERS :

- alpha : Real vector [1,N_alpha], N_alpha <= N_Q Singularity support of the multifractal Legendre spectrum
- f_alpha : real vector [1,N_alpha] Multifractal Legendre spectrum
- logpart : real matrix [N_scale,N_Q] Log-partition function
- tau : real vector [1,N_Q] Regression function

SEE ALSO: :
cwtspec, FWT, WTStruct, MakeQMF, flt, iflt
EXAMPLE: :

```
N = 2048 ; H = 0.3 ; Q = linspace(-4,4,11) ;
[x] = fbmlevinson(N,H) ;
qmf = MakeQMF('daubechies',2) ;
[wt] = FWT(x,log2(N),qmf) ;
[alpha,f_alpha,logpart,tau] = dwtspec(wt,Q,1) ;
plot(alpha,f_alpha),
```

15.0.57 fbmfwt ___________ Discrete wavelet based synthesis of a fBm

Author: Paulo Goncalves

Generates a 1/f Gaussian process from a discrete wavelet transform

USAGE :

```
[x] = fbmfwt(N,H,[noctave,Q,randseed]) ;
```

INPUT PARAMETERS :

- N : Positive integer Sample size of the fBm
- H : Real in [0,1] Holder exponent
- noctave : integer Maximum resolution level (should not exceeed log2(N))
- Q : real vector. Analyzing QMF (e.g. Q = MakeQMF('daubechies',4))
- randseed : real scalar Random seed generator

OUTPUT PARAMETERS :

- x : real vector [1,N] Time samples of the 1/f Gaussian process

ALGORITHM DETAILS :
Generates a 1/f Gaussian process by inverting a discrete wavelet transform. Step 1: generates y a [1,N] i.i.d. standard Gaussian noise Step 2: computes the discrete wavelet coefficients y Step 3: weigth the wavelet coefficients y with the corresponding scale power law Step 4: invert the weighted discrete wavelet transform
SEE ALSO: :
fbmlevinson, synth2, FWT, MakeQMF
EXAMPLE: :

```
Q = MakeQMF('daubechies',4) ;
[x] = fbmfwt(1024,0.5,10,Q) ;
[wt,scale,f] = contwt(x,2^(-8),2^(-1),64,8) ;
[H] = cwttrack(wt,scale,0,1,1,8,1,1) ;
```

15.0.58 fbmlevinson ____ Levinson synthesis of a fractional Brownian motion

Author: Paulo Goncalves

Generates a Fractional Brownian Motion (fBm) using Cholesky/Levinson factorization

USAGE :

```
[x,y,r] = fbmlevinson(N,H,[seed])
```

INPUT PARAMETERS :

- N : Positive integer Sample size of the fBm
- H : Real in [0,1] Holder exponent
- seed : real scalar Random seed generator

OUTPUT PARAMETERS :

- x : real vector [1,N] Time samples of the fBm
- y : real vector [1,N] Vector of N i.i.d. white standard Gaussian r.v.'s (input process of the generator)
- r : real vector [1,N] First row of the var/cov Toeplitz matrix R of the increment process w[k] = x[k+1] - x[k].

ALGORITHM DETAILS :
Generates a Fractional Brownian Motion using Levinson algorithm to triangularize the covariance matrix. R = E W * W' being the variance/covariance matrix of the fBm increment W[n ; shift] = X[n+shift] - X[n-shift], R = L*L', with L the lower left triangle matrix (Choleski or Levinson factorization). Then, we pose Z = L(-1) * W <=> W = L * Z with Rz the var/cov matrix of the process Z, Rz = E { Z * Z' } Rz = E { L(-1) * W * W' * (L(-1))' } Rz = L(-1) * R * (L(-1))' Rz = L(-1) * L * L' * (L(-1))' Rz = I (identity) Thus, Z is a white Gaussian noise with unit variance.
SEE ALSO: :
mbmlevinson
EXAMPLE: :

```
[x,y,r] = fbmlevinson(1024,0.8) ;
```

15.0.59 fft1d ________ Operates a column-wise direct or inverse FFT

Author: Paulo Goncalves

Operates a column-wise direct or inverse FFT on a matrix

USAGE :

```
Y = fft1d(X,DirInv) ;
```

INPUT PARAMETERS :

- X : Real or complex valued matrix [rx,cx]
- DirInv : +1 / -1 flag -1 Direct Fast Fourier Transform +1 Inverse Fast Fourier Transform

OUTPUT PARAMETERS :

- Y : Real or complex valued matrix [rx,cx] Each column of Y contains the FFT (resp IFFT) of the corresponding column of X

SEE ALSO: :
fft
EXAMPLE: :
Matrix synthesis:


```
t = linspace( 0,1,128 ) ;
f0 = [4 8 16 32]
X = sin( 2*%pi*t(:)*f0 ) ;
Y = abs( fft1d( X , -1 ) ) ;
Y = [Y(65:128,:) ; Y(1:64,:)] ;
f = linspace(-64,63,128) ;
plot2d(f(ones(4,1),:)',Y) ;
```

15.0.60 findWTLM ____________ Finds local maxima lines of a CWT

Author: Paulo Goncalves

Finds the local maxima of a continuous wavelet transform

USAGE :

```
[maxmap] = findWTLM(wt,scale[,depth])
```

INPUT PARAMETERS :

- wt : Complex matrix [N_scale,N] Wavelet coefficients of a continuous wavelet transform (output of FWT or contwt)
- scale : real vector [1,N_scale] Analyzed scale vector
- depth : real in [0,1] maximum relative depth for the peaks search. Default value is 1 (all peaks found)

OUTPUT PARAMETERS :

- maxmap : 0/1 matrix [N_scale,N] If maxmap(m,n) = 0 : the coefficient wt(m,n) is not a local maximum If maxmap(m,n) = 1 : the coefficient wt(m,n) is a local maximum

SEE ALSO: :
contwt, cwt
EXAMPLE: :

```
N = 2048 ; H = 0.3 ; Q = linspace(-4,4,11) ;
[x] = fbmlevinson(N,H) ;
[wt,scale] = cwt(x,2^(-6),2^(-1),36,0) ;
[maxmap] = findWTLM(wt,scale) ;
```

Vizualisation in Matlab:

```
viewWTLM(maxmap,scale,wt) ,
axis([1024 - 64 1024 + 64 0 log2(max(scale))]) ,
```

Vizualisation in Scilab: Not implemeted yet!...

15.0.61 flt — Fast Legendre transform

USAGE :

```
[u,s] = flt(x,y[,ccv])
```

INPUT PARAMETERS :

- x : real valued vector [1,N] samples support of the function y
- y : real valued vector [1,N] samples of function y = y(x)
- ccv : optional argument to choose between convex (ccv = 0) and concave (ccv = 1) envelope. Default value is ccv = 1 (concave)

OUTPUT PARAMETERS :

- u : real valued vector [1,M] Legendre transform of input y. Note that, since u stems from the envelope of y, in general M <= N.
- s : real valued vector [1,M] Variable of the Legendre transform of y.

DESCRIPTION :
Computes the Legendre transform of y y*(s) = sup_{x in X}[s.x - y(x)]
EXAMPLE: :
Function synthesis

```
m0 = .55 ; m1 = 1 - m0 ;
m2 = .95 ; m3 = 1 - m2 ;
q = linspace(-20,20,201) ;
tau1 = - log2(exp(q.*log(m0)) + exp(q.*log(m1))) ;
tau2 = - log2(exp(q.*log(m2)) + exp(q.*log(m3))) ;
tau3 = min(tau1 , tau2) ;
```

Legendre Transforms

```
[u1,s1] = flt(q,tau1) ;
[u2,s2] = flt(q,tau2) ;
[u3,s3] = flt(q,tau3) ;
```

Vizualisation - Matlab

```
plot(s1,u1,'g',s2,u2,'b',s3,u3,'r') ; grid ;
legend('u(tau1(q))','u(tau2(q))','u(tau3(q))') ;
```


Vizualisation - Scilab

```
plot2d(s3,u3,17) ; plot2d(s1,u1,18,'001') ; plot2d(s2,u2,19,'001') ;
```

AUTHOR : Paulo Goncalves

15.0.62 FWT2D — 2D Forward Disrete Wavelet Transform

Author: Bertrand Guiheneuf

This routine computes discrete wavelet transforms of a 2D real signal. Two transforms are possible : Orthogonal and Biorthogonal

USAGE :
[wt,index,length]=FWT2D(Input,NbIter,f1,[f2])
INPUT PARAMETERS :

- Input : real matrix [m,n] Contains the signal to be decomposed.
- NbIter : real positive scalar Number of decomposition Levels
- f1 : Analysis filter
- f2 : real unidimensional matrix [m,n] Synthesis filter

OUTPUT PARAMETERS :

- wt : real matrix Wavelet transform. Contains all the datas of the decomposition.
- index : real matrix [NbIter,4] Contains the indexes (in wt) of the projection of the signal on the multiresolution subspaces
- length : real matrix [NbIter,2] Contains the dimensions of each projection

DESCRIPTION :

INTRODUCTION :
The 2D discrete wavelet transform of Input is a projection on 2D multiresolution Spaces. The number of scales NbIter tells how many convolutions are computed. Each convolution is followed by a downsampling of the signal in both direction. For example, if the original matrix is (256,512), a resulting projection after the first iteration is (128,256). In 2D, there are 4 projections for each iteration corresponding to 2 projections in the row directions and 2 in the column direction. In each direction, the 2 projections are obtained through the convolutions with a low-pass filter and its associated high-pass filter. The projections are then HL HH LH LL where the first letter represents the filter used for the row filtering and the second letter is the filter used for column filtering. H is High-Pass filter and L Low-pass filter. Except for the last level where the four convolutions are kept, the LL output is always used as the input of the following iteration. Two types of filters are available : Quadrature Mirror Filters (Orthogonal) or Conjugate Quadrature Filters (Biorthogonal). Each one allows perfect reconstruction of the signal but only CQF pairs can be symetric. The advantage of QMF is that synthesis and reconstruction filters are the same.
PARAMETERS :
Input must be a real matrix. All dimensions are allowed but for a 1D vector, FWT is best suited. NbIter is the number of scales computed. It must be a positive integer greater than one and should

be smaller then log2(max(size(Input))) but this is not necessary. f1 is the linear FIR filter used for the analysis and might be obtained with MakeQMF() or MakeCQF() f2 is the linear FIR filter to use for the reconstruction. It is only necessary if f1 has been obtained with MakeCQF(). wt is the wavelet decomposition structure. The next two parametres must be used to read the wavelet coefficients. index contains the indexes of the first coefficient of each output. At each scale Scale, the output indexes are: index(Scale,1) : HL index(Scale,2) : LH index(Scale,3) : HH index(Scale,4) : LL on the last scale and 0 otherwise length contains the dimensions (height, width) of each output at a given Iteration.

ALGORITHM DETAILS :

Convolutions are computed through discrete linear convolutions in time domain. No FFT is used. The signal is mirrored at its boundaries. The wavelet structure (wt) is a vector and NOT a 2D matrix. It contains all the informatiosn for the reconstruction: wt(1) : height of the original signal wt(2) : width of the original signal wt(3) : Number of iterations wt(4) : Number of causal coefficients of the synthesis filter wt(5) : Number of anticausal coefficients of the synthesis filter then the Synthesis filter coefficients and finaly the wavelet coefficient are stored .

EXAMPLES :

a=rand(256,256); q=MakeQMF('daubechies',4); wt,wti,wtl = FWT2D(a,3,q); V=WT2Dext(wt,1,2); viewmat(V); Then to suppress the Lowest Frequency component and then reconstruction: index=0; for i=1:wtl(3,1), for j=1:wtl(3,2), wt(wti(3,4)+index)=0; end; end; result=IWT2D(wt);

REFERENCES :

Meyer Y. : Wavelets, Algorithms & Applications, SIAM. Meyer Y. : Ondelettes et Operateurs (I) : Hermann, Paris Daubechies I. : Ten Lectures on Wavelets, CBMS-NSF Regional conference series in applied mathematics.

SEE ALSO :

IWT2D, MakeQMF, MakeCQF, WT2Dext, WT2DVisu, WT2DStruct

15.0.63 FWT ______ 1D Forward Discrete Wavelet Transform

USAGE :

[wt,index,length]=FWT(Input,NbIter,f1,[f2])

INPUT PARAMETERS :

- Input : real matrix [1,n] or [n,1] Contains the signal to be decomposed.
- NbIter : real positive scalar Number of decomposition Levels to compute
- f1 : Analysis filter
- f2 : real unidimensional matrix [m,n] Synthesis filter. Useful only for biorthogonal transforms. If not precised, the filter f1 is used for the synthesis.

OUTPUT PARAMETERS :

- wt : real matrix Wavelet transform. Contains the wavelet coefficients plus other informations.
- index : real matrix [1,NbIter+1] Contains the indexes (in wt) of the projection of the signal on the multiresolution subspaces
- length : real matrix [1,NbIter+1] Contains the dimension of each projection

DESCRIPTION :

This routine computes discrete wavelet transforms of a 1D real signal. Two transforms are possible : Orthogonal and Biorthogonal

INTRODUCTION :

The discrete wavelet transform of Input is a projection on multiresolution Spaces. The number of scales NbIter tells how many convolutions are computed. Each convolution is followed by a

downsampling of the output. For example, if the original signal size is 500, the resulting size of the projection after the first iteration is 250. Each iteration consists then in two convolution/downsampling steps. One is high-pass (H) and the other one is low-pass (L). Except for the last iteration, the low-pass output is used as the input of the next iteration. Thus, only the high-pass is stored in wt except at the last iteration where both the outputs are stored. This explains why the wti array dimension is equal to NbIter+1. The last index index(NbIter+1) is the index of firts element of the last low-pass projection. Two types of filters are available : Quadrature Mirror Filters (Orthogonal) or Conjugate Quadrature Filters (Biorthogonal). Each one allows perfect reconstruction of the signal but only CQF pairs can be symetric. The advantage of QMF is that synthesis and reconstruction filters are the same.

PARAMETERS :

Input must be a real unidimensional matrix. NbIter is the number of scales computed. It must be a positive integer greater than one and should be smaller than log2(max(size(Input))) but this is not necessary. f1 is the linear FIR filter used for the analysis and might be obtained with MakeQMF() or MakeCQF() f2 is the linear FIR filter to use for the reconstruction. It is only necessary if f1 has been obtained with MakeCQF(). wt is the wavelet decomposition structure. The next two parametres must be used to read the wavelet coefficients. index contains the indexes of the first coefficient of each output. length contains the dimension of each output.

ALGORITHM DETAILS :

Convolutions are computed through discrete linear convolutions in time domain. No FFT is used. The signal is mirrored at its boundaries. The wavelet structure contains all the informations for the reconstruction: wt(1) : size of the original signal wt(2) : Number of iterations wt(3) : Number of causal coefficients of the synthesis filter wt(4) : Number of anticausal coefficients of the synthesis filter then the Synthesis filter coefficients and finaly the wavelet coefficient are stored .

EXAMPLES :

```
a=rand(1,250);
q=MakeQMF('daubechies',4);
wt,wti,wtl = FWT(a,6,q);
M=WTMultires(wt);
plot(M(2,:));
//Then to suppress the Lowest Frequency component and then reconstruction:
for i=1:wtl(6),wt(wti(6)+i-1)=0;end;
result=IWT(wt);
```

REFERENCES :

Meyer Y. : Wavelets, Algorithms & Applications, SIAM. Meyer Y. : Ondelettes et Operateurs (I) : Hermann, Paris Daubechies I. : Ten Lectures on Wavelets, CBMS-NSF Regional conference series in applied mathematics.

AUTHOR : Author: Bertrand Guiheneuf

SEE ALSO :

IWT, MakeQMF, MakeCQF, WTStruct, WTNbScales, WTMultires

15.0.64 gauss ______ Gaussian window

Author: Paulo Goncalves

Returns a Gaussian window

USAGE :

```
Win = gauss(N[,A])
```

INPUT PARAMETERS :

- N : Positive integer Number of points defining the time support of the window
- A : Real positive scalar Attenuation in dB at the end of the window (10(-A)). Default value is A = 2.

OUTPUT PARAMETERS :

- Win : real vector [1,N] Gaussian window in time.

SEE ALSO: :
mexhat, morlet

EXAMPLE: :

```
t = linspace(-1,1,128) ;
Win1 = gauss(128,2) ;
Win2 = gauss(128,5) ;
```

Vizualisation - Matlab

```
plot(t,win1,'b',t,win2,'r') ;
legend('Gaussian window 1','Gaussian window 2')
```

Vizualisation - Scilab

```
plot2d([t(:) t(:)],[Win1(:) Win2(:)],[17 19])
```

15.0.65 GeneWei ______ Generalized Weierstrass function

Author: Paulo Goncalves

Generates a Generalized Weierstrass function

USAGE :

```
[x,Ht]=GeneWei(N,ht,lambda,tmax,randflag)
```


INPUT PARAMETERS :

- N : Positive integer Sample size of the synthesized signal
- ht : Real vector or character string ht : real vector of size [1,N]: each element prescribes the local Holder regularity of the function. All elements of ht must be in the interval [0,1]. ht : character string : contains the analytic expression of the Holder trajectory (e.g. '0.5*sin(16*t) + 0.5')
- lambda : positive real Geometric progression of the Weierstrass function. Default value is lambda = 2.
- tmax : positive real Time support of the Weierstrass function. Default value is tmax = 1.
- randflag : flag 0/1 flag = 0 : deterministic Weierstrass function flag = 1 : random Weierstrass process Default value is randflag = 0

OUTPUT PARAMETERS :

- x : real vector [1,N] Time samples of the Weierstrass function
- Fj : real vector [1,N] Holder trajectory of the Weierstrass function

SEE ALSO: :

EXAMPLE: :
[x,Ht] = GeneWei(1024,'abs(sin(16*t))',2,1,0) ;

15.0.66 gifs2wave ____ wavelet coefficients from new GIFS coefficients

Author: Khalid Daoudi

Computes the wavelet c0efficients of the synthetic 1-D real signal from its new GIFS coefficients.

USAGE :
[wt_new]=gifs2wave(Ci_new,wt,wt_idx,wt_lg)

INPUT PARAMETERS :

- Ci_new : Real matrix Contains the new GIFS coefficients
- wt : Real matrix contains the wavelet coefficients (obtained using FWT)
- wt_idx : Real matrix [1,n] contains the indexes (in wt) of the projection of the signal on the multiresolution subspaces
- wt_lg : Real matrix [1,n] contains the dimension of each projection

OUTPUT PARAMETERS :

- wi_new : Real matrix Contains the new wavelet coefficients plus other informations.

DESCRIPTION :

PARAMETERS :

ALGORITHM DETAILS :

SEE ALSO: :

wave2gifs.

EXAMPLE: :

15.0.67 gifseg _ Replaces nodes of the diadic tree by a ceratin unique value.

Author: Khalid Daoudi

Replaces at each scale the left (resp. right) nodes of the diadic tree, associated to the GIFS coefficients, that belong to [cmin,cmax] by a ceratin unique value.

USAGE :

[Ci_new, marks, L]=gifseg(Ci,[cmin,cmax,epsilon])

INPUT PARAMETERS :

- Ci : Real matrix Contains the GIFS coefficients (obtained using FWT)
- cmin : Real scalar [1,n] Specifies the minimal value of the Ci's to be considered (cmin=0 by default)
- cmax : Real scalar [1,n] Specifies the maximal value of the Ci's to be considered (cmin=0 by default)
- epsilon : real scalar Specifies the maximal error desied on the Ci's approximation.

OUTPUT PARAMETERS :

- Ci_new : Real matrix Contains the the new GIFS coefficients.
- marks : Real vector Contains the segmentation marques. length(marks)-1 is the number of segmented parts.
- L : Real matrix A structure containing the left and right lambda_i's corresponding to each segmented part.

DESCRIPTION :

PARAMETERS :

ALGORITHM DETAILS :

SEE ALSO: :

hist, wave2gifs.

EXAMPLE: :

15.0.68 holder2d _ holder exponents of a measures defined on 2D real signal

USAGE :

[holder]=holder2d(Input,[Meas],[Res],[Ref],[RefMeas])

INPUT PARAMETERS :

- Input : real matrix [m,n] Contains the signal to be analysed.
- Meas : string Analysing measure. Must choosen be in {"sum", "var", "ecart", "min", "max", "iso", "riso", "asym", "aplat", "contrast", "lognorm", "varlog", "rho", "pow", "logpow", "frontmax", "frontmin", "diffh", "diffv", "diffmin", "diffmax"} (default : "sum")
- res : Number of resolutions used for the computation. (default : 1)
- Ref : real matrix [m,n] Contains the reference signal i.e. the signal on which the reference measure will be computed. Input and Ref must have the same dimensions.
- RefMeas : string Reference measure. (default : "sum")

OUTPUT PARAMETERS :

- holder : real matrix [m,n] Contains the Holder exponents.

DESCRIPTION :

INTRODUCTION :

This routines computes holder exponents by regressing an analysing measure (in a log-log plot) at different scales. Given a pixel, one defines a (square) window around it. The window size is called the resolution. The specified measure (or capacity) is then evaluated on the set defined by the window. For example, in the case of the "sum" measure, at resolution 2, a 5x5 square center on a pixel p0 is extracted from the input image. The mean of the gray levels of the obtained pixels defines the measure at pixel p0 and resolution 2. The measure type is thus given by the input parameter Meas whereas the actual measure is obtained through the input signal. In the case of a simple measure analysis, the regression is computed with respect to the size of the window, this corresponds to comparing the analysing measure to the Lebesgue measure. Nevertheless, it is possible to compute the regression by comparison with a reference measure given by the last two parameters.

Author : Author: Pascal Mignot - Bertrand Guiheneuf

15.0.69 icontwt ___ Inverse Continuous L2 wavelet transform

Author: Paulo Goncalves

Computes the inverse continuous wavelet transform: reconstructs a 1-D signal from its wavelet coefficients. The scale operator is unitary with respect to the L2 norm.

USAGE :

```
[x_back]=icontwt(wt,f,wl_length)
```

INPUT PARAMETERS :

- wt : Real or complex matrix [N,nt] coefficient of the wavelet transform
- f : real vector of size [N,1] or [1,N] which elements are in /[0,0.5], in decreasing order.
- wl_length : scalar or matrix specifies the reconstruction wavelet: 0: Mexican hat wavelet (real) Positive real integer: real Morlet wavelet of size 2*wl_length+1) at finest scale 1 Positive imaginary integer: analytic Morlet wavelet of size 2*wl_length+1) at finest scale 1 Real valued matrix with N columns: each column contains a dilated versions of an arbitrary synthesis wavelet.

OUTPUT PARAMETERS :

- x_back : Real or complex vector [1,nt] Reconstructed signal.

DESCRIPTION :

PARAMETERS :

- wt : coefficient of the wavelet transform. X-coordinated corresponds to time (uniformly sampled), Y-coordinates correspond to frequency (or scale) voices (geometrically sampled between fmax (resp. 1) and fmin (resp. fmax / fmin). First row of wt corresponds to the highest analyzed frequency (finest scale). Usually, wt is the output matrix wt of contwt .
- scale : analyzed scales (geometrically sampled between 1 and fmax /fmin. Usually, scale is the output vector scale of contwt .
- wl_length : specifies the synthesis wavelet: 0: Mexican hat wavelet (real). The size of the wavelet is automatically fixed by the analyzing frequency Positive real integer: real Morlet wavelet of size 2*wl_length+1) at finest scale (1) Positive imaginary integer: analytic Morlet wavelet of size 2*|wl_length|+1) at finest scale 1. The corresponding wavelet transform is then complex. May be usefull for event detection purposes. Real valued matrix: usually, for reconstruction wl_length is the output matrix wavescaled from contwt.

ALGORITHM DETAILS :
The reconstruction algorithm Inverse Wavelet Transform , proceeds by convolving the wavelet coefficients (obtained from contwt) by the synthesis wavelet. As we deal with continuous wavelet decomposition, the analyzing wavelet and its dual for reconstruction are the same (continuous basis). This operation is iterated at each analyzed scale j yielding N corresponding band-passed signal versions. The reconstructed signal is the scale weighting sum of these N vectors.

SEE ALSO: :
contwt, contwtmir

EXAMPLE: :
Signal synthesis

```
x = morlet(0.1,64) ;
t = 1:129 ;
```

A Morlet (of size 2*8+1 samples) wavelet transform

```
[wtMorlet,scale,f,scaloMorlet] = contwt(x,0.01,0.5,128,8) ;
viewmat(scaloMorlet,1:129,f,[1 1 24]) ;
```


Reconstruction with the same synthesis wavelet

```
[x_back] = icontwt(wtMorlet,f,8) ;
plot([t(:) t(:)],[x(:) x_back(:)]) ;
```

15.0.70 idmt — Inverse Discrete Mellin transform

Author: Paulo Goncalves

Computes the Inverse Fast Fourier-Mellin transform of a signal.

USAGE :
[x,t] = idmt(mellin,beta,[M])

INPUT PARAMETERS :

- mellin : complex vector [1,N] Fourier-Mellin transform to be inverted. For a correct inversion of the Fourier-Mellin transform, the direct Fourier-Mellin transform mellin must have been computed from fmin to 0.5 cycles per sec.
- beta : real vector [1,N] Variable of the Mellin transform mellin.
- M : positive integer. Number of time samples to be recovered from mellin.

OUTPUT PARAMETERS :

- x : complex vector [1,M] Inverse Fourier-Mellin transform of mellin.
- t : time variable of the Inverse Fourier-Mellin transform x.

DESCRIPTION :
The Inverse Fourier-Mellin transform can be viewed as an Inverse Fast Fourier Transform which result is assumed geometrically sampled. To recover the initial time signal, a Discrete Inverse Fourier Transform is applied to this geometrically Fourier representation. Important The Inverse Fourier-Mellin transform is correct only if the direct Fourier-Mellin transform has been computed from fmin to 0.5 cycles per sec.

SEE ALSO: :
dmt, dilate

EXAMPLE: :
Signal synthesis

```
x = morlet(0.1,32) ;
plot(x)
```

Computation of the Mellin transform

```
[mellin,beta] = dmt(x,0.01,0.5,128) ;
plot(beta,abs(mellin))
```

Computation of the Inverse Mellin transform

```
[y,t] = idmt(mellin,beta,65) ;
plot(t,abs(x-y))
```

15.0.71 integ ______ Approximate 1-D integral

Author: Paulo Goncalves

Approximate 1-D integral. integ(y,x) approximates the integral of y with respect to the variable x

USAGE :

```
SOM = integ(y[,x])
```

INPUT PARAMETERS :

- y : real valued vector or matrix [ry,cy] Vector or matrix to be integrated. For matrices, integ(Y) computes the integral of each column of Y
- x : row-vector [ry,1] Integration path of y. Default value is (1:cy)

OUTPUT PARAMETERS :

- SOM : real valued vector [1,cy] Finite sum approximating the integral of y w.r.t the integration path x

SEE ALSO: :
integ2d

EXAMPLE: :
Cumulative Normal Distribution

```
sigma = 1 ; N = 100 ;
x = logspace(log10(0.001),log10(3),N/2) ;
x = [ -fliplr(x) x ] ;
y = 1/sqrt(2*pi) * exp( -(x.^2)./2 ) ;
plot(x,y)
for n = 1:N
  PartialSom(n) = integ( y(1:n),x(1:n) ) ;
end
```

Vizualisation Matlab


```
plot(x,PartialSom,x,PartialSom,'or')
grid ; xlabel('x') ; title('\int_{-\infty}^{x} g(u) du')
```

Vizualisation Scilab

```
xbasc()
plot2d(x,PartialSom,-1)
```

15.0.72 isempty ______ Checks if a matrix is empty

Author: Paulo Goncalves

isempty True for empty matrix. isempty(x) returns %T if x is an empty array and %F otherwise. An empty array has no elements, that is prod(size(X))==0.

USAGE :

```
isempty(x)
```

INPUT PARAMETERS :

- x : Real or complex valued matrix [rx,cx]

SEE ALSO: :
all

15.0.73 IWT2D ______ 2D Inverse Disrete Wavelet Transform

Author: Bertrand Guiheneuf

This routine computes inverse discrete wavelet transforms of a 2D real signal. Two inverse transforms are possible : Orthogonal and Biorthogonal

USAGE :
[result]=IWT2D(wt,[f])

INPUT PARAMETERS :

- wt : real unidimensional matrix [m,n] Contains the wavelet transform (obtained with FWT2D).
- f : real unidimensional matrix [m,n] Synthesis filter.

OUTPUT PARAMETERS :

- result : real matrix Result of the reconstruction.

DESCRIPTION :

INTRODUCTION :
The wavelet transform is an invertible linear transform. This routines is the inverse transform. For details on the algorithm procedure, see FWT2D.
PARAMETERS :
Input must be a real matrix. It's generally obtained with FWT2D but can be created "by hand". In that case, it's strongly recommended to decompose a null signal with FWT2D. f is the linear FIR filter to use for the reconstruction. It is only necessary if the analysis filter had been obtained with MakeCQF() and the reconstruction filter had not been passed to FWT2D. If not specified the filter given in the synthesis is used. (See FWT2D). result is the reconstructed signal. It has the same dimensions as the original one.
ALGORITHM DETAILS :
Convolutions are computed through discrete linear convolutions in time domain. No FFT is used. The signal is mirrored at its boundaries. .SH Example a=rand(256,256); q=MakeQMF('daubechies',4); wt,wti,wtl = FWT2D(a,8,q); wt=abs(wt); result=IWT2D(wt);.SH References Meyer Y. : Wavelets, Algorithms & Applications, SIAM. Meyer Y. : Ondelettes et Operateurs (I) : Hermann, Paris Daubechies I. : Ten Lectures on Wavelets, CBMS-NSF Regional conference series in applied mathematics.
SEE ALSO :
FWT2D, MakeQMF, MakeCQF, WT2Dext, WT2DVisu

15.0.74 IWT ______ 1D Inverse Discrete Wavelet Transform

Author: Bertrand Guiheneuf

This routine computes inverse discrete wavelet transforms of a real signal. Two inverse transforms are possible : Orthogonal and Biorthogonal

USAGE :
[result]=IWT2D(wt,[f])
INPUT PARAMETERS :

- wt : real unidimensional matrix [m,n] Contains the wavelet transform (obtained with FWT).
- f : real unidimensional matrix [m,n] Synthesis filter.

OUTPUT PARAMETERS :

- result : real unidimensional matrix Result of the reconstruction.

DESCRIPTION :

INTRODUCTION :

The wavelet transform is an invertible linear transform. This routines is the inverse transform. For details on the algorithm procedure, see FWT.
PARAMETERS :
Input must be a real matrix. It's generally obtained with FWT but can be created "by hand". In that case, it's strongly recommended to decompose a null signal with FWT. f is the linear FIR filter to use for the reconstruction. It is only necessary if the analysis filter had been obtained with MakeCQF() and the reconstruction filter had not been passed to FWT. If not specified the filter given in the synthesis is used. (See FWT). result is the reconstructed signal. It has the same dimension as the original one.
ALGORITHM DETAILS :
Convolutions are computed through discrete linear convolutions in time domain. No FFT is used. The signal is mirrored at its boundaries. .SH Example a=rand(1,256); q=MakeQMF('daubechies',4); wt,wti,wtl = FWT(a,8,q); wt=abs(wt); result=IWT(wt);.SH References Meyer Y. : Wavelets, Algorithms & Applications, SIAM. Meyer Y. : Ondelettes et Operateurs (I) : Hermann, Paris Daubechies I. : Ten Lectures on Wavelets, CBMS-NSF Regional conference series in applied mathematics.
SEE ALSO :
FWT, MakeQMF, MakeCQF, WTMultires, WTStruct

15.0.75 Koutrouvelis ______ Stable Law parameters estimation (Koutrouvelis method)

Author: Lotfi Belkacem

This routine estimates parameters of a stable law using the Koutrouvelis (1985) method.

USAGE :
[alpha,beta,mu,gamma]=Koutrouvelis(data)
INPUT PARAMETERS :

- proc : real vector [size,1] corresponding to the data sample.

OUTPUT PARAMETERS :

- alpha : real positive scalar between 0 and 2. This parameter is often referred to as the characteristic exponent.
- beta : real scalar between -1 and +1. This parameter is often referred to as the skewness parameter.
- mu : real scalar This parameter is often referred to as the location parameter. It is equal to the expectation when alpha is greater than 1.
- gamma : real positive scalar. This parameter is often referred to as the scale parameter. It is equal to the standard deviation over two squared when alpha equal 2.

EXAMPLE :
[proc1,inc1]=sim_stable(1,0,0,1,10000); generates a standard 1-stable motion (Cauchy process). [alpha,beta,mu,gamma]=Koutrouvelis(inc1); estimates parameters of the previous simutated 1-stable random variable inc1.
REMARQUE :
Skewness and location parameters are badly estimated with this methode.

15.0.76 lambdak — k's lambda functions for pseudoAW

Author: Paulo Goncalves

Computes the parametrizing function lambdak defining the Affine Wigner distributions.

USAGE :
[y] = lambdak(u,k)

INPUT PARAMETERS :

- u : real vector [1,n] Argument of the function lambdak.
- k : real scalar Parameter of the lambdak function. K=-1 corresponds to the Unterberger distribution; K=0 corresponds to the Bertrand distribution; K=0.5 corresponds to the D-Flandrin distribution; K=2 corresponds to the Wigner-Ville distribution on analytic signals.

OUTPUT PARAMETERS :

- y : real vector [1,n] Result of the function lambdak .

SEE ALSO: :
pseudoAW

EXAMPLE: :
x = linspace(-10,10,101) ; y0 = lambdak(x,-1) ; y1 = lambdak(x,2) ; plot(y0) plot(y1)

15.0.77 lepskiiap — lepskii adaptive procedure

Author: Christophe Canus

This C_LAB routine is an implementation of the Lepskii's adaptive procedure. This algorithm selects the "best" estimator which balances the bias-variance tradeoff in a sequence of noisy and biased estimators theta_hat_j of a non-random parameter theta with the assumption that when j increases, bias b_j increases as variance sigma2_j decreases.

USAGE :
[K_star,j_hat,I_c_j_min,I_c_j_max,E_c_j_hat_min,E_c_j_hat_max]= lepskiiap(theta_hat_j,[sigma2_j,K])

INPUT PARAMETERS :

- theta_hat_j : real vector [1,J] or [J,1] Contains the sequence of estimators.
- sigma2_j : strictly positive real vector [1,J] or [J,1] Contains the sequence of variances.
- K : strictly positive real scalar Contains the confidence constant.

OUTPUT PARAMETERS :

- K_star : strictly positive real scalar Contains the optimal confidence constant.
- j_hat : strictly positive real (integer) scalar Contains the selected index.
- I_c_j_min : real vector [1,J] Contains the minimum bounds of the confidence intervals.
- I_c_j_max : real vector [1,J] Contains the maximum bounds of the confidence intervals.
- E_c_j_hat_min : real scalar Contains the minimum bound of the selected intersection interval.
- E_c_j_hat_max : real scalar Contains the maximum bound of the selected intersection interval.

DESCRIPTION :

PARAMETERS :
The sequence of variances sigma_j must be stricly positive, decreasing when j increases and of the same size than theta_hat_j. When no sequence of variances is given as input or when it is uniformly equal to 0, the algorithm computes the sequence of variances as sigma2_j=1./j. The default value for epsilon is 1./[1:J]. The confidence constant K must be >=1. For the meaning of the ouput parameters, see next section.

ALGORITHM DETAILS :
Define the sequence of confidence intervals I_c_j=[theta_hat_j-K*sigma_j,theta_hat_j+K*sigma_j], the sequence of their decreasing intersections E_c_j and j_hat as the largest index j such as that E_c_j is non void. The best estimator with respect to the Lepskii's adaptive procedure is selected as theta_hat_j_hat in E_c_j_hat. The two parameters to be handled are the sequence of variances sigma2_j and the confidence constant K. sigma2_j can be any sequence dominating the estimator variance. Choosing a smaller K speeds up the selection and results to smaller j_hat.

EXAMPLES :

MATLAB :

```
T=33;
% linear model
f_t=linspace(0,1,T);
% jump for t=floor(3/4*T)
f_t(floor(3/4*T):T)=2*f_t(floor(3/4*T):T);
% Wiener process
W_t=randn(1,T);
sigma=.1;
Y_t=f_t+sigma*W_t;
subplot(2,1,1);
plot(f_t);hold on;plot(Y_t);
title('White noise model Y(t)');
xlabel('index: t');
ylabel('Y(t)=f(t)+\sigma W(t)');
% estimation for t=t_0=floor(T/2)
t_0=floor(T/2)+1;
Y_t=f_t+sigma*W_t;
for t=1:floor(T/2)
  f_hat_t(t)=mean(Y_t(t_0-t:t_0+t));
end
% Lespkii's adaptive procedure
[K_star,t_hat,I_c_t_min,I_c_t_max,E_c_t_hat_min,E_c_t_hat_max]=lepskiiap(f_hat_t,.005*1./[1:floo
% plot and disp results
plot(t_0,Y_t(t_0),'k*');
plot(t_0-t_hat,Y_t(t_0-t_hat),'kd');
plot(t_0+t_hat,Y_t(t_0+t_hat),'kd');
subplot(2,1,2);
```

```
plot(f_hat_t);
hold on;
plot(I_c_t_max,'r^');
plot(I_c_t_min,'gV');
title(['estimator \theta_t(t_0) vs. index t with t_0=',num2str(floor(T/2)+1)]);
xlabel('index: t');
ylabel('estimator: \theta_t(t_0)');
plot(t_hat,E_c_t_hat_min,'ko');
plot(t_hat,E_c_t_hat_max,'ko');
disp(['linear estimation of f_t for t=t_0=',num2str(t_0)]);
disp(['selected index t=',num2str(t_hat)]);
disp(['estimated f_t_0 in [',num2str(E_c_t_hat_min),',',num2str(E_c_t_hat_min),']']);
```

SCILAB :

```
//
```

REFERENCES :
To be published..SH See Also monolr (C_LAB routine).

15.0.78 linearlt ——— linear time legendre transform

Author: Christophe Canus

This C_LAB routine the Legendre transform of a function using the linear time Legendre transform algorithm.

USAGE :
[s,u_star_s]=linearlt(x,u_x)
INPUT PARAMETERS :

- x : real vector [1,N] or [N,1] Contains the abscissa.
- y : real vector [1,N] or [N,1] Contains the function to be transformed.

OUTPUT PARAMETERS :

- s : real vector [1,M] Contains the abscissa of the regularized function.
- u_star_s : real vector [1,M] Contains the Legendre conjugate function.

DESCRIPTION :

PARAMETERS :
The abscissa x and the function u_x to be transformed must be of the same size [1,N] or [N,1]. The abscissa s and the Legendre conjugate function u_star_s are of the same size [1,M] with M<=N.
ALGORITHM DETAILS :

The linear time Legendre transform algorithm is based on the use of a concave regularization before slopes' computation.
EXAMPLES :

MATLAB :

```
x=linspace(-5.,5.,1024);
u_x=-1+log(6+x);
plot(x,u_x);
% looks like a Reyni exponents function, isn't it ?
[s,u_star_s]=linearlt(x,u_x);
plot(s,u_star_s);
```

SCILAB :

```
//
```

REFERENCES :
None..SH See Also bbch (C_LAB routine).

15.0.79 mbmlevinson ——— Levinson synthesis of a multifractional Brownian motion

Author: Paulo Goncalves

Generates a Multi-Fractional Brownian Motion (mBm) using Cholesky/Levinson factorization

USAGE :

```
[x,y,r] = mbmlevinson(N,H,[seed])
```

INPUT PARAMETERS :

- N : Positive integer Sample size of the fBm
- H : Real vector [1,N] of character string H real vector: contains the Holder exponents at each time. Each element in [0,1]. H character string: analytic expression of the Holder function (e.g. 'abs(0.5 * (1 + sin(16 t)))')
- seed : real scalar Random seed generator

OUTPUT PARAMETERS :

- x : real vector [1,N] Time samples of the mBm
- y : real vector [1,N] Vector of N i.i.d. white standard Gaussian r.v.'s (input process of the generator)
- r : real matrix [N,N] Matrix containing columnwise each first row of the var/cov Toeplitz matrices R(n) of the non-stationary increment process w[n] = x[n+1] - x[n].

ALGORITHM DETAILS :
For each time n, a fbm process with constant Holder exponent H[n/] is synthesized over N points (see fbmlevinson). Only the sample at rank n is kept. As a result of this computationally expensive procedure, only small sample sizes of mBms can be generated (typically less than 1024 samples).
SEE ALSO: :
mbmlevinson
EXAMPLE: :

```
[x,y,r] = mbmlevinson(512,AtanH(512,2,1,0.5)) ;
plot(x) ;
```

15.0.80 McCulloch ___ Stable law parameters estimation (McCulloch method)

Author: Lotfi Belkacem

This routine estimates parameters of a stable law using the Mc-Culloch (1985) method.

USAGE :
[param,sd_param]=McCulloch(data)
INPUT PARAMETERS :

- data : real vector [size,1] corresponding to the data sample.

OUTPUT PARAMETERS :

- param : real vector [4,1] corresponding to the four estimated parameters of the fited stable law. the order is respectively alpha (characteristic exponent), beta (skewness parameter), mu (location parameter), gamma (scale parameter)
- sd_param : real vector [4,1] corresponding to estimated standard deviation of the four previous parameters.

EXAMPLE :
[proc1.5,inc1.5]=sim_stable(1.5,0,0,1,10000); generates a standard 1.5-stable motion. [param,sd_param]=McCulloch(estimates parameters of the previous simutated 1.5-stable random variable inc1.5 To visualize the estimates parameters or their sd-deviations use respectively param or sd_param. alpha=param(1), beta=param(2), mu=param(3), gamma=param(4). sd_alpha=sd_param(1), sd_alphabeta=sd_param(2), sd_alphamu=sd_param(3), sd_gamma=sd_param(4).
REMARQUE :
Skewness parameter and its sd-deviation estimations are not very accurate. Specially when the characteristic exponent is arround 2.

15.0.81 mcfg1d _ Continuous large deviation spectrum estimation on 1d measure

Author: Christophe Canus

This C_LAB routine estimates the continuous large deviation spectrum on 1d measure.

USAGE :
[alpha,fgc_alpha,[pc_alpha,epsilon_star,eta,alpha_eta_x]]= mcfg1d(mu_n,[S_min,S_max,J],progstr,ballstr,N,epsilon,co

INPUT PARAMETERS :

- mu_n : strictly positive real vector [1,N_n] or [N_n,1] Contains the 1d measure.
- S_min : strictly positive real scalar Contains the minimum size.
- S_max : strictly positive real scalar Contains the maximum size.
- J : strictly positive real (integer) scalar Contains the number of scales.
- progstr : string Contains the string which specifies the scale progression.
- ballstr : string Contains the string which specifies the type of ball.
- N : strictly positive real (integer) scalar Contains the number of Hoelder exponents.
- epsilon : strictly positive real vector [1,N] or [N,1] Contains the precisions.
- contstr : string Contains the string which specifies the definition of continuous spectrum.
- adapstr : string Contains the string which specifies the precision adaptation.
- kernstr : string Contains the string which specifies the kernel form.
- normstr : string Contains the string which specifies the pdf's normalization.
- I_n : strictly positive real vector [1,N_n] or [N_n,1] Contains the intervals on which the pre-multifractal 1d measure is defined.

OUTPUT PARAMETERS :

- alpha : real vector [1,N] Contains the Hoelder exponents.
- fgc_alpha : real matrix [J,N] Contains the spectrum(a).
- pc_alpha : real matrix [J,N] Contains the pdf('s).
- epsilon_star : strictly positive real matrix [J,N] Contains the optimal precisions.
- eta : strictly positive real vector [1,J] Contains the sizes.
- alpha_eta_x : strictly positive real matrix [J,N_n] Contains the coarse grain Hoelder exponents.

DESCRIPTION :

PARAMETERS :
The continuous large deviation spectrum (alpha,fgc_alpha) is estimated for J sizes eta_j and for the precision vector epsilon by taking into account the resolution of the 1d measure mu_n. The minimum size S_min sets the equivalent size eta_1 in the unit interval at which the first spectrum is estimated. eta_1 is equal to S_min*eta_n where eta_n is related to the resolution of the 1d measure (eta_n=N_n{-1} when all intervals are of equal size else it is max(|I_n|{-1}). It must be >=1. The default value for S_min is 1. The maximum size S_max sets the equivalent size eta_J in the unit interval at which the last spectrum is estimated. eta_J is equal to S_max*eta_n. It must be >=S_min. The default value for S_max is 1. The number of scales J sets the number of computed spectra. The bigger J is, the slower the computation is. It must be >=1. The default value for J is 1. The scale progression string progstr specifies the type of scale discretization. It can be

'dec' for decimated, 'log' for logarithmic or 'lin' for linear scale. The default value for progstr is 'dec'. The ball string ballstr specifies the type of ball B_eta(x). It can be 'asym' for asymmetric, 'cent' for centered or 'star' for three times bigger asymmetric ball. The default value for ballstr is 'asym'. The number N sets the discretization of the Hoelder exponents interval. They are linearly spaced between alpha_eta_min and alpha_eta_max which are the minimum and maximum values of the coarse grain Hoelder exponents at size eta. The bigger N is, the slower the computation is. It must be >=1. The default value for N is 100. The precision vector epsilon sets the precisions at which the spectrum is estimated. It must be of size [1,N] or [N,1]. When no precision vector is given as input or when it is uniformly equal to 0, the algorithm determines the optimal precisions vector epsilon_star. The default value for epsilon is zeros(1,N). The continuous string contstr specifies the definition of continuous spectrum. It can be equal to 'hnokern' for definition without precision and kernel or 'hkern' for definition with precision and kernel. The default value for contstr is 'hkern'. The precision adaptation string adapstr specifies the local adaptation of the precision w.r.t. the Hoelder exponents alpha. It can be equal to 'maxdev' for maximum deviation or 'maxadaptdev' for maximum adaptive deviation. The default value for adapstr is 'maxdev'. The kernel string kernstr specifies the kernel. It can be equal to 'box' for boxcar, 'tri' for triangle, 'mol' for mollifier, 'epa' for epanechnikhov or 'gau' for gaussian kernel. The default value for kernstr is 'gau'. The normalization string normstr specifies the type of pdf's normalization conducted before double log-normalization. It can be equal to 'nonorm' for no normalization conducted, 'suppdf' for normalization w.r.t the supremum of pdf's, 'infsuppdf' for normalization w.r.t the infimum and the supremum of pdf's. The default value for normstr is 'suppdf'. The intervals vector I_n can be useful when the intervals on which the pre-multifractal 1d measure is defined are not of equal size (not implemented yet). The pdf of the coarse grain Hoelder exponents matrix or vector pc_alpha, the optimal precisions matrix or vector epsilon_star, the sizes vector eta and the coarse grain Hoelder exponents matrix or vector alpha_eta_x can be obtained as outputs parameters.

ALGORITHM DETAILS :

The coarse Hoelder exponents are estimated on each point x of the unit interval discretization by summing interval measures into a sliding window of size eta containing x (which corresponds to ball B_eta(x)). The probability density function pc_alpha is obtained by integrating horizontal sections.

EXAMPLES :

MATLAB :

```
% synthesis of pre-multifractal binomial measure: mu_n
% resolution of the pre-multifractal measure
n=10;
% parameter of the binomial measure
p_0=.4;
% synthesis of the pre-multifractal beiscovitch 1d measure
mu_n=binom(p_0,'meas',n);
% continuous large deviation spectrum estimation: fgc_alpha
%  minimum size, maximum size & # of scales
S_min=1;S_max=8;J=4;
% # of hoelder exponents, precision vector
N=200;epsilon=zeros(1,N);
% estimate the continuous large deviation spectrum
[alpha,fgc_alpha,pc_alpha,epsilon_star]=mcfg1d(mu_n,[S_min,S_max,J],'dec','cent',N,epsilon,'hker
% plot the continuous large deviation spectrum
plot(alpha,fgc_alpha);
title('Continuous Large Deviation spectrum');
xlabel('\alpha');
ylabel('f_{g,\eta}^{c,\epsilon}(\alpha)');
```


SCILAB :

```
// computation of pre-multifractal besicovitch measure: mu_n
// resolution of the pre-multifractal measure
n=10;
// parameter of the besicovitch measure
p_0=.4;
// synthesis of the pre-multifractal besicovitch 1d measure
[mu_n,I_n]=binom(p_0,'meas',n);
// continuous large deviation spectrum estimation: fgc_alpha
// minimum size, maximum size & # of scales
S_min=1;S_max=8;J=4;
// # of hoelder exponents, precision vector
N=200;epsilon=zeros(1,N);
// estimate the continuous large deviation spectrum
[alpha,fgc_alpha,pc_alpha,epsilon_star]=mcfg1d(mu_n,[S_min,S_max,J],'dec','cent',N,epsilon,'hker
// plot the Continuous Large Deviation spectrum
plot2d(a,f,[6]);
xtitle(["Continuous Large Deviation spectrum";" "],"alpha","fgc(alpha)");
```

REFERENCES :

To be published..SH See Also mch1d, fch1d, fcfg1d, cfg1d (C_LAB routines). MFAG_continuous, MFAG_epsilon, MFAG_eta, MFAG_epsilon_eta (Matlab and/or Scilab functions).

15.0.82 mdfl1d ______ Discrete Legendre spectrum estimation on 1d measure

Author: Christophe Canus

This routine estimates the discrete Legendre Spectrum on 1d measure.

USAGE :

[alpha,f_alpha]=mdfl1d(mu_n,N,n)

INPUT PARAMETERS :

- mu_n : strictly positive real vector [1,nu_n] Contains the pre-multifractal measure.
- N : strictly positive real (integer) scalar Contains the number of Hoelder exponents.
- n : strictly positive real (integer) scalar Contains the final resolution.

OUTPUT PARAMETERS :

- alpha : real vector [1,N] Contains the Hoelder exponents.
- f_alpha : real vector [1,N] Contains the dimensions.

DESCRIPTION :

PARAMETERS :

The discrete Legendre spectrum f_alpha is estimated on the finite finer resolution of the pre-multifractal 1d measure mu_n. The three steps of the estimatation are:

- estimation of the partition function;
- estimation of the Reyni exponents;
- estimation of the Legendre transform.

ALGORITHM DETAILS :

The discrete partition function is estimated by coarse-graining masses mu_n into non-overlapping boxes of increasing diameter (box method). If nu_n is a power of 2, 2n corresponds to the coarser scale. The reyni exponents are estimated by least square linear regression. The Legendre transform of the mass exponent function is estimated with the linear-time Legendre transform.

SEE ALSO :
mdzq1d,mdzq2d,reynitq,linearlt,mdfl2d.

15.0.83 mdfl2d ______ Discrete Legendre spectrum estimation on 2d measure

Author: Christophe Canus

This routine estimates the discrete Legendre spectrum on a pre-multifractal 2d measure.

USAGE :
[alpha,fl_alpha]=mdfl2d(mu_n,N,n)

INPUT PARAMETERS :

- mu_n : strictly positive real matrix [nux_n,nuy_n] Contains the pre-multifractal measure.
- N : strictly positive real (integer) scalar Contains the number of Hoelder exponents.
- n : strictly positive real (integer) scalar Contains the final resolution.

OUTPUT PARAMETERS :

- alpha : real vector [1,N] Contains the Hoelder exponents.
- fl_alpha : real vector [1,N] Contains the dimensions.

DESCRIPTION :

PARAMETERS :

The discrete Legendre spectrum fl_alpha is estimated on the finite finer resolution of the 2d measure mu_n. The three steps of the estimation are:

- estimation of the discrete partition function;
- estimation of the Reyni exponents;
- estimation of the Legendre transform.

ALGORITHM DETAILS :
The discrete partition function is estimated by coarse-graining masses mu_n into non-overlapping boxes of increasing diameter (box method). If nux_n and nuy_n are power of 2, 2n corresponds to the coarser scale. The Reyni exponents are estimated by least square linear regression. The Legendre transform of the mass exponent function is estimated with the linear-time Legendre transform.

SEE ALSO :
mdznq1d,mdznq2d,reynitq,linearlt,mdfl1d.

15.0.84 mdznq1d ______ Discrete partition function estimation on 1d measure

Author: Christophe Canus

This routine computes the discrete partition function on a pre-multifractal 1d measure.

USAGE :
[mznq]=mdznq1d(mu_n,n,q)

INPUT PARAMETERS :

- mu_n : strictly positive real vector Contains the pre-multifractal measure.
- n : strictly positive real (integer) vector Contains the resolutions.
- q : strictly positive real vector Contains the exponents.

OUTPUT PARAMETERS :

- mznq : real matrix [size(q),size(n)] Contains the partition function.

DESCRIPTION :

PARAMETERS :

The discrete partition function mznq is computed on the pre-multifractal 1d measure mu_n. The vector of resolutions n and the vector of exponents q sets the size of the output real matrix mznq to size(q)*size(n).

ALGORITHM DETAILS :

The discrete partition function mznq is computed by coarse-graining masses mu_n into non-overlapping boxes of increasing diameter (box method). If nu_n is a power of 2, n corresponds to the resolution.

SEE ALSO :
mdzq2d,reynitq,linearlt,mdfl1d,mdfl2d.

15.0.85 mdznq2d ______ Discrete partition function estimation on 2d measure

Author: Christophe Canus

This routine computes the discrete partition function on a pre-multifractal 2d measure.

USAGE :
[mznq]=mdznq2d(mu_n,n,q)

INPUT PARAMETERS :

- mu_n : strictly positive real matrix Contains the pre-multifractal measure.
- n : strictly positive real (integer) vector Contains the resolutions.
- q : strictly positive real vector Contains the exponents.

OUTPUT PARAMETERS :

- mznq : real matrix [size(q),size(n)] Contains the discrete partition function.

DESCRIPTION :

PARAMETERS :

The discrete partition function mznq is computed on the pre-multifractal 2d measure mu_n. The vector of resolutions n and the vector of exponents q sets the size of the output real matrix mznq to size(q)*size(n).

ALGORITHM DETAILS :

The discrete partition function mznq is computed by coarse-graining masses mu_n into non-overlapping boxes of increasing diameter (box method). If nux_n and nuy_n are power of 2, n corresponds to the resolution.

SEE ALSO :
mdznq1d,reynitq,linearlt,mdfl1d,mdfl2d.

15.0.86 mexhat ______ Mexican hat wavelet

Author: Paulo Goncalves

Computes a Mexican Hat wavelet (seconde derivative of the gaussian).

USAGE :
[wavelet,alpha] = mexhat(nu)

INPUT PARAMETERS :

- nu : real scalar between 0 and 1/2 Central (reduced) frequency of the wavelet.

OUTPUT PARAMETERS :

- wavelet : real vector [1,2*N+1] Mexican Hat wavelet in time.
- alpha : real scalar Attenuation exponent of the Gaussian enveloppe of the Mexican Hat wavelet.

SEE ALSO: :
morlet, contwt

EXAMPLE: :
wavelet synthesis

```
wavelet1 = mexhat(0.05) ;
wavelet2 = mexhat(0.2) ;
plot(wavelet1) ; pause
plot(wavelet2)
```


15.0.87 monolr ______ monovariate linear regression

Author: Christophe Canus

This C_LAB routine provides six different algorithms to proceed linear regression on monovariate data: least square, weighted least square, penalized least square, multiple least square, maximum likelyhood and Lepskii's adaptive procedure least square, in one sole routine.

USAGE :
[a_hat,[b_hat,y_hat,e_hat,sigma2_e_hat,optvarargout]= monolr(x,y,[lrstr,optvarargin])

INPUT PARAMETERS :

- x : real vector [1,J] or [J,1] Contains the abscissa.
- y : real vector [1,J] or [J,1] Contains the ordinates to be regressed.
- lrstr : string Contains the string which specifies the type of linear regression to be used.
- optvarargin : Contains optional variable input arguments. Depending on the choice of linear regression, the fourth parameter can be
- w : strictly positive real vector [1,J] or [J,1] If weighted least square is chosen, contains the weights.
- I : strictly positive real (integer) scalar If penalized least square is chosen, contains the number of iterations.
- sigma2_j : strictly positive real vector [1,J] or [J,1] If Lepskii's adaptive procedure least square is chosen, contains the sequence of variances.

  The fifth parameter can be
- m : real scalar If penalized least square is chosen, contains the mean of the normal weights.
- K : strictly positive real scalar If Lepskii's adaptive procedure least square is chosen, contains the confidence constant.

  The sixth parameter can be
- s : strictly positive real scalar If penalized least square is chosen, contains the variance of the normal weights.

OUTPUT PARAMETERS :

- a_hat : real scalar or vector [1,J] Contains the estimated slope.
- b_hat : real scalar or vector [1,J] Contains the estimated ordimate at the origin.
- y_hat : real vector [1,J] or [1,(J+2)*(J-1)/2] Contains the regressed ordinates.
- e_hat : real vector [1,J] or [1,(J+2)*(J-1)/2] Contains the residuals.
- sigma2_e_hat : real scalar Contains the residuals' variance (that is, the mean square error).
- optvarargout : Contains optional variable output arguments. If Lepskii's adaptive procedure least square is chosen, the parameters are
- K_star : strictly positive real scalar Contains the optimal confidence constant.
- j_hat : strictly positive real (integer) scalar Contains the selected index.
- I_c_j_min : real vector [1,J] Contains the minimum bounds of the confidence intervals.
- I_c_j_max : real vector [1,J] Contains the maximum bounds of the confidence intervals.
- E_c_j_hat_min : real scalar Contains the minimum bound of the selected intersection interval.
- E_c_j_hat_max : real scalar Contains the maximum bound of the selected intersection interval.

DESCRIPTION :

PARAMETERS :
The abscissa x and the ordinate y to be regressed with must be of the same size [1,J] or [J,1]. The linear regression string lrstr specifies the type of linear regression used. It can be 'ls' for least square, 'wls' for weighted least square, 'pls' for penalized least square, 'mls' for multiple least square (that is for j varying from 1 to J), 'ml' for maximum likelyhood, 'lapls' for Lepskii's adaptive procedure least square. The default value for lrstr is 'ls'. The weights w or the sequence of variances sigma2_j must be strictly positive and of size [1,J] or [J,1]. For the meaning of the variable optional input parameters sigma2_j and K, see lepskiiap (Lepskii's Adaptive Procedure) C_LAB routine's help. The number of iterations I must be >=2. The variance of the normal weights s must be strictly positive. If multiple least square, maximum likelyhood or Lepskii's adaptive procedure least square is chosen, the estimated slope a_hat and the ordinate at the origin b_hat are vectors of size [1,J], resp. the regressed ordinates y_hat and the residuals e_hat vectors are of size [1,(J+2)*(J-1)/2] (as they contains results for multiple linear regression, be aware of that when vizualising them :-), see examples), otherwise there are scalars, resp. vectors of size [1,J]. For maximum likelyhood, multiple least square linear regressions are proceeded in order to obtain variance estimates. Then maximum likelyhood linear regression is proceeded (corresponding results are found in a_hat(1), b_hat(1), y_hat(1:J), e_hat(1:J) and sigma2_e_hat(1), see examples). For the meaning of the variable optional output parameters K_star, j_hat, I_c_j_min, I_c_j_max, E_c_j_max, and E_c_j_max, see lepskiiap (Lepskii's Adaptive Procedure) C_LAB routine's help.

ALGORITHM DETAILS :
For the details of the Lepskii's adaptive procedure, see lepskiiap (Lepskii's Adaptive Procedure) C_LAB routine's help.

EXAMPLES :

MATLAB :

```
J=32;
x=1+linspace(0,1,J);
% Wiener process
W=randn(1,J);
epsilon=.1;
y=x+epsilon*W;
% least square
[a_hat,b_hat,y_hat,e_hat,sigma2_e_hat]=monolr(x,y);
plot(x);hold on;plot(y);plot(y_hat,'kd');
plot(epsilon.*W);hold on;plot(e_hat);
title('least square');
disp('type return');
pause;
clf;
% weighted least square
epsilon=linspace(.05,.5,J);
y=x+epsilon.*W;
[a_hat,b_hat,y_hat,e_hat,sigma2_e_hat]=monolr(x,y,'wls',1./epsilon);
plot(x);hold on;plot(y);plot(y_hat,'kd');
plot(epsilon.*W);hold on;plot(e_hat);
title('weighted least square');
disp('type return');
pause;
clf;
% penalized least square
[a_hat,b_hat,y_hat,e_hat,sigma2_e_hat]=monolr(x,y,'pls',30);
plot(x);hold on;plot(y);plot(y_hat);
```



```
title('penalized least square');
disp('type return');
pause;
clf;
% multiple least square
[a_hat,b_hat,y_hat,e_hat,sigma2_e_hat]=monolr(x,y,'mls');
plot(x);hold on;plot(y)
start_j=0;
hold on;
for j=2:J
  plot([1:j],y_hat(start_j+1:start_j+j),'k');
  disp(['estimated slope a_hat =',num2str(a_hat(j))]);
  disp('type return');
  pause;
  start_j=start_j+j;
  j=j+1;
end
clf
% maximum likelyhood
[a_hat,b_hat,y_hat,e_hat,sigma2_e_hat]=monolr(x,y,'ml');
plot(x);hold on;plot(y_hat(1:J),'kd');
plot(epsilon.*W);hold on;plot(e_hat(1:J));
clf;
% Lespkii's adaptive procedure
epsilon=.01;
y(1:16)=x(1:16)+epsilon*W(1:16);
y(16:32)=2*x(16:32)+epsilon*W(16:32);
[a_hat,b_hat,y_hat,e_hat,sigma2_e_hat,K_star,j_hat,I_c_j_min,I_c_j_max,E_c_j_hat_min,E_c_j_hat_m
plot(a_hat);
hold on;
plot(I_c_j_max,'r^');
plot(I_c_j_min,'gV');
title('LAP: estimator vs. index');
xlabel('index: j');
ylabel('estimator: \theta_j');
plot(j_hat,E_c_j_hat_min,'ko');
plot(j_hat,E_c_j_hat_max,'ko');
```

SCILAB :

```
//
```

REFERENCES :
To be published..SH See Also lepskiiap (C_LAB routine).

15.0.88 morlet ______ Morlet wavelet

Author: Paulo Goncalves

Computes a Morlet wavelet.

USAGE :
[wavelet,alpha] = morlet(nu,[N,analytic])

INPUT PARAMETERS :

- nu : real scalar between 0 and 1/2 Central (reduced) frequency of the wavelet
- N : Positive integer Half length of the wavelet transform. Default value corresponds to a total length of 4.5 periods.
- analytic : boolean (0/1) under Matalb or (%F/%T) under Scilab. 0 or %F : real Morlet wavelet 1 or %T : analytic Morlet wavelet

OUTPUT PARAMETERS :

- wavelet : real or complex vector [1,2*N+1] Morlet wavelet in time.
- alpha : real scalar Attenuation exponent of the Gaussian enveloppe of the Morlet wavelet.

SEE ALSO: :
mexhat, contwt

EXAMPLE: :
wavelet synthesis

```
wavelet1 = morlet(0.1,64) ;
wavelet2 = morlet(0.1) ;
plot(wavelet1) ; pause
plot(wavelet2)
```

15.0.89 multim1d ______ multinomial 1d measure synthesis

Author: Christophe Canus

This C_LAB routine synthesizes a large range of pre-multifractal measures related to the multinomial 1d measure (deterministic, shuffled, pertubated) and computes linked theoretical functions (partition sum function, Reyni exponents function, generalized dimensions, multifractal spectrum).

USAGE :
[varargout,[optvarargout]]=multim1d(b,p,str,varargin,[optvarargin])

INPUT PARAMETERS :

- b : strictly positive real (integer) scalar Contains the base of the multinomial.
- p : strictly positive real vector [1,b] Contains the weights of the multinomial.
- str : string Contains the type of ouput.
- varargin : variable input argument Contains the variable input argument.
- optvarargin : optional variable input arguments Contains optional variable input arguments.

OUTPUT PARAMETERS :

- varargout : variable output argument Contains the variable output argument.
- optvarargout : optional variable output argument Contains an optional variable output argument.

DESCRIPTION :

PARAMETERS :

The multinomial 1d measure is completly characterized by its base b and its weights p(i) (i=1 to b). The first parameter b must be >1. The second parameter must be a vector of size equal to b. The weights p(i) must be >0., <1. and their sum must be =1. (the case of p(i)=1/b corresponds to the Lebesgue measure) (i=1 to b). The third parameter str is a variable string used to determine the desired type of output. There are six suffix strings ('meas' for measure, 'cdf' for cumulative distribution function , 'pdf' for probability density function, 'part' for partition sum function, 'Reyni' for Reyni exponent function , 'spec' for multifractal spectrum) for the deterministic multinomial measure and two prefix strings for related measures ('shuf' for shuffled , 'pert' for pertubated) which can be added to the first ones to form composed strings. For example, 'shufmeas' is for the synthesis of a shuffled multinomial 1d pre-multifractal measure. Note that all combinaisons of strings are not implemented yet. When a string containing suffix string 'meas' is given as third input, a pre-multifractal measure mu_n (first output argument) is synthesized on the b-adic intervals I_n (second optional output argument) of the unit interval. In that case, the fourth input argument is a strictly positive real (integer) scalar n which contains the resolution of the pre-multifractal measure. The size of the output real vectors mu_n (and I_n if used) is equal to bn (so be aware the stack size ;-)). This option is implemented for the deterministic ('meas'), shuffled ('shufmeas') and pertubated ('pertmeas') multinomial 1d measure. When a string containing prefix 'shuf' is given as third input, the synthesis is made for a shuffled multinomial measure. At each level of the multiplicative cascade and for all nodes of the corresponding binary tree, the vector of weights p is shuffled. This option is implemented only for the multinomial 1d measure ('shufmeas'). When a string containing prefix 'pert' is given as third input, the synthesis is made for a pertubated multinomial measure. In that case, the fifth input argument is a strictly positive real scalar epsilon which contains the pertubation around weights. The weights are independant random variables identically distributed between p(i)-epsilon and p(i)+epsilon which must be >0., <1. (i=1 to b). This option is implemented only for the multinomial 1d measure ('pertmeas'). When replacing suffix string 'meas' with suffix string 'cdf', respectively suffix string 'pdf', the cumulative distribution function F_n, respectively the probability density function p_n, related to this pre-multifractal measure is computed (first output argument). When string 'part' is given as third input, the partition sum function znq of multifractal measure is computed as sole output argument. In that case, the fourth input argument is a strictly positive real (integer) vector vn which contains the resolutions, and the fifth input argument is a real vector q which contains the measure exponents. The size of the output real matrix znq is equal to size(q)*size(vn). This option is implemented only for the multinomial 1d measure. When string 'Reyni' is given as third input, the Reyni exponents function tq (and the generalized dimensions Dq if used) of the multifractal measure is computed as first output argument (and second optional output argument if used). In that case, the fourth input argument is a real vector q which contains the measure's exponents. The size of the output real vector tq is equal to size(q)). This option is implemented only for the multinomial 1d measure. When string 'spec' is given as third input, the multifractal spectrum f_alpha (second output argument) is synthesized on the Hoelder exponents alpha (first output argument). In that case, the fourth input argument is a strictly positive real (integer) scalar N which contains the number of Hoelder exponents. The size of both output real vectors alpha and f_alpha is equal to N. This option is implemented only for the multinomial 1d measure.

ALGORITHM DETAILS :

For the deterministic multinomial, the pre-multifractal measure synthesis algorithm is implemented is a iterative way (supposed to run faster than a recursive one). For the shuffled or the pertubated multinomial, the synthesis algorithm is implemented is a recursive way (to be able to pick up a i.i.d. r.v. at each level of the multiplicative cascade and for all nodes of the corresponding binary tree w.r.t. the given law). In the case of the pertubated multinomial, the weights of each node are normalised by their sum for the measure to remain conservative. Note that the shuffled multinomial 1d measure is not conservative.

EXAMPLES :

MATLAB :

```
b=3;
p=[.1 .3 .6];
n=8;
% synthesizes a pre-multifractal multinomial 1d measure
[mu_n,I_n]=multim1d(b,p,'meas',n);
plot(I_n,mu_n);
% synthesizes the cdf of a pre-multifractal shuffled multinomial 1d measure
F_n=multim1d(b,p,'shufcdf',n);
plot(I_n,F_n);
e=.09;
% synthesizes the pdf of a pre-multifractal pertubated multinomial 1d measure
p_n=multim1d(b,p,'pertpdf',n,e);
plot(I_n,p_n);
vn=[1:1:8];
q=[-5:.1:+5];
% computes the partition sum function of a multinomial 1d measure
znq=multim1d(b,p,'part',vn,q);
plot(-vn*log(2),log(znq));
% computes the Reyni exponents function of a multinomial 1d measure
tq=multim1d(b,p,'Reyni',q);
plot(q,tq);
N=200;
% computes the multifractal spectrum of a multinomial 1d measure
[alpha,f_alpha]=multim1d(b,p,'spec',N);
plot(alpha,f_alpha);
```

SCILAB :

```
b=3;
p=[.1 .3 .6];
n=8;
// synthesizes a pre-multifractal multinomial 1d measure
[mu_n,I_n]=multim1d(b,p,'meas',n);
plot(I_n,mu_n);
// synthesizes the cdf of a pre-multifractal shuffled multinomial 1d measure
F_n=multim1d(b,p,'shufcdf',n);
plot(I_n,F_n);
e=.09;
// synthesizes the pdf of a pre-multifractal pertubated multinomial 1d measure
```



```
p_n=multim1d(b,p,'pertpdf',n,e);
plot(I_n,p_n);
xbasc();
vn=[1:1:8];
q=[-5:.1:+5];
// computes the partition sum function of a multinomial 1d measure
znq=multim1d(b,p,'part',vn,q);
mn=zeros(max(size(q)),max(size(vn)));
for i=1:max(size(q))
   mn(i,:)=-vn*log(2);
end
plot2d(mn',log(znq'));
// computes the Reyni exponents function of a multinomial 1d measure
tq=multim1d(b,p,'Reyni',q);
plot(q,tq);
N=200;
// computes the multifractal spectrum of a multinomial 1d measure
[alpha,f_alpha]=multim1d(b,p,'spec',N);
plot(alpha,f_alpha);
```

REFERENCES :

"Multifractal Measures", Carl J. G. Evertsz and Benoit B. MandelBrot. In Chaos and Fractals, New Frontiers of Science, Appendix B. Edited by Peitgen, Juergens and Saupe, Springer Verlag, 1992 pages 921-953. "A class of Multinomial Multifractal Measures with negative (latent) values for the "Dimension" f(alpha)", Benoit B. MandelBrot. In Fractals' Physical Origins and Properties, Proceeding of the Erice Meeting, 1988. Edited by L. Pietronero, Plenum Press, New York, 1989 pages 3-29.

SEE ALSO :

binom, sbinom, multim2d, smultim1d, smultim2d (C_LAB routines). MFAS_measures, MFAS_dimensions, MFAS_spectra (Matlab and/or Scilab demo scripts).

15.0.90 multim2d multinomial 2d measure synthesis

Author: Christophe Canus

This C_LAB routine synthesizes a large range of pre-multifractal measures related to the multinomial 2d measure (deterministic, shuffled, pertubated) and computes linked theoretical functions (partition sum function, Reyni exponents function, generalized dimensions, multifractal spectrum).

USAGE :

[varargout,[optvarargout]]=binom(bx,by,p,str,varargin,[optvarargin])

INPUT PARAMETERS :

- bx : strictly positive real (integer) scalar Contains the abscissa base of the multinomial.
- by : strictly positive real (integer) scalar Contains the ordonate base of the multinomial.
- p : strictly positive real vector [by,bx] Contains the weights of the multinomial.
- str : string Contains the type of ouput.
- varargin : variable input argument Contains the variable input argument.
- optvarargin : optional variable input arguments Contains optional variable input arguments.

OUTPUT PARAMETERS :

- varargout : variable output argument Contains the variable output argument.
- optvarargout : optional variable output argument Contains an optional variable output argument.

DESCRIPTION :

PARAMETERS :

The multinomial 2d measure is completly characterized by its abscissa base bx, ordonate base by and its weights p(i) (i=1 to bx*by). The first two parameters bx and by must be >1. The third parameter must be a vector of size equal to bx*by. The weights p(i) must be >0., <1. and their sum must be =1. (the case of p(i)=1/(bx*by) corresponds to the Lebesgue measure) (i=1 to bx*by). The fourth parameter str is a variable string used to determine the desired type of output. There are six suffix strings ('meas' for measure, 'cdf' for cumulative distribution function , 'pdf' for probability density function, 'part' for partition sum function, 'Reyni' for Reyni exponent function , 'spec' for multifractal spectrum) for the deterministic multinomial measure and two prefix strings for related measures ('shuf' for shuffled , 'pert' for pertubated) which can be added to the first ones to form composed strings. For example, 'shufmeas' is for the synthesis of a shuffled multinomial 2d pre-multifractal measure. Note that all combinaisons of strings are not implemented yet. When a string containing suffix string 'meas' is given as fourth input, a pre-multifractal measure mu_n (first output argument) is synthesized on the bx-adic and by-adic intervals I_nx and I_ny (second and third optional output argument) of the unit square. In that case, the fifth input argument is a strictly positive real (integer) scalar n which contains the resolution of the pre-multifractal measure. The size of the output real matrix mu_n is equal to bxn*byn and the one of the output real vectors I_nx and I_ny (if used) is equal to bxn and byn (so be aware the stack size ;-)). This option is implemented for the deterministic ('meas'), shuffled ('shufmeas') and pertubated ('pertmeas') multinomial 2d measure. When a string containing prefix 'shuf' is given as fourth input, the synthesis is made for a shuffled multinomial measure. At each level of the multiplicative cascade and for all nodes of the corresponding binary tree, the vector of weights p is shuffled. This option is implemented only for the multinomial 2d measure ('shufmeas'). When a string containing prefix 'pert' is given as fourth input, the synthesis is made for a pertubated multinomial measure. In that case, the fifth input argument is a strictly positive real scalar epsilon which contains the pertubation around weights. The weights are independant random variables identically distributed between p(i)-epsilon and p(i)+epsilon which must be >0., <1. (i=1 to bx*by). This option is implemented only for the multinomial 2d measure ('pertmeas'). When replacing suffix string 'meas' with suffix string 'cdf', respectively suffix string 'pdf', the cumulative distribution function F_n, respectively the probability density function p_n, related to this pre-multifractal measure is computed (first output argument). When string 'part' is given as fourth input, the partition sum function znq of multifractal measure is computed as sole output argument. In that case, the fifth input argument is a strictly positive real (integer) vector vn which contains the resolutions, and the sixth input argument is a real vector q which contains the measure exponents. The size of the output real matrix znq is equal to size(q)*size(vn). This option is implemented only for the multinomial 2d measure. When string 'Reyni' is given as third input, the Reyni exponents function tq (and the generalized dimensions Dq if used) of the multifractal measure is computed as first output argument (and second optional output argument if used). In that case, the fifth input argument is a real vector q which contains the measure's exponents. The size of the output real vector tq is equal to size(q)). This option is implemented only for the multinomial 2d measure. When string 'spec' is given as fourth input, the multifractal spectrum f_alpha (second output argument) is synthesized on the Hoelder exponents alpha (first output argument). In that case, the fifth input argument is a strictly positive real (integer) scalar N which contains the number of Hoelder exponents. The size of both output real vectors alpha and f_alpha is equal to N. This option is implemented only for the multinomial 2d measure.

ALGORITHM DETAILS :

For the deterministic multinomial, the pre-multifractal measure synthesis algorithm is implemented is a iterative way (supposed to run faster than a recursive one). For the shuffled or the pertubated multinomial, the synthesis algorithm is implemented is a recursive way (to be able to pick up a i.i.d. r.v. at each level of the multiplicative cascade and for all nodes of the corresponding binary tree w.r.t. the given law). In the case of the pertubated multinomial, the weights of each node are normalised by their sum for the measure to remain conservative. Note that the shuffled multinomial 2d measure is not conservative.

EXAMPLES :

MATLAB :

```
bx=2;
by=3;
p=[.05 .1; .15 .2; .24 .26];
n=5;
% synthesizes a pre-multifractal multinomial 2d measure
[mu_n,I_nx,I_ny]=multim2d(bx,by,p,'meas',n);
mesh(I_nx,I_ny,mu_n);
% synthesizes the cdf of a pre-multifractal shuffled multinomial 2d measure
F_n=multim2d(bx,by,p,'shufcdf',n);
mesh(I_nx,I_ny,F_n);
e=.049;
% synthesizes the pdf of a pre-multifractal pertubated multinomial 2d measure
p_n=multim2d(bx,by,p,'pertpdf',n,e);
mesh(I_nx,I_ny,p_n);
vn=[1:1:8];
q=[-5:.1:+5];
% computes the partition sum function of a multinomial 2d measure
znq=multim2d(bx,by,p,'part',vn,q);
plot(-vn*log(2),log(znq));
% computes the Reyni exponents function of a multinomial 2d measure
tq=multim2d(bx,by,p,'Reyni',q);
plot(q,tq);
N=200;
% computes the multifractal spectrum of a multinomial 2d measure
[alpha,f_alpha]=multim2d(bx,by,p,'spec',N);
plot(alpha,f_alpha);
```

SCILAB :

```
bx=2;
by=3;
p=[.05 .1; .15 .2; .24 .26];
n=5;
// synthesizes a pre-multifractal multinomial 2d measure
[mu_n,I_nx,I_ny]=multim2d(bx,by,p,'meas',n);
plot3d(I_nx,I_ny,mu_n);
// synthesizes the cdf of a pre-multifractal shuffled multinomial 2d measure
```

```
F_n=multim2d(bx,by,p,'shufcdf',n);
plot3d(I_nx,I_ny,F_n);
e=.049;
// synthesizes the pdf of a pre-multifractal pertubated multinomial 2d measure
p_n=multim2d(bx,by,p,'pertpdf',n,e);
plot3d(I_nx,I_ny,p_n);
xbasc();
vn=[1:1:8];
q=[-5:.1:+5];
// computes the partition sum function of a multinomial 2d measure
znq=multim2d(bx,by,p,'part',vn,q);
mn=zeros(max(size(q)),max(size(vn)));
for i=1:max(size(q))
   mn(i,:)=-vn*log(2);
end
plot2d(mn',log(znq'));
// computes the Reyni exponents function of a multinomial 2d measure
tq=multim2d(bx,by,p,'Reyni',q);
plot(q,tq);
N=200;
// computes the multifractal spectrum of a multinomial 2d measure
[alpha,f_alpha]=multim2d(bx,by,p,'spec',N);
plot(alpha,f_alpha);
```

REFERENCES :

"Multifractal Measures", Carl J. G. Evertsz and Benoit B. MandelBrot. In Chaos and Fractals, New Frontiers of Science, Appendix B. Edited by Peitgen, Juergens and Saupe, Springer Verlag, 1992 pages 921-953. "A class of Multinomial Multifractal Measures with negative (latent) values for the "Dimension" f(alpha)", Benoit B. MandelBrot. In Fractals' Physical Origins and Properties, Proceeding of the Erice Meeting, 1988. Edited by L. Pietronero, Plenum Press, New York, 1989 pages 3-29. .SH See also binom, sbinom, multim1d, smultim1d, smultim2d (C_LAB routines). MFAS_measures, MFAS_dimensions, MFAS_spectra (Matlab and/or Scilab demo scripts).

15.0.91 nextpowQ _ Rounds a number to the up-nearest power of an integer

Author: Paulo Goncalves

Rounds a number x to the up-nearest power of an integer Q

USAGE :

```
[xup2Q,powQ] = nextpowQ(x[,Q])
```

INPUT PARAMETERS :

- x : Real positive number

- Q : Positive integer. Default value is Q = 2

OUTPUT PARAMETERS :

- xup2Q : Positive integer x rounded to the closest power of Q
- powQ : Positive integer xup2Q = powQQ.

SEE ALSO: :

log, log2

15.0.92 oscillsing _ Oscillating Singularity synthesis

Author: Paulo Goncalves

Generates oscillating singularities located in the interval [0 .. 1]

USAGE :

[x,Fj,Fs] = oscillsing(alpha,beta,sing_pos,N,show) ;

INPUT PARAMETERS :

- alpha : Real positive vector [1,n_sing] or [n_sing,1] Holder strenghts of the singularities
- beta : Real positive vector [1,n_sing] or [n_sing,1] Chirp exponents of the singularities
- sing_pos : Real vector [1,n_sing] or [n_sing,1] Location of the singularities in the interval [0 .. 1]
- N : Integer Sample size for the synthesized signal
- show : flag 0/1 flag = 0 : no display flag = 1 : displays the instantaneous frequencies and the synthesized signal

OUTPUT PARAMETERS :

- x : real vector [1,N] Time samples of the synthesized signal
- Fj : real matrix [N,n_sing] instantaneous frequencies (each column of Fj contains the frequency chirp of each singularity)
- Fs : real sampling frequency

SEE ALSO: :

EXAMPLE: :

[x,Fj,Fs] = oscillsing([1/2 1 2],[1 2 4],[-0.5 0 0.5],256,1) ;

15.0.93 prescrib _ Generation of signals with prescribed Holder function

Author: Khalid Daoudi

Using the GIFS method, this routine generates a continous function with prescribed Holder function, while interpolating a set of point.

USAGE :
[x,y]=prescrib(Interp_pts, Holder_funct, nbr_iter)

INPUT PARAMETERS :

- Interp_pts : Real matrix [n,2] Contains the interpolation points in the format : abscissa-ordinate.
- Holder_funct : Character string Specifies the Holder function you want to prescribe. It must have the form of compositions of matlab functions of variable t ('2*sqrt(1-t)' for instance). The use of the variable t is crucial. For shake of simplicity, this variable t is supposed to vary in [0,1].
- nbr_iter : integer Number of iteration wanted in the generation process of the GIFS attractor.

OUTPUT PARAMETERS :

- x : Real vector Contains the abscissa of the attractor graph.
- y : Real vector Contains the ordinates of the attractor graph.

DESCRIPTION :

PARAMETERS :

- Interp_pts is a real matrix [n,2] containing the cordinates of the interpolation points.
- Holder_funct is a character string specifying the Holder function you want to prescribe. This means that GIFS attrcator will have, at a point t, a Holder exponent equal to the value of this function at pint t.
- nbr_iter is the number of iteration wanted in the generation process of the GIFS attractor.
- x and y contain the cordinates of the GIFS attractor.

ALGORITHM DETAILS :
Generalized Iterated Functions Systems (GIFS) are a generalization of the usual IFS. This generalization consists in allowing the contarations to change at each step (scale) of the attractor generation process. We also allow their number and their support to change. Here, we use the GIFS to construct continuous function with prescribed local regularity. More precisely, if H(t) is the prescribed Holder function, then for each j=1,...,nbr_iter-1, and for each k=0,...,pow(m,j)-1, the GIFS coefficient c_kj is definied as : c_kj = pow(m,H(k*pow(m,-j))), where m+1 is the number of interpolation points. The resulting attractor is the graph of a continuous function F such that the Holder exponent of F, at each point t, is H(t). Moreover, if {(t_i, y_i), i=1,...,m+1} is the set of interpolation points, then F(t_i)=y_i for each i=1,...,m+1.

SEE ALSO: :
gifs and alphagifs

EXAMPLE: :
I = [0 0 1 0]; [x,y] = prescrib(I,'abs(sin(3*pi*t))',10); plot(x,y) [x,y] is the graph of a continuous function F which interpolates {(0,0); (0.5 1); (1,0)} and such that the Holder exponent of F, at each point t, is abs(sin(3*pi*t)).

15.0.94 pseudoAW — Pseudo affine Wigner distribution

Author: Paulo Goncalves

Computes a Pseudo Affine Wigner Distributions of a 1-D signal (real or complex).

USAGE :
[tfr,scale,f,wt] = pseudoAW(x,K,[wave,smooth,fmin,fmax,N]);

INPUT PARAMETERS :

- x : Real or complex vector [1,nt] or [nt,1] Time samples of the signal to be analyzed.
- K : real scalar Parameter of the pseudo affine Wigner distributions. K = -1 : pseudo Unterberger K = 0 : pseudo Bertrand K = 1/2 : pseudo D-Flandrin K = 2 : pseudo affine Wigner-Ville, etc ...
- wvlt_length : positive integer specifies the analyzing wavelet: 0: Mexican hat wavelet (real) Positive real integer: real Morlet wavelet of size 2*wvlt_length+1) at finest scale 1 Positive imaginary integer: analytic Morlet wavelet of size 2*wvlt_length+1) at finest scale 1 Default value is the Mexican hat wavelet (wvlt_length = 0)
- smooth : positive integer half length of the time smoothing window. SMOOTH = 0 corresponds to the Pseudo affine Wigner distribution with no time-smoothing. Default value is mooth = 0.
- fmin : real in [0,0.5] Lower frequency bound of the analysis. When not specified, this parameter forces the program to interactive mode.
- fmax : real in [0,0.5] and fmax > Upper frequency bound of the analysis. When not specified, this parameter forces the program to interactive mode.
- N : positive integer. number of analyzing voices. When not specified, this parameter forces the program to interactive mode.

OUTPUT PARAMETERS :

- tfr : Real matrix [N,nt] time-frequency distribution
- scale : real vector [1,N] analyzed scales
- f : real vector [1,N] analyzed frequencies
- wt : real or complex matrix [N,nt] matrix of the wavelet coefficients (intermediate step)

DESCRIPTION :

PARAMETERS :

- K : fixes the function lambda_K(u) = K ((exp(-u)-1)/(exp(-Ku)-1))(1/(k-1)) used in the generalized affine convolution to define the K-order pseudo affine Wigner distribution.
- smooth : fixes the ammount of smooth in time of the distribution. This ammount can vary continuously from an unsmoothed pseudo affine Wigner distribution up to a maximum smoothness corresponding to a scalogram (squared magnitude of the intermediate wavelet coefficients)
- N : number of analyzing voices geometrically sampled between minimum scale fmax/fmax and maximum scale fmax/fmin.

- tfr : Samples of the pseudo affine Wigner distribution. X-coordinated corresponds to time (uniformly sampled), Y-coordinates correspond to frequency (or scale) voices (geometrically sampled between fmax (resp. 1) and fmin (resp. fmax / fmin). First row of tfr corresponds to the highest analyzed frequency (finest scale).
- scale : analyzed scales (geometrically sampled between 1 and fmax /fmin
- f : analyzed frequencies (geometrically sampled between fmax and fmin . f corresponds to fmax/scale
- wt : coefficients of the intermediate-step wavelet transform. X-coordinated corresponds to time (uniformly sampled), Y-coordinates correspond to frequency (or scale) voices (geometrically sampled between fmax (resp. 1) and fmin (resp. fmax / fmin). First row of wt corresponds to the highest analyzed frequency (finest scale).

ALGORITHM DETAILS :

A pseudo affine Wigner distribution requires to compute a continuous wavelet transform first. For each time, the corresponding column of the wavelet transform is affine convolved (generalized affine convolution defined through function lambda_K(u)) with itself.

SEE ALSO: :

contwt, cwt and lambdak

EXAMPLE: :

Signal synthesis

```
x = morlet(0.35,32)+morlet(0.1,32) ;
```

K = -1 pseudo affine Wigner distribution with no time smoothing

```
[tfr,scale,f,wt] = pseudoAW(x,-1,12*i,0,0.01,0.5,128) ;
viewmat(tfr,1:length(x),f,[1 0 .5]) ;
```

K = -1 time smoothed pseudo affine Wigner distribution

```
[tfr,scale,f,wt] = pseudoAW(x,-1,12*i,3,0.01,0.5,128) ;
viewmat(tfr,1:length(x),f,[1 0 0]) ;
```

15.0.95 regdim _ Estimate the regularization dimension of a 1d or 2d sample.

Author: Francois Roueff

Computes the regularization dimension of the graph of a 1d or 2d sampled function. Two kernels are available: the Gaussian or the Rectangle.

USAGE :

```
dim = regdim(x,sigma,voices,Nmin,Nmax,kernel,mirror,reg,graphs)
```

INPUT PARAMETERS :

- x : 1d: Real vector [1,nt] or [nt,1] 2d: Real matrix [nt,pt] Time samples of the signal to be analyzed.
- sigma : Real positive number Standard Deviation of the noise. Its default value is null (noisefree)
- voices : Positive integer. number of analyzing voices. When not specified, this parameter is set to 128.
- Nmin : Integer in [2,nt/3] Lower scale bound (lower length) of the analysing kernel. When not specified, this parameter is set to 2.
- Nmax : Integer in [Nmin,2nt/3] Upper scale bound (upper length) of the analysing kernel. When not specified, this parameter is set to nt/3.
- kernel : String specifies the analyzing kernel: "gauss": Gaussian kernel (default) "rect": Rectangle kernel
- mirror : Boolean

  specifies wether the signal is to be mirrorized for the analyse (default: 0).
- reg : Boolean

  specifies wether the regression is to be done by the user or automatically (default: 0).
- graphs : Boolean:

  for one dimensional signals, it specifies wether the regularized graphs have to be displayed (default: 0). In two dimensional sugnals and for matlab only, all the regularized samples contours are plotted on a same figure.

OUTPUT PARAMETERS :

- dim : Real Estimated regularization dimension.
- handlefig (for Matlab only): Integer vector Handles of the figures opened during the procedure.

DESCRIPTION :

For a more complete explanation of the regularization dimension, one can refer to: "A regularization approach to fractionnal dimension estimation", F. Roueff, J. Levy-Vehel, submitted to Fractal 98 conference. The regularized graphs of x are computed via convolutions of x with dilated versions of the kernel at different scales. The lengths of the regularized graphs are computed via convolutions of x with the derivatives of the dilated versions of the kernel. The regularization dimension is computed either via an automatic range regression or via a regression by hand on the loglog plot of the lengths versus scales. If sigma is strictly positive, an estimation of the lengths without noise is used for the regression. These lengths are displayed in red while those of the noisy signal are in black. They should seperate at fine scales. When one specifies the range regression, the loglog plot of the lengths versus scales appears. Above are either increments (when sigma is null) or a loglog plot of the noise prevalence in the lengths. One selects the scale range of the regression. In the case of noisefree signals, select a scale region with stable increments. In the case of a strictly positive sigma, select a scale region where the noise prevalence is not too close to 1 (0 in log10): it should correspond to an approximately linear region for the red estimations. The number of scales (voices) tells how many convolutions are computed. The bigger it is, the slower the computation is. The scale axis is geometrically sampled (i.e. its log is arithmetically sampled). The gaussian kernel should give a better result but the rectangle is faster.

SEE ALSO: :

cwttrack, cwtspec.

EXAMPLE: :

```
//          1D:
// Signal synthesis
x = GeneWei(1024,0.6,2,1.0,0);
plot(x);
//Dimension of the graph with a regression by hand
dim = regdim(x,0,128,10,500,'gauss',0,1,1);
```

```
//          2D
// Signal synthesis
z = GeneWei(200,0.6,2,1.0,0);
y = GeneWei(200,0.4,3,1.0,0);
w = z'*y;
plot3d(linspace(0,1,200),linspace(0,1,200),w);
//Dimension of the graph with a regression by hand
dim = regdim(w,0,25,10,50,'gauss',0,1);
```

15.0.96 reynitq — Reyni exponents estimation

Author: Christophe Canus

This routine estimates the Reyni exponents on a partition function.

USAGE :
[tq,[Dq]]=reynitq(znq,n,q)

INPUT PARAMETERS :

- znq : strictly positive real matrix Contains the partition function.
- n : strictly positive real (integer) vector Contains the resolutions.
- q : strictly positive real vector Contains the exponents.

OUTPUT PARAMETERS :

- tq : real vector [1,size(q)] Contains the discrete Legendre Spectrum.
- Dq : real vector [1,size(q)] Contains the generalized dimensions.

DESCRIPTION :

PARAMETERS :

The mass exponents tq and the generalized dimensions Dq (if used) are computed on the partition function znq. The input real matrix znq must be of height size(q) and of width size(n).
The output real vectors tq and Dq (if used) are of size size(q).

ALGORITHM DETAILS :

The mass exponent function tq by least mean square linear-fit.

SEE ALSO :
mdzq1d,mdzq2d,linearlt,mdfl1d,mdfl2d.

15.0.97 sbinom — stochastic binomial measure synthesis

Author: Christophe Canus

This C_LAB routine synthesizes two types of pre-multifractal stochastic measures related to the binomial measure paradigm (uniform law and lognormal law) and computes linked multifractal spectrum.

USAGE :
[varargout,[optvarargout]]=sbinom(str,varargin,[optvarargin])

INPUT PARAMETERS :

- str : string Contains the type of ouput.
- varargin : variable input argument Contains the variable input argument.
- optvarargin : optional variable input arguments Contains optional variable input arguments.

OUTPUT PARAMETERS :

- varargout : variable output argument Contains the variable output argument.
- optvarargout : optional variable output argument Contains an optional variable output argument.

DESCRIPTION :

PARAMETERS :

The first parameter str is a variable string used to determine the desired type of output. There are four suffix strings ('meas' for measure, 'cdf' for cumulative distribution function , 'pdf' for probability density function, 'spec' for multifractal spectrum) and a two prefix strings for the type of stochastic measure ('unif' for uniform and 'logn' for lognormal) which must added to the first ones to form composed. For example, 'unifmeas' is for the synthesis of a uniform law binomial pre-multifractal measure and 'lognspec' is for the computation of the multifractal spectrum of a lognormal binomial measure. When a string containing suffix string 'meas' is given as second input, a pre-multifractal measure mu_n (first output argument) is synthesized on the dyadic intervals I_n (second optional output argument) of the unit interval. In that case, the third input argument is a strictly positive real (integer) scalar n which contains the resolution of the pre-multifractal measure. The size of the output real vectors mu_n (and I_n if used) is equal to 2n (so be aware the stack size ;-)). This option is implemented for the uniform law ('unifmeas') and the lognormal law ('lognmeas') binomial measures. When a string containing prefix 'unif' is given as second input, the synthesis or the computation is made for a uniform law binomial measure. In that case, the optional fourth input argument is a strictly positive real scalar epsilon which contains the pertubation around weight .5. The weight is an independant random variable identically distributed between epsilon and 1-epsilon which must be >0., <1. The default value for epsilon is 0. When a string containing prefix 'logn' is given as second input, the synthesis or the computation is made for a lognormal law binomial measure. In that case, the optional fourth input argument is a strictly positive real scalar sigma which contains the standard deviation of the lognormal law. When replacing suffix string 'meas' with suffix string 'cdf', respectively suffix string 'pdf', the cumulative distribution function F_n, respectively the probability density function p_n, related to this pre-multifractal measure is computed (first output argument). When a string containing suffix string 'spec' is given as second input, the multifractal spectrum f_alpha (second output argument) is synthesized on the Hoelder exponents alpha (first output argument). In that case, the third input argument is a strictly positive real (integer) scalar N which contains the number of Hoelder exponents. The size of both output real vectors alpha and f_alpha is equal to N. This option is implemented for the uniform law ('unifspec') and the lognormal law ('lognspec') binomial measures.

ALGORITHM DETAILS :

For the uniform and lognormal law binomial, the synthesis algorithm is implemented is a recursive way (to be able to pick up a i.i.d. r.v. at each level of the multiplicative cascade and for all nodes of the corresponding binary tree w.r.t. the given law). Note that the lognormal law binomial measure is not conservative.

EXAMPLES :

MATLAB :

```
n=10;
% synthesizes a pre-multifractal uniform Law binomial measure
[mu_n,I_n]=sbinom('unifmeas',n);
plot(I_n,mu_n);
s=1.;
% synthesizes the cdf of a pre-multifractal lognormal law binomial measure
F_n=sbinom('logncdf',n,s);
plot(I_n,F_n);
e=.19;
% synthesizes the pdf of a pre-multifractal uniform law binomial measure
p_n=sbinom('unifpdf',n,e);
plot(I_n,p_n);
N=200;
s=1.;
% computes the multifractal spectrum of the lognormal law binomial measure
[alpha,f_alpha]=sbinom('lognspec',N,s);
plot(alpha,f_alpha);
```

SCILAB :

```
n=10;
// synthesizes a pre-multifractal uniform Law binomial measure
[mu_n,I_n]=sbinom('unifmeas',n);
plot(I_n,mu_n);
s=1.;
// synthesizes the cdf of a pre-multifractal lognormal law binomial measure
F_n=sbinom('logncdf',n,s);
plot(I_n,F_n);
e=.19;
// synthesizes the pdf of a pre-multifractal uniform law binomial measure
p_n=sbinom('unifpdf',n,e);
plot(I_n,p_n);
N=200;
// computes the multifractal spectrum of the lognormal law binomial measure
[alpha,f_alpha]=sbinom('lognspec',N,s);
plot(alpha,f_alpha);
```

REFERENCES :

"A class of Multinomial Multifractal Measures with negative (latent) values for the "Dimension" f(alpha)", Benoit B. MandelBrot. In Fractals' Physical Origins and Properties, Proceeding of the Erice Meeting, 1988. Edited by L. Pietronero, Plenum Press, New York, 1989 pages 3-29.

"Limit Lognormal Multifractal Measures", Benoit B. MandelBrot. In Frontiers of Physics, Landau Memorial Conference, Proceeding of the Tel-Aviv Meeting, 1988. Edited by Errol Asher Gotsman, Yuval Ne'eman and Alexander Voronoi, New York Pergamon, 1990 pages 309-340.

SEE ALSO :

binom, multim1d, multim2d, smultim1d, smultim2d (C_LAB routines). MFAS_measures, MFAS_dimensions, MFAS_spectra (Matlab and/or Scilab demo scripts).

15.0.98 sgifs — Semi Generalized IFS generation

Author: Khalid Daoudi

This routine generates stochastical Semi-Generalized Iterated Functions Systems (SGIFS) attractors.

USAGE :

[x, y, Ci]=sgifs(Interp_pts, coefs, nbr_iter,law_type,var)

INPUT PARAMETERS :

- Interp_pts : Real matrix [n,2] Contains the interpolation points in the format : abscissa-ordinate.
- coefs : Real vector Contains the fundamental contractions ratios.
- nbr_iter : Integer Number of iterations wanted in the generation process of the SGIFS attractor.
- law_type : Character string Specifies the type of law desired. You have the choice between 'uniform' and 'normal'.
- var : Real scalar Rules the variance decrease across scales. At each scale j, the variance would be 1/pow(j,var).

OUTPUT PARAMETERS :

- x : Real vector Contains the abscissa of the attractor graph.
- y : Real vector Contains the ordinates of the attractor graph.
- Ci : Real vector Contains all the coefficients of the so generated GIFS.

DESCRIPTION :

PARAMETERS :

- Interp_pts is a real matrix [n,2] containing the interpolation points.
- coefs is a real vector containing the fundamental contractions ratios, i.e. coefs(1) (resp. coef(2)) would be the mean of the even (resp. odd) ceofficients.
- nbr_iter is the number of iterations wanted in the generation process of the SGIFS attractor.
- law_type is a character string which specifies the type of law desired for the GIFS coefficients.
- var is a real scalar ruling the variance decrease across scales. At each step j of the attractor generation, the variance of the chosen law would be 1/pow(j,var).
- [x,y] contains the resulting attractor.
- Ci is a real vector containing all the coefficients of the so generated GIFS.

ALGORITHM DETAILS :
Semi-Generalized Iterated Functions Systems (SGIFS) are a generalization of the usual IFS. This generalization consists in allowing the contarations to change at each step (scale) of the attractor generation process. Here, we use GIFS to construct stocastical SGIFS. More precisely, at each scale j, the GIFS coefficients c_kj, for k even (resp. odd), are a random variable of law law_type, of mean coefs(1) (resp. coefs(2)) and of variance 1/pow(j,var) Moreover, if {(t_i, y_i), i=1,...,n+1} is the set of interpolation points, then any realisation of the attractor is the graph of a continuous function F such that : F(t_i)=y_i for each i=1,...,n+1.

SEE ALSO: :
fif, alphagifs and prescrib

EXAMPLE: :
I = [0 0 1 0]; coefs = [.3 -.9]; [x,y,Ci] = sgifs(I,coefs,10,'uniform',1); plot(x,y)

15.0.99 sim_stable ________ Generation of stable random processes

Author: Lotfi Belkacem

This routine generates a stable random process and its increments using the Chambers, Mallows and Stuck (1976) algorithm.

USAGE :
[proc,inc]=sim_stable(alpha,beta,mu,gamma,size)

INPUT PARAMETERS :

- alpha : real positive scalar between 0 and 2. This parameter is often referred to as the characteristic exponent.
- beta : real scalar between -1 and +1. This parameter is often referred to as the skewness parameter.
- mu : real scalar This parameter is often referred to as the location parameter. It is equal to the expectation when alpha is greater than 1.
- gamma : real positive scalar. This parameter is often referred to as the scale parameter. It is equal to the standard deviation over two squared when alpha equal 2.
- size : integer positive scalar. size of the simulated sample.

OUTPUT PARAMETERS :

- proc : real vector [size,1] corresponding to the stable random process.
- inc : real vector [size,1] corresponding to the increments of the simulated process.

EXAMPLES :

EXAMPLE 1 :
[proc,inc]=sim_stable(2,0,0,1.4142136,5000); generates a standard stable random process with alpha=2, beta=0 (symmetric), mu=0 and gamma=1.4142 which coinside with a standard gaussian process (Brownian motion). To visualize the process or the increments use plot(proc) or plot(inc).

EXAMPLE 2 :
[proc,inc]=sim_stable(1.5,0,0,1,5000); generates a standard 1.5-stable motion

15.0.100 smultim1d ________ multinomial 1d measure synthesis -

Author: Christophe Canus

This C_LAB routine synthesizes two types of pre-multifractal stochastic measures related to the multinomial 1d measure (uniform law and lognormal law) and computes linked multifractal spectrum.

USAGE :
[varargout,[optvarargout]]=sbinom(b,str,varargin,[optvarargin])

INPUT PARAMETERS :

- b : strictly positive real (integer) scalar Contains the base of the multinomial.
- str : string Contains the type of ouput.
- varargin : variable input argument Contains the variable input argument.
- optvarargin : optional variable input arguments Contains optional variable input arguments.

OUTPUT PARAMETERS :

- varargout : variable output argument Contains the variable output argument.
- optvarargout : optional variable output argument Contains an optional variable output argument.

DESCRIPTION :

PARAMETERS :
The stochastic multinomial 1d measure is completly characterized by its base b. This first parameter must be >1.

The second parameter str is a variable string used to determine the desired type of output. There are four suffix strings ('meas' for measure, 'cdf' for cumulative distribution function q, 'pdf' for probability density function, 'spec' for multifractal spectrum) and a two prefix strings for the type of stochastic measure ('unif' for uniform and 'logn' for lognormal) which must added to the first ones to form composed. For example, 'unifmeas' is for the synthesis of a uniform law multinomial 1d pre-multifractal measure and 'lognspec' is for the computation of the multifractal spectrum of a lognormal multinomial 1d measure. When a string containing suffix string 'meas' is given as second input, a pre-multifractal measure mu_n (first output argument) is synthesized on the b-adic intervals I_n (second optional output argument) of the unit interval. In that case, the third input argument is a strictly positive real (integer) scalar n which contains the resolution of the pre-multifractal measure. The size of the output real vectors mu_n (and I_n if used) is equal to bn (so be aware the stack size ;-)). This option is implemented for the uniform law ('unifmeas') and the lognormal law ('lognmeas') multinomial 1d measures. When a string containing prefix 'unif' is given as second input, the synthesis or the computation is made for a uniform law multinomial 1d measure. In that case, the optional fourth input argument is a strictly positive real scalar epsilon which contains the pertubation around weight .5. The weight is an independant random variable identically distributed between epsilon and 1-epsilon which must be >0., <1. The default value for epsilon is 0. When a string containing prefix 'logn' is given as second input, the synthesis or the computation is made for a lognormal law multinomial 1d measure. In that case, the optional fourth input argument is a strictly positive real scalar sigma which contains the standard deviation of the lognormal law. When replacing suffix string 'meas' with suffix string 'cdf', respectively suffix string 'pdf', the cumulative distribution function F_n, respectively the probability density function p_n, related to this pre-multifractal measure is computed (first output argument). When a string containing suffix string 'spec' is given as second input, the multifractal spectrum f_alpha (second output argument) is synthesized on the Hoelder exponents alpha (first output argument). In that case, the third input argument is a strictly positive real (integer) scalar N which contains the number of Hoelder exponents. The size of both output real vectors alpha and f_alpha is equal to N. This option is implemented only for the lognormal law ('lognspec') multinomial 1d measures.

ALGORITHM DETAILS :

For the uniform and lognormal law multinomial 1d, the synthesis algorithm is implemented is a recursive way (to be able to pick up a i.i.d. r.v. at each level of the multiplicative cascade and for all nodes of the corresponding binary tree w.r.t. the given law). Note that the lognormal law multinomial 1d measure is not conservative.

EXAMPLES :

MATLAB :

```
n=10;
% synthesizes a pre-multifractal uniform Law multinomial 1d measure
[mu_n,I_n]=smultim1d(b,'unifmeas',n);
plot(I_n,mu_n);
s=1.;
% synthesizes the cdf of a pre-multifractal lognormal law multinomial 1d measure
F_n=smultim1d(b,'logncdf',n,s);
plot(I_n,F_n);
e=.19;
% synthesizes the pdf of a pre-multifractal uniform law multinomial 1d measure
p_n=smultim1d(b,'unifpdf',n,e);
plot(I_n,p_n);
N=200;
s=1.;
% computes the multifractal spectrum of the lognormal law multinomial 1d measure
[alpha,f_alpha]=smultim1d(b,'lognspec',N,s);
plot(alpha,f_alpha);
```

SCILAB :

```
n=10;
// synthesizes a pre-multifractal uniform Law multinomial 1d measure
[mu_n,I_n]=smultim1d(b,'unifmeas',n);
plot(I_n,mu_n);
s=1.;
// synthesizes the cdf of a pre-multifractal lognormal law multinomial 1d measure
F_n=smultim1d(b,'logncdf',n,s);
plot(I_n,F_n);
e=.19;
// synthesizes the pdf of a pre-multifractal uniform law multinomial 1d measure
p_n=smultim1d(b,'unifpdf',n,e);
plot(I_n,p_n);
N=200;
// computes the multifractal spectrum of the lognormal law multinomial 1d measure
```



```
[alpha,f_alpha]=smultim1d(b,'lognspec',N,s);
plot(alpha,f_alpha);
```

REFERENCES :

"A class of Multinomial Multifractal Measures with negative (latent) values for the "Dimension" f(alpha)", Benoit B. MandelBrot. In Fractals' Physical Origins and Properties, Proceeding of the Erice Meeting, 1988. Edited by L. Pietronero, Plenum Press, New York, 1989 pages 3-29. "Limit Lognormal Multifractal Measures", Benoit B. MandelBrot. In Frontiers of Physics, Landau Memorial Conference, Proceeding of the Tel-Aviv Meeting, 1988. Edited by Errol Asher Gotsman, Yuval Ne'eman and Alexander Voronoi, New York Pergamon, 1990 pages 309-340.

SEE ALSO :

binom, sbinom, multim1d, multim2d, smultim2d (C_LAB routines). MFAS_measures, MFAS_dimensions, MFAS_spectra (Matlab and/or Scilab demo scripts).

15.0.101 smultim2d ____________ multinomial 2d measure synthesis -

Author: Christophe Canus

This C_LAB routine synthesizes two types of pre-multifractal stochastic measures related to the multinomial 2d measure (uniform law and lognormal law).

USAGE :

[varargout,[optvarargout]]=sbinom(bx,by,str,varargin,[optvarargin])

INPUT PARAMETERS :

- bx : strictly positive real (integer) scalar Contains the abscissa base of the multinomial.
- by : strictly positive real (integer) scalar Contains the ordonate base of the multinomial.
- str : string Contains the type of ouput.
- varargin : variable input argument Contains the variable input argument.
- optvarargin : optional variable input arguments Contains optional variable input arguments.

OUTPUT PARAMETERS :

- varargout : variable output argument Contains the variable output argument.
- optvarargout : optional variable output argument Contains an optional variable output argument.

DESCRIPTION :

PARAMETERS :

The stochastic multinomial 2d measure is completly characterized by its abscissa base bx, ordonate base by. These first two parameters must be >1.

The third parameter str is a variable string used to determine the desired type of output. There are four suffix strings ('meas' for measure, 'cdf' for cumulative distribution function q, 'pdf' for probability density function, 'spec' for multifractal spectrum) and a two prefix strings for the type of stochastic measure ('unif' for uniform and 'logn' for lognormal) which must added to the first ones to form composed. For example, 'unifmeas' is for the synthesis of a uniform law multinomial 2d pre-multifractal measure and 'lognspec' is for the computation of the multifractal spectrum of a lognormal multinomial 2d measure. When a string containing suffix string 'meas' is given as third input, a pre-multifractal measure mu_n (first output argument) is synthesized on the bx-adic and by-adic intervals I_nx and I_ny (second and third optional output argument) of the unit square. In that case, the fourth input argument is a strictly positive real (integer) scalar n which contains the resolution of the pre-multifractal measure. The size of the output real matrix mu_n is equal to bxn*byn and the one of the output real vectors I_nx and I_ny (if used) is equal to bxn and byn (so be aware the stack size ;-)). This option is implemented for the uniform law ('unifmeas') and the lognormal law ('lognmeas') multinomial 2d measures. When a string containing prefix 'unif' is given as third input, the synthesis or the computation is made for a uniform law multinomial 2d measure. In that case, the optional fourth input argument is a strictly positive real scalar epsilon which contains the pertubation around weight .5. The weight is an independant random variable identically distributed between epsilon and 1-epsilon which must be >0., <1. The default value for epsilon is 0. When a string containing prefix 'logn' is given as third input, the synthesis or the computation is made for a lognormal law multinomial 2d measure. In that case, the optional fifth input argument is a strictly positive real scalar sigma which contains the standard deviation of the lognormal law. When replacing suffix string 'meas' with suffix string 'cdf', respectively suffix string 'pdf', the cumulative distribution function F_n, respectively the probability density function p_n, related to this pre-multifractal measure is computed (first output argument).

ALGORITHM DETAILS :

For the uniform and lognormal law multinomial 2d, the synthesis algorithm is implemented is a recursive way (to be able to pick up a i.i.d. r.v. at each level of the multiplicative cascade and for all nodes of the corresponding binary tree w.r.t. the given law). Note that the lognormal law multinomial 2d measure is not conservative.

EXAMPLES :

MATLAB :

```
n=5;
bx=2;
by=3;
% synthesizes a pre-multifractal uniform Law multinomial 2d measure
[mu_n,I_nx,I_ny]=smultim2d(bx,by,'unifmeas',n);
mesh(I_nx,I_ny,mu_n);
s=1.;
% synthesizes the cdf of a pre-multifractal lognormal law multinomial 2d measure
F_n=smultim2d(bx,by,'logncdf',n,s);
mesh(I_nx,I_ny,F_n);
e=.19;
% synthesizes the pdf of a pre-multifractal uniform law multinomial 2d measure
p_n=smultim2d(bx,by,'unifpdf',n,e);
mesh(I_nx,I_ny,p_n);
```

SCILAB :

```
n=5;
bx=2;
by=3;
// synthesizes a pre-multifractal uniform Law multinomial 2d measure
[mu_n,I_nx,I_ny]=smultim2d(bx,by,'unifmeas',n);
```



```
mesh(I_nx,I_ny,mu_n);
s=1.;
// synthesizes the cdf of a pre-multifractal lognormal law multinomial 2d measure
F_n=smultim2d(bx,by,'logncdf',n,s);
mesh(I_nx,I_ny,F_n);
e=.19;
// synthesizes the pdf of a pre-multifractal uniform law multinomial 2d measure
p_n=smultim2d(bx,by,'unifpdf',n,e);
mesh(I_nx,I_ny,p_n);
```

REFERENCES :

"A class of Multinomial Multifractal Measures with negative (latent) values for the "Dimension" f(alpha)", Benoit B. MandelBrot. In Fractals' Physical Origins and Properties, Proceeding of the Erice Meeting, 1988. Edited by L. Pietronero, Plenum Press, New York, 1989 pages 3-29. "Limit Lognormal Multifractal Measures", Benoit B. MandelBrot. In Frontiers of Physics, Landau Memorial Conference, Proceeding of the Tel-Aviv Meeting, 1988. Edited by Errol Asher Gotsman, Yuval Ne'eman and Alexander Voronoi, New York Pergamon, 1990 pages 309-340.

SEE ALSO :

binom, sbinom, multim1d, multim2d, smultim1d (C_LAB routines). MFAS_measures, MFAS_dimensions, MFAS_spectra (Matlab and/or Scilab demo scripts).

15.0.102 sortup ___ Sorts the elements of an array in increasing order

Author: Paulo Goncalves

Sorts the elements of an array in increasing order

USAGE :

```
[yup,kup] = sortup(x[,how])
```

INPUT PARAMETERS :

- x : Real valued array [rx,cx]
- how : option argument '*' : x is treated as x(:). sortup returns a [rx,cx] array 'c' : x is treated columnwise. sortup returns a [rx,cx] array which columns are sorted in increasing order 'r' : x is treated in row. sortup returns a [rx,cx] array which rows are sorted in increasing order Default value is '*'

OUTPUT PARAMETERS :

- yup : Real matrix [rx,cx] Sorted elements of x
- kup : Integer matrix [rx,cx] Indices of the sorted elements of x

SEE ALSO: :
sort
EXAMPLE: :

```
[y,x] = sort(rand(4,4)) ;
x
xSortAll = sortup(x,'*')
xSortCol = sortup(x,'c')
xSOrtRow = sortup(x,'r')
```

15.0.103 stable_cov ______ Covariation of two jointly symmetric Alpha-Stable random variables

Author: Lotfi Belkacem

This routine estimates the covariation of two jointly symmetric alpha-stable random variables.

USAGE :
[cov]=stable_cov(data1,data2)
INPUT PARAMETERS :

- data1 : real vector [size,1] corresponding to the the first data sample.
- data2 : real vector [size,1] corresponding to the second data sample.

OUTPUT PARAMETERS :

- sm : real scalar corresponding to the estimation the covariation of data1 on data2.

DESCRIPTION :
The covariation of two jointly symmetric alpha stable random variables is defined only for alpha between 1 and 2. It designed to replace the covariance when the latter is not defined (alpha<2). Unfortunately, it lacks some of the desirable properties of the covariance (not symmetric, ...). It is however, a useful quantity and appears naturally in many settings, for example, in the context of linear regression..SH Example for two given signals S1 and S2, cov=stable_cov(S1,S2); estimates the covariation of S1 on S2.

15.0.104 stable_sm ___ Spectral measure of a bivariate Stable random vector

Author: Lotfi Belkacem

This routine estimates a normalized spectral measure of a bivariate stable random vector.

USAGE :
[theta,sm]=stable_sm(data1,data2)
INPUT PARAMETERS :

- data1 : real vector [size,1] corresponding to the the first data sample.
- data2 : real vector [size,1] corresponding to the second data sample.

OUTPUT PARAMETERS :

- theta : real vector corresponding to the the input argument of the estimated spectral measure. Components of the vector theta are varying between 0 and 2*PI.
- sm : real vector corresponding to the estimation of the normalized spectral measure of the bivariate vector (data1,data2).

EXAMPLE :
for two given signals S1 and S2, [theta,sm]=stable_sm(S1,S2); estimates the normalized spectral measure of the data vector (S1,S2). To visualize it use plot(theta,sm).

15.0.105 stable_test ______ stable law conformicy test

Author: Lotfi Belkacem

This routine tests the stability property of a signal.

USAGE :
[param,sd_param]=stable_test(maxr,data)
INPUT PARAMETERS :

- maxr : integer positive scalar. maximum resolution witch depend on the size of the sample.
- data : real vector [size,1] corresponding to the data sample (increments of the signal).

OUTPUT PARAMETERS :

- param : real matrix [maxr,4] corresponding to the four estimated parameters of the fited stable law at each level of resolution. param(i,:), for i=1, ...maxr, gives respectively alpha(characteristic exponent), beta (skewness parameter), mu (location parameter), gamma (scale parameter) estimated at the resolution i.
- sd_param : real matrix [maxr,4] corresponding to the estimated standard deviations of the four previous parameters at each level of resolution. sd_param(i,:), for i=1, ...maxr, gives respectively standard deviation of alpha, beta, mu and gamma estimated at the resolution i.

DESCRIPTION :
The stability test consists on estimating parameters of a fited alpha-satble law at different level of resolution. the variable is said to be stable if the charateristic exponent alpha remains approximatively constant at different resolution, and the scale parameter follows a scaling law with exponent (1/alpha)-1. .SH Example under scilab type: [proc1_5,inc1_5]=sim_stable(1.5,0,0,1,20000); [param,sd_param]=stable_test(7,inc1_5); alpha=param(:,1); m=(1:7)'; lnm=log(m); plot2d(m,alpha,1,'111','alpha',[1 gamma=param(:,4); lngamma=log(gamma); plot(lnm,lngamma); [a,b,sig]=reglin(lnm',lngamma'); slope=a th_slope=1/1.5-1

- we generate a standard 1.5-stable motion and its increments.
- we test the stability property of the previous simutated 1.5-stable random variable "inc1_5" at 7 resolutions.
- we list estimated alpha at different scales.
- we visualize the stability of the shape parameter alpha.
- we list estimated gamma at different scales.
- we visualize the scaling law of the scale parameter gamma with a log-log plot in the space (scale,scale parameter).
- we compute the slope "a" of the fited line which will be compared to (1/alpha-1).

:

15.0.106 strfbm Structure Function of a Brownian Field

Author: B. Pesquet-Popescu (ENS-Cachan)

Creates the structure function of an isotropic fBm

USAGE :

```
[Y] = strfbm(x,y,H)
```

INPUT PARAMETERS :

- x : Real vector [1,N] vertical coordinate
- y : Real scalar [1,M] horizontal coordinate
- H : Real in [0,1] Hurst parameter

OUTPUT PARAMETERS :

- Y : Matrix [N,M] Matrix containing the values of the structure function

SEE ALSO: :
synth2

EXAMPLE: :

```
x = 1:128 ;
y = 1:128 ;
[Y] = strfbm(x,y,0.8) ;
```


15.0.107 symcori Symmetrization of a periodic 2D correlation field

Author: B. Pesquet-Popescu (ENS-Cachan)

Symmetrization of a periodic 2D correlation field

USAGE :

```
Ss = symcori(S) ;
```

INPUT PARAMETERS :

- S : Matrix [N/2+1 , N] Periodic 2D correlation field S(1:N/2+1,1:N) of a complex 2D NxN field. Values of S(1,N/2+2:N) may be arbitrary.

OUTPUT PARAMETERS :

- Ss : Matrix [N , N] Symetrized correlation field

SEE ALSO: :
synth2, strfbm

EXAMPLE: :

15.0.108 synth2 Stationary Increments 2D Process

Author: B. Pesquet-Popescu (ENS-Cachan)

Incremental Fourier synthesis method for processes with stationary increments of order (0,1) and (1,0)

USAGE :

```
[B] = synth2(M,H,core)
```

INPUT PARAMETERS :

- M : Positive integer Vertical/Horizontal dimension of the generated field
- H : Real in [0,1] parameter of the structure function (e.g. : Hurst parameter)
- core : string Name of the structure function of type core(x,y,H) with x,y : vertical/horizontal coordinates

OUTPUT PARAMETERS :

- B : real matrix [N,N] Synthesized random field

REFERENCES :
L.M. Kaplan, C.C. J Kuo : IEEE Tran. on IP, May 1996 (extended version).

SEE ALSO: :
fbmlevinson, fbmcwt, fbmfwt, mbmlevinson

EXAMPLE: :

```
[B] = synth2(128,0.8,'strfbm') ;
viewmat(B)
```

15.0.109 viewmat — Vizualisation of a matrix

USAGE :

```
viewmat(Z [,X,Y])
```

INPUT PARAMETERS :

- Z : Real valued matrix [ny,nx] 2-D matrix to be displayed
- X : Real vector [1,nx] or [nx,1] x-axis
- Y : Real vector [1,ny] or [ny,1] Controls the vertical axis. y forces the vertical axis to be numbered from bottom to top in the increasing order. When not specified, the coordinate system is set to its "Cartesian" axes mode. The coordinate system origin is at the lower left corner. The x axis is horizontal and is numbered from left to right. The y axis is vertical and is numbered from bottom to top.
- type = 0 : image
- type = 1 : pseudo color
- type = 2 : contour plot
- type = 3 : mesh plot
- type = 4 : shaded surface with lighting
- scale = 0 : linear dynamic
- scale = 1 : logarithmic dynamic
- level : scalar setting the minimum level of the display $0 < \text{level} < +1$ for linear scale 0 dB $<$ level $<$ Infty dB for logarithmic scale

  Scilab version: cmd is ineffective and frozen to [1 0 0] .

REMARK :
viewmat changes the color map

SEE ALSO: :
plot3d, grayplot

EXAMPLE: :


```
//Signal synthesis:
x = oscillsing(1,1,0,128) ;
X = x(:)*x(:)' ;
//Matrix vizualisation:

viewmat(abs(X))
```

Author : Author: Paulo Goncalves - Bertrand Guiheneuf

15.0.110 viewWTLM — Vizualises the local maxima lines of a CWT

Author: Paulo Goncalves

Displays the local maxima of a continuous wavelet transform

USAGE :

```
viewWTLM(maxmap[,scale,wt])
```

INPUT PARAMETERS :

- maxmap : 0/1 matrix [N_scale,N] Indicator matrix of the local wavelet coefficients maxima
- scale : real vector [1,N_scale] Analyzed scale vector
- wt : Complex matrix [N_scale,N] Wavelet coefficients of a continuous wavelet transform (output of FWT or contwt)

SEE ALSO: :
findWTLM, viewmat, contwt, cwt

EXAMPLE: :

```
N = 2048 ; H = 0.3 ; Q = linspace(-4,4,11) ;
[x] = fbmlevinson(N,H) ;
[wt,scale] = cwt(x,2^(-6),2^(-1),36,0) ;
[maxmap] = findWTLM(wt,scale)
```

Vizualisation in Matlab:

```
viewWTLM(maxmap,scale,wt) ,
axis([1024 - 64 1024 + 64 0 log2(max(scale))]) ,
```

Vizualisation in Scilab: Not implemeted yet!...

15.0.111 wave2gifs _ Computation of IFS coef. with Discrete Wavelet coefficients

Author: Khalid Daoudi

Computes the GIFS cefficients of a 1-D real signal as the ratio between (synchrounous) wavelets coefficients at successive scales. You have to compute the wavelet coefficients of the given signal (using FWT) before using wave2gifs.

USAGE :

[Ci, Ci_idx, Ci_lg, pc0, pc_ab]=wave2gifs(wt, wt_idx, wt_lg, [M0, a, b])

INPUT PARAMETERS :

- wt : Real matrix [1,n] Contains the wavelet coefficients (obtained using FWT).
- wt_idx : Real matrix [1,n] Contains the indexes (in wt) of the projection of the signal on the multiresolution subspaces (obtained also using FWT).
- wt_lg : Real matrix [1,n] Contains the dimension of each projection (obtained also using FWT).
- M0 : Real positive scalar If specified, each GIFS coefficient whose absolute value belong to]1,M0[will be replaced by 0.99 (keeping its signe).
- a,b : Real positive scalars The routine gives the percentage of the Ci's whose absolute value belong to]a,b[(if not specified,]a,b[=]0,2[).

OUTPUT PARAMETERS :

- Ci : Real matrix Contains the GIFS coefficients plus other informations.
- Ci_idx : Real matrix Contains the the indexes of the first Ci at each scale (the finest scale is 1).
- Ci_lg : Real matrix Contains the length of Ci's at each scale.
- pc0 : Real scalar Gives the percentage of vanishing Ci's
- pc_ab : Real scalar Gives the percentage of Ci's which belong to]a,b[

DESCRIPTION :

PARAMETERS :

- wt is a real matrix which is a structure containg the wavelet coefficients and other informations. It is obtained using FWT.
- wt_idx is a real vector which contains the indexes (in wt) of the first wavelet coefficient a each scale. For instance, wt(wt_idx(1) : wt_idx(2)-1) is a vector containing the wavelet coefficients a the finest scale.
- wt_lg is a real vector which contains the length of wavelet coefficients a each scale. For instance, wt_lg(1) is the number of the wavelet coefficients a the finest scale.
- M0 is a real positive scalar such that each GIFS coefficient (ci) whose absolute value belong to]1,M0[will be replaced by 0.99*signe(ci).
- a and b are two real positive scalars. The routine gives the percentage of the Ci's whose absolute values belong to]a,b[(if not specified,]a,b[=]0,2[).
- Ci is a real matrix which contains the GIFS coefficients, the size of the signal in Ci(lenght(Ci)) and the number of scales used in the wavelet decomposition in Ci(lenght(Ci)-1).

- Ci_idx is a real matrix which ontains the the indexes of the first Ci at each scale. For instance, Ci(Ci_idx(j) : Ci_idx(j)+ Ci_lg(j) - 1) is a vector containing the GIFS coefficients at scale j (the finest scale is j=1).
- Ci_lg is a real vector which contains the length of GIFS coefficients a each scale. For instance, Ci_lg(1) is the number of the wavelet coefficients a the finest scale.
- pc0 is a real scalar which gives the percentage of vanishing GIFS coefficients.
- pc_ab is a real scalar which gives the percentage of GIFS coefficients which belong to]a,b[.

ALGORITHM DETAILS :

SEE ALSO: :

FWT and MakeQMF.

EXAMPLE: :

15.0.112 WT2Dext _ Extract a Projection from a 2D WT

Author: Bertrand Guiheneuf

This routine extracts a projection from the wavelet transform of a 2D matrix.

USAGE :

[V]=WT2Dext(wt, Scale, Num)

INPUT PARAMETER :

- wt : real unidimensional matrix [m,n] Contains the wavelet transform (obtained with FWT2D).
- w Scale : real scalar Contains the scale level of the projection to extract.
- w Num : real scalar Contains the number of the output to extract in level Scale (between 1 and 4)

OUTPUT PARAMETER :

- V : real matrix [m,n] Contains the matrix to be visualized directly

DESCRIPTION :

INTRODUCTION :

At each scale, a wavelet transform contains 4 outputs (HL, HH, LH and HH at the last scale). This routine is used to extract a particular component at a desired scale.

PARAMETERS :

wt must be a real matrix. It's generally obtained with FWT2D. It contains the wavelet transform coefficients. Num is 1,2,3, or 4 (at the last scale). Each number coresponds to a particular 2D frequency component.

- 1 : HL High frequency in row direction, Low in column direction.
- 2 : HH High frequency in row direction, High in column direction.
- 3 : LH Low frequency in row direction, High in column direction.

- 4 : LL Low frequency in row direction, Low in column direction. Only for the last scale (equals 0 for the other scales).

  V the wavelet coefficents at scale Scale with fequency component given by Num

EXAMPLE :
a=rand(256,256); q=MakeQMF('daubechies',4); wt = FWT2D(a,8,q); V=WT2Dext(wt,2,2); viewmat(V);

SEE ALSO :
FWT2D, IWT2D, WT2DVisu,

15.0.113 WT2DStruct — Retrieve the Structure of a 2D DWT

Author: Bertrand Guiheneuf

This routine retrieve the structure informations contained in a 2D Wavelet Transform.

USAGE :
[ScIndex, ScLength]=WT2DStruct(wt)

INPUT PARAMETERS :

- wt : real unidimensional matrix [m,n] Contains the wavelet transform (obtained with FWT2D).

OUTPUT PARAMETERS :

- index : real matrix [NbIter,4] Contains the indexes (in wt) of the projection of the signal on the multiresolution subspaces
- length : real matrix [NbIter,2] Contains the dimensions of each projection

DESCRIPTION :

INTRODUCTION :
This routine is used to retreive the structure information of a wavelet transform. It must be used in all routine that might work on a wavelet transform whose structure is not passed as an imput parameter. (That should be the case of all routines taking a Wavelet Transform as input parameter to minimize the input).see FWT2D.

PARAMETERS :
Input must be a real matrix. It's generally obtained with FWT2D. It contains the wavelet transform. index contains the indexes of the first coefficient of each output. At each scale Scale, the output indexes are: index(Scale,1) : HL index(Scale,2) : LH index(Scale,3) : HH index(Scale,4) : LL on the last scale and 0 otherwise length contains the dimensions (height, width) of each output at a given Iteration.

EXAMPLE :
a=rand(256,256); q=MakeQMF('daubechies',4); wt = FWT2D(a,8,q); (a few days pass...) wti, wtl=WT2DStruct(wt); wtl

SEE ALSO :
FWT2D, IWT2D, WT2Dext, WT2DVisu

15.0.114 WT2DVisu — Visualise a 2D Multiresolution

Author: Bertrand Guiheneuf

This routine constructs a matrix that shows all the wavelet coefficients of a 2D matrix.

USAGE :
[V]=WT2DVisu(wt)

INPUT PARAMETER :

- wt : real unidimensional matrix [m,n] Contains the wavelet transform (obtained with FWT2D).

OUTPUT PARAMETER :

- V : real matrix [m,n] Contains a matrix to be visualized directly

DESCRIPTION :

INTRODUCTION :
This routine is used to display all the scales and all the frequency components of a wavelet transform.

PARAMETERS :
wt must be a real matrix. It's generally obtained with FWT2D. V the wavelet coefficents.

EXAMPLE :
a=rand(256,256); q=MakeQMF('daubechies',4); wt = FWT2D(a,8,q); V=WT2DVisu(wt); viewmat(V);

SEE ALSO :
FWT2D, IWT2D, WT2Dext,

15.0.115 WTMultires — Construct a 1D Multiresolution Representation

Author: Bertrand Guiheneuf

This routine constructs a matrix that shows the projections of the signal on each multiresolution subspace

USAGE :
[V]=WTMultires(wt)

INPUT PARAMETER :

- wt : real unidimensional matrix Contains the wavelet transform (obtained with FWT).

OUTPUT PARAMETER :

- V : real matrix [Nbiter,n] Contains the projections on the Multiresolution. Each line is a projection on a subspace different "low-pass" space Vj

DESCRIPTION :

INTRODUCTION :
This routine is used to display all the scales of a wavelet transform. The projections are different from the wavelet coefficients as they represent "filtered" signals. Here each projection

PARAMETERS :
wt must be a real matrix containing the wavelet coefficients but also misc informations such as the original signal dimension and the reconstruction filter. It's generally obtained with FW. V Is the matrix containing the projection of the signal (decomposed in wt) on each Multiresolution subspace. The Nbiter first ones are the projections on the details subspaces. The last one is the projection on the last trend subspace.

EXAMPLE :
x=0.1:0.005:1; s=(x.0.7) .* sin(x.(-2)) ; q1 q2=MakeCQF(1); wt = FWT(s,3,q1,q2); V=WTMultires(wt); plot(V');

SEE ALSO :
FWT, IWT, WTStruct,

15.0.116 WTStruct ______ Retrieve a 1D Discrete Wavelet Structure.

Author: Bertrand Guiheneuf

This routine retrieves the structure informations contained in a 1D Wavelet Transform.

USAGE :
[ScIndex, ScLength]=WT2DStruct(wt)

INPUT PARAMETERS :

- wt : real unidimensional matrix [1,n] Contains the wavelet transform (obtained with FWT).

OUTPUT PARAMETERS :

- index : real matrix [1,NbIter] Contains the indexes (in wt) of the projection of the signal on the multiresolution subspaces
- length : real matrix [1,NbIter] Contains the dimensions of each projection

DESCRIPTION :

INTRODUCTION :
This routine is used to retreive the structure information of a wavelet transform. It must be used in all routine that might work on a wavelet transform whose structure is not passed as an imput parameter. (That should be the case of all routines taking a Wavelet Transform as input parameter to minimize the input).

PARAMETERS :
Input must be a real matrix. It's generally obtained with FWT. It contains the wavelet transform. index contains the indexes of the first coefficient of each output. The first "NbIter" indexes are the

indexes (in wt) of the "high-pass" subspaces projections (Wj), the last one is the last "low-pass" projection (Vj); length contains the dimension of each output.

EXAMPLE :
a=rand(1,256); q=MakeQMF('daubechies',4); wt = FWT2D(a,8,q); wti, wtl=WTStruct(wt); wtl

SEE ALSO :
FWT2D, IWT2D, WT2Dext, WT2DVisu

Chapter 16

TCL/Tk interface

16.0.117 close _____ close a figure

Author: Bertrand Guiheneuf

This routine allows to close a tksci figure (window).

USAGE :
close([h[)

INPUT PARAMETER :

- h : integer Handle of the window to close

OUTPUT PARAMETERS :
None

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.

EXAMPLE :

```
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

16.0.118 figure _____ create a figure

Author: Bertrand Guiheneuf

This routine allows to create a tksci figure (window).

USAGE :
figure([h[, [prop1, value1 [...)

INPUT PARAMETER :

- o h : integer Handle of the window to create. If not specified, the first free handle is used
- o prop{1, 2 ...} : character string name of a property to set
- o val{1, 2 ...} : scilab object value to give to the corresponding property

OUTPUT PARAMETERS :
handle of the newly created window

DESCRIPTION :
This routine creates a tksci figure (toplevel window). If a handle is given, the figure corresponding to this handle is created . Otherwise, the window is created with the first free handle, that is the lowest integer not already used by a window. The property named 'position' allows to control the geometrical aspect of the control. It is a [1,4] real 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.

One can also set this property by giving a string where the fields are separated by a '|', ie "x|y|w|h".

EXAMPLE :

```
h=figure(3);
// creates  figure number 1.
uicontrol( h, 'style','text', ...
 'string','This is a figure', ...
 'position',[50 70 100 100], ...
 'fontsize',15);
// put a  text in figure 3
figure();
// create figure 1
uicontrol( 'style','text', ...
 'string','Another figure', ...
 'position',[50 70 100 100], ...
 'fontsize',15);
// put a text in figure 1
close();
// close the current graphic window (ie fig. 1)
close(h);
// close figure 3
```

SEE ALSO :
close, gcf

16.0.119 findobj ______ find an object with specified property

Author: Bertrand Guiheneuf

Allows to find a graphic tksci object testing the value of one of its properties.

USAGE :
h=findobj(prop,value)

INPUT PARAMETER :

- o prop : string character Name of the property to test.
- o value : string character specify the value the tested propoerty should be equal to.

OUTPUT PARAMETERS :
handle of the found object.

DESCRIPTION :
This routine is currentlyt used to find objects knowing their 'tag' property. It returns handle of the first found object which property 'prop' is equal to 'value'. If such an object does not exist, the function returns a void matrix.

EXAMPLE :

```
h=figure();
// creates  figure number 1.
uicontrol( h, 'style','text', ...
 'string','This is a figure', ...
 'position',[50 70 100 100], ...
 'fontsize',15, ...
 'tag','Alabel');
// put a  text in figure 1
lab=findobj('tag','Alabel');
// find the object which 'tag' value is 'Alabel'
disp('the handle of the label is '+string(lab));
close();
```

SEE ALSO :
uicontrol, uimenu, set, get

16.0.120 gcf ______ gets the current figure

Author: Bertrand Guiheneuf

Retrieve the current active tksci figure (toplevel window).

USAGE :
h=gcf()

INPUT PARAMETER :
None

OUTPUT PARAMETERS :
handle of the current figure.

DESCRIPTION :
The current figure is the last created (and still existent) figure.

EXAMPLE :

```
figure(5);
figure();
figure();
gcf()
// returns 2
close(gcf());
// close figure 2
gcf()
// returns 1
close(1);
```

```
gcf()
// returns 5
close(5);
```

SEE ALSO :
figure, close

16.0.121 get _______ get an UI object property value

Author: Bertrand Guiheneuf

Retrieve a property value from an User Interface object.

USAGE :
val=get(h,prop)

INPUT PARAMETERS :

- h : integer the handle of the object to retrieve a property
- prop : character string name of the property

OUTPUT PARAMETER :

- val : scilab object value of the property

DESCRIPTION :
This routine can be used to retrieve a specified property from a GUI object. Property name are character strings like 'style', 'position' .... This routine returns the value associated to the specified property. Obviously, the type of the returned object depends on the property one aims at querying. For example, the 'style' property which represents the kind of Object the UI control is (ie button, label, list, ...) will be represented as a string. On the contrary, the 'position' property, which represents the geometrical aspect of the UI control, will be coded as a [1,4] vector.

EXAMPLE :

```
h=uicontrol('string', 'Button');
// Opens a window with a  button.
p=get(h,'position');
// get the geometric qspect of the button
disp('Button width: ' + string(p(3)));
// print the width of the button
close();
// close figure
```

SEE ALSO :
uicontrol, uimenu, set

16.0.122 ScilabEval ___ tcl instruction : Evaluate a string with scilab interpreter

Author: Bertrand Guiheneuf

This routine is used inside tcl/tk script executed from scilab. It allows to evaluate any string from the scilab interpreter. It's mainly used to execute callbacks from tk widgets.

USAGE :
ScilabEval str

INPUT PARAMETER :

- str : tcl string character Contains the string to evaluate with the current scilab interpreter.

OUTPUT PARAMETERS :
None

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). The string str is put in the scilab interpreter buffer which then evaluates it. This has in general no border effect in the tcl/tk interpreter.

EXAMPLE (TCL/TK SCRIPT) :

```
# this text must be saved into a file, for instance
#  TMPDIR+'/test.tcl'
# then launch scilab and type TK_EvalFile(TMPDIR+'/test.tcl');
toplevel .w1
button .w1.b -text "Click here to see a new Scilab Graphic Window"\
  -command {ScilabEval "xselect()"}
pack .w1.b
```

SEE ALSO :
TK_EvalFile, TK_EvalStr, TK_GetVar, TK_Setvar

16.0.123 set _______ set an UI object property value

Author: Bertrand Guiheneuf

set a property value of a User Interface object.

USAGE :
get(h,prop,val)

INPUT PARAMETERS :

- h : integer the handle of the object which to set a property up
- prop : character string name of the property
- val : scilab object value to give to the property

OUTPUT PARAMETER :
None

DESCRIPTION :
This routine can be used to set a GUI object specified property. Property name are character strings like 'style', 'position' .... The type of the value field depends on the property one aims at setting. For example, the 'style' property which represents the kind of Object the UI control is (ie

button, label, list, ...) will be represented as a string. On the contrary, the 'position' property, which represents the geometrical aspect of the UI control, will be coded as a [1,4] vector.

EXAMPLE :

```
h=uicontrol('string', 'Button');
// Opens a window with a  button.
set(h,'position',[ 50 50 100 100]);
// set the geometric aspect of the button
close();
// close figure
```

SEE ALSO :
uicontrol, uimenu, get

16.0.124 TK_EvalFile ________ Reads and evaluate a tcl/tk file

Author: Bertrand Guiheneuf

With this routine, one can read and evaluate the content of a file containing tcl/tk scripts. This allows to create powerful tk interfaces.

USAGE :
TK_EvalFile(filename)

INPUT PARAMETER :

- filename : string character Contains the name of the file to read and evaluate.

OUTPUT PARAMETERS :
None

DESCRIPTION :
The filename might be relative or absolute. It is absolute if begining with a leading slash (/). When relative, the specified path refers to the directory where scilab was launched.

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 th 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 sould only be used when one aims at adressing 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 bother your code. It is generaly a good idea to take a look to the tcl/tk scripts evaluated when scilab is lauched. They are located in the directory $(SCI)/routines/tksci/SCRIPTS.

SEE ALSO :
ScilabEval, TK_EvalStr, TK_GetVar, TK_Setvar

16.0.125 TK_EvalStr Evaluate a string whithin the tcl/tk interpreter

Author: Bertrand Guiheneuf

This routine allows to evaluate tcl/tk instructions with the tcl/tk interpreter launched with scilab.

USAGE :
TK_EvalStr(str)

INPUT PARAMETER :

- str : string character Contains the string to evaluate within tcl/tk

OUTPUT PARAMETERS :
None

DESCRIPTION :
When tcl/tk support is enabled in scilab, you can evaluate tcl/tk expression from scilab interpreter. In fact, scilab launches a slave tcl/tk interpreter. The scilab instruction TK_EvalStr() can be used to evaluate expression without having to write a tcl/tk in a separated file (this is done using TK_EvalFile).

EXAMPLE :

```
TK_EvalStr('toplevel .foo');
// creates a toplevel TK window.
TK_EvalStr('label .foo.l -text ""TK married Scilab !!!""');
// create a static label
TK_EvalStr('pack .foo.l');
// pack the label widget. It appears on the screen.
text='button .foo.b -text close -command {destroy .foo}';
TK_EvalStr(text);
TK_EvalStr('pack .foo.b');
```

SEE ALSO :
ScilabEval, TK_EvalFile, TK_GetVar, TK_Setvar

16.0.126 TK_GetVar ________ Get a tcl/tk variable value

Author: Bertrand Guiheneuf

This routine allows to get a tcl/tk variable value.

USAGE :
valueTK_SetVar(varname)

INPUT PARAMETER :

- o varname : string character Contains the name of the tcl/tk variable.

OUTPUT PARAMETERS :

- o value : string character Contains the value of the tcl/tk variable 'varname'.

DESCRIPTION :
When tcl/tk support is enabled in scilab, this routine can be used to retreive the value of a tcl/tk variable.

EXAMPLE :

```
TK_EvalStr('toplevel .foo');
// creates a toplevel TK window.
TK_EvalStr('entry .foo.e -textvariable tvar');
// create an editable entry
TK_EvalStr('set tvar foobar');
// set the entry value
TK_EvalStr('pack .foo.e');
// pack the entry widget. It appears on the screen.
text=TK_GetVar('tvar')
// retrieve the variable value
// change the entry text and repeat the last command ...
```

SEE ALSO :
ScilabEval, TK_EvalFile, TK_EvalStr, TK_SetVar

16.0.127 TK_SetVar ______ Set a tcl/tk variable value

Author: Bertrand Guiheneuf

This routine allows to set a variable within the tcl/tk interpreter.

USAGE :
TK_SetVar(varname, value)

INPUT PARAMETER :

- o varname : string character Contains the name of the tcl/tk variable to set.
- o value : string character Contains the value to set up in the tcl/tk variable

OUTPUT PARAMETERS :
None

DESCRIPTION :
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 without having to build a tcl/tk instruction (and use TK_EvalStr).

EXAMPLE :
TK_EvalStr('toplevel .foo'); // creates a toplevel TK window. TK_EvalStr('label .foo.l -textvariable tvar'); // create a static label TK_EvalStr('pack .foo.l'); // pack the label widget. It appears on the screen. TK_SetVar('tvar','This text has been set directly within scilab');

SEE ALSO :
ScilabEval, TK_EvalFile, TK_EvalStr, TK_GetVar

16.0.128 uicontrol ______ create a Graphic User Interface object

Author: Bertrand Guiheneuf

This routine is the one which enventualy creates the Graphich User Interface Controls in the figures. This is a scingle instruction, but in conjonction with set(), it allows to create button, lists, ...

USAGE :
h=uicontrol([prop1, val1] [, prop2, val2] ...) or h=uicontrol(f, [prop1, val1] [, prop2, val2] ...)

INPUT PARAMETER :

- o f : integer handle of the figure which will contain the control
- o prop{1, 2 ...} : character string name of a property to set
- o val{1, 2 ...} : scilab object value to give to the corresponding property

OUTPUT PARAMETERS :
handle of the created object

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 coresponding 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 coincerning the 'style' field. 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 get(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. Those are descibed under:

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 String evaluated by the scilab interpreter when an usicontrol is activated. (for example when you click on a button).

FONTANGLE :
string : {'normal'} | italic | oblique For a control containing some text, this property sets the slant of the font.

FONTSIZE :
real For a control containing some text, this property sets the size of the font in FontUnits.

FONTUNITS :
string : {points} | pixels | normalized For a control containing some text, this property sets the units with which the fontsize is specified.

FONTWEIGHT :
string : light | {normal} | demi | bold For a control containing some text, this property sets the weight of the used font

LISTBOXTOP :
integer 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 differnet meaning on each uicontrol:

- Check Boxes : Max is the value the 'value' property take when control is checked
- Silders : Maximinum value of the slider
- List boxes : if (Max-Min)>1 the list allows multiple selection, Otherwise not.

MAX :
scalar Specifies the lowest value the 'value' property can be set to. It has however differnet meaning on each uicontrol:

- Check Boxes : Min is the value the 'value' property take when control is unchecked
- Silders : Mininum value of the slider
- List boxes : if (Max-Min)>1 the list allows multiple selection, Otherwise not.

PARENT :
integer Handle of the control parent. Changing this property allows to move a control from a figure to another.

POSITION :
[1,4] real vector or string This property is used to set or get the geometrical configuration of a control. It is a real; 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. The unit is determined by the 'Unit' property. One can also set this property by giving a string where the fields are separated by a '|', ie "x|y|w|h".

SLIDERSTEP :
[1,2] real vector or string small big This property represents the step a slider is moved when the user click on the arrow (small step) or on the slide bar (big step).

STRING :
string Generally, this property represents the text appearing in a uicontrol. Its exact meaning sometimes depends on the uicontrol style:

- List Boxes, Popup Menu the value can be a vector of string or a string where the items are separated by a '|'.

STYLE :
string : {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 whith to states : on or off.
- checkbox a small uicontrol that have to state : on or off
- edit an editable string control
- 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 of related controls.
- listbox a control representing a list of item that can be scrolled. The item 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 conjontion with findobj().

UNITS :
string : {points} | pixels | normalized Set the units used to specify the 'position' property.

USERDATA :
scilab object this can be used to associate any scilab object to an uicontrol.

VALUE :
Value of the uicontrol. The eact meaning depends on the style of the uivontrol.

- Check boxes, Radio buttons value is set to Max (see above) when on and Min when off.
- List Boxes, Popu Menu value is a vector of indexes corresponding to the index of the selected entry in the list. 1 is the first item of the list.
- Sliders value indicated by the slider bar.

EXAMPLE :

```
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();
// close the figure
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 (highlight) the item 1 and 3 in the list
close();
//close the figure
```

SEE ALSO :
figure, set, get, uimenu

16.0.129 uimenu ____ Create a menu or a submenu in a figure

Author: Bertrand Guiheneuf

This routine allows to add a menu or a submenu to the menu bar of a figure

USAGE :
h=uimenu(parent,prop1, val1, prop2, valu2 ...)

INPUT PARAMETER :

- parent : integer Handle of menu's parent
- prop? : string character name of a propoerty to set up
- val?? : scilab object value to affect to the corresponding property

OUTPUT PARAMETERS :

- 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. The 'callback' property allows to set up the scilab instruction to call when the item is selected by the user. The 'label' property allows to set up the text appearing for the item.

EXAMPLE :

```
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

Chapter 17

Language and data translation tools

17.0.130 ascii _______________ string ascii conversions

CALLING SEQUENCE :

```
a=ascii(txt)
txt=ascii(a)
```

PARAMETERS :

`txt` : character string.
`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

SEE ALSO : `code2str` 13, `str2code` 99

17.0.131 excel2sci _______________ reads ascii Excel files

CALLING SEQUENCE :

```
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 `exel2sci(fname)` returns the corresponding Scilab matrix of strings. Use `exel2sci(fname,sep )` for an other choice of separator.

Note: You may eval all or part ot `M` using function `evstr`.

SEE ALSO : `read` 88, `evstr` 21

17.0.132 mfile2sci _______ Matlab M_file to scilab translation function

CALLING SEQUENCE :

```
mfile2sci(M_file_path [,result_path [,Imode [,Recmode]]])
```

PARAMETERS :

`M_file_path` : a character string which gives the path of Matlab M_file to translate
`result_path` : a character string which gives the directory where the result has to be written. Default value is current directory.
`Imode` : Boolean flag, If true mfile2sci ask user for variable type and sizes when he cannot infer them. Default value : %f
`Recmode` : Boolean flag, used by translatepaths function. Must be %f to translate a single mfile.

DESCRIPTION :
mfile2sci, is Matlab M-file to Scilab function traduction tools. It tries whenever possible to replace call to Matlab functions by the equivalent scilab primitives and functions.

To translate 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 translate the calls to this Matlab M_file in other Matlab M_files. this function may be improved "by hand".

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.

Some other functions like plot, has no straightforward translation 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 began by "//!"

REMARKS :
This function is a still under developpement and is delivered as beta test.

Some Matlab4 basic functions are not yet translated. It is quite simple to add it. See <SCIDIR>/macros/m2sci/README for more details.

KNOWN BUGS :

1- : eval function instructions passed as strings are not translated.
2- : most of plot function are not yet translated
3- : if, for, endded by the end of file produce an error, add the closing end's
4- : Loop variable of for clause is available afterwards if loops terminates normally in matlab; it is cleared in Scilab generated code.
5- : inequality comparison which implies complex numbers produce a run time error such as "undefined variable : %s_2_s". User can define these operation with Matlab meaning with the following function definition:

  deff('r=%s_1_s(a,b)','r=real(a)<real(b)')

  deff('r=%s_2_s(a,b)','r=real(a)>real(b)')

  deff('r=%s_3_s(a,b)','r=real(a)<=real(b)')

  deff('r=%s_4_s(a,b)','r=real(a)>=real(b)')
6- : When i is a vector, Matlab allows insertions like a(i)=v for any v. In scilab v must have the same shape as a(i). This produces run time errors "submatrix incorrectly defined". Rewrite them as a(i)=v.' .

EXAMPLE :

```
//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']);
// translate it dor 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` 689

AUTHOR : Serge Steer, INRIA

17.0.133 mtlb_load ____ load variables from file with matlab4 format.

CALLING SEQUENCE :

```
mtlb_load fname
mtlb_load xxx.yyy
mtlb_load fname -ascii
```

PARAMETERS :

`fname` : a file name
`xxx.yyy` : a file name with extension

DESCRIPTION :
mtlb_load load variables on file with matlab4 formats.

`mtlb_load fname` loads in scilab all variables stored in file binary `fname.mat` .
`mtlb_load fname -ascii` loads in scilab variable stored in ascii file `fname`, which must contain a rectangular array of numeric data, arranged in m lines with n values in each line. The result is an m-by-n matrix named `fname` .
`mtlb_load xxx.yyy` reads the ascii file `xxx.yyy`, which must contain a rectangular array of numeric data, arranged in m lines with n values in each line. The result is an m-by-n matrix named `xxx`.
`"stdio" value for fname` doesnt redirect load from standard input.

SEE ALSO : `mtlb_save` 686, `save` 91, `load` 57

17.0.134 mtlb_save ______ save variables on file with matlab4 format.

CALLING SEQUENCE :

```
mtlb_save fname
mtlb_save fname X
mtlb_save fname X Y Z
mtlb_save fname X Y Z  -ascii
mtlb_save fname X Y Z  -ascii -double
mtlb_save fname X Y Z  -ascii -double -tabs
```

PARAMETERS :

`fname` : a file name
`X Y Z` : variable names

DESCRIPTION :
mtlb_save save variables on file with matlab4 formats.

`mtlb_save fname` saves all the current scilab variables which have corresponding matlab type to the binary "MAT-file" named `fname.mat`. The data may be retrieved with `mtlb_load`.
`mtlb_save fname X` saves only variable `X`.
`mtlb_save fname X Y Z` saves variables `X`, `Y`, and `Z`.
`mtlb_save fname X Y Z -ascii` uses 8-digit ASCII form instead of binary.
`mtlb_save fname X Y Z -ascii -double` uses 16-digit ASCII form.
`mtlb_save fname X Y Z -ascii -double -tabs` delimits with tabs.
`"stdio" value for fname` doesnt redirect save to standard output.

SEE ALSO : `mtlb_load` 686, `save` 91, `load` 57

17.0.135 pol2tex ________ convert polynomial to TeX format

CALLING SEQUENCE :

```
[y]=pol2tex(x)
```

PARAMETERS :

`x` : polynomial
`y` : list

DESCRIPTION :
Latex source code for the polynomial `x`. (For use with `texprint`)

EXAMPLE :

```
s=poly(0,'s');
p=s^3+2*s-5;
pol2tex(p)
```

SEE ALSO : `texprint` 688

17.0.136 sci2for ________ scilab function to Fortran routine conversion

CALLING SEQUENCE :

```
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` 36

17.0.137 texprint ________ TeX output of Scilab object

CALLING SEQUENCE :

```
[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).

EXAMPLE :

```
s=poly(0,'s');
texprint([1/s,s^2])
```

SEE ALSO : pol2tex 687, pol2str 418

17.0.138 translatepaths __ translate a set of Matlab M_file directories to scilab

CALLING SEQUENCE :

```
translatepaths(dirs_path ,res_path)
```

PARAMETERS :

dir_path : a character string vector which gives the paths of Matlab M_file directories to translate
res_path : a character string which gives the path of the directory where the scilab functions are written to.

DESCRIPTION :

translatepaths, translate all Matlab M-file contained in a set of directories to Scilab functions. Each function is translated by mfile2sci.
Trace of translation informations are stored in a file named "log" in the res_path directory

SEE ALSO : mfile2sci 684

AUTHOR : Serge Steer, INRIA

Chapter 18

Interprocess communication toolbox

"Scilab description"

18.0.139 AdCommunications ___ advanced communication toolbox for parallel programming

DESCRIPTION :

This the beta version of the Advanced Communications Toolbox (ACT).

This toolbox is based on existing libraries, such as

PVM - Parallel Vitual Machine

PBLAS - Message Passing Library dedicated to Matrix

ScaLapack - Parallel linear algebra Library

ACT manage remote executions of softwares and allow efficient exchanges of messages between those softwares. It offers the possibility to exploit numerous machines on a network, as a virtual computer, by creating a distributed group of independent softwares.

SEE ALSO : `Example` 692

18.0.140 Example ___ just to test the environment

DESCRIPTION :

We are the knights who say ni!

18.0.141 pvm_addhosts ___ add hosts to the virtual machine.

CALLING SEQUENCE :

```
[infos] = pvm_addhosts(hosts)
```

PARAMETERS :

`hosts` : row of strings, naming the hosts to be added.
`infos` : row of integer, corresponding to the status for each host.

DESCRIPTION :

`pvm_addhosts` adds the computers named in hosts to the configuration of computers making up the virtual machine. The names should have the same syntax as lines of a pvmd hostfile (see man page for pvmd3): A hostname followed by options of the form xx=y.

The array infos can be checked to determine the status for each host. Values less than zero indicate an error, while positive values are TIDs of the new hosts.

The status of hosts can be requested by the application using pvm_config. If a host fails it will be automatically deleted from the configuration. Using pvm_addhosts a replacement host can be added by the application, however it is the responsibility of the application developer to make his application tolerant of host failure. Another use of this feature would be to add more hosts as they become available, for example on a weekend, or if the application dynamically determines it could use more computational power.

EXAMPLE :

```
info = pvm_addhosts(["isostar","loop"])
```

SEE ALSO : `pvm_delhosts` 696, `pvm_config` 693

18.0.142 pvm_bcast — broacasts a message to all members of a group

CALLING SEQUENCE :

```
[info] = pvm_bcast(group, buff, msgtag)
```

PARAMETERS :

`group` : string, string group name of an existing group.
`buff` : data to be send.
`msgtag` : integer, message tag supplied by the user.
`info` : integer,

DESCRIPTION :
`pvm_bcast` broadcasts a message to all the members of group. In PVM 3.2 and later the broadcast message is not sent back to the sender. Any PVM task can call pvm_bcast(), it need not be a member of the group. The content of the message can be distinguished by msgtag. If pvm_bcast is successful, info will be 0. If some error occurs then info will be < 0.

pvm_bcast is asynchronous. Computation on the sending processor resumes as soon as the message is safely on its way to the receiving processors. This is in contrast to synchronous communication, during which computation on the sending processor halts until a matching receive is executed by all the receiving processors.

pvm_bcast first determines the tids of the group members by checking a group data base. A multicast is performed to these tids. If the group is changed during a broadcast the change will not be reflected in the broadcast. Multicasting is not supported by most multiprocessor vendors. Typically their native calls only support broadcasting to all the user's processes on a multiprocessor. Because of this omission, pvm_bcast may not be an efficient communication method on some multiprocessors.

EXAMPLE :

```
info = pvm_bcast( "worker",[12+%i,4,5;3,4+%i,5],10)
```

SEE ALSO : `pvm_joingroup` 699

18.0.143 pvm_config — sends a message

CALLING SEQUENCE :

```
res =  pvm_config()
```

PARAMETERS :

`res, list of 7 elements such that:`

res(1): integer returning the number of hosts (pvmds) in the virtual machine.

res(2): integer returning the number of different data formats being used.

res(3): integer returning pvmd task ID for host.

res(4): character string returning name of host.

res(5): character string returning architecture name of host

res(6): integer returning relative speed of host. Default value is 1000.

res(7): integer status code returned by the routine.

DESCRIPTION :
`pvm_config` returns information about the present virtual machine. The information returned is similar to that available from the console command conf.

The `pvm_config` function returns information about the entire virtual machine in one call.

If pvm_config is successful, info will be 0. If some error occurs then info will be < 0.

EXAMPLE :

```
res = pvm_config()
```

SEE ALSO : `pvm_tasks` ??

18.0.144 pvmd3 — PVM daemon

SYNOPSIS :
pvmd3 [-options] [hostfile] &

DESCRIPTION :
pvmd3 is a daemon process which coordinates hosts in a virtual machine. One pvmd must run on each host in the group. They provide the communication and process control functions needed by the user's PVM processes. The daemon can be started manually with a host file argument that will automatically start the remote pvmds. The local and remote pvmds can also be started from the PVM console program pvm.

The name of the daemon executable is pvmd3. It is usually started by a shell script, $PVM_ROOT/lib/pvmd.

Local daemon may also be started by the scilab instruction pvm_start() and remote daemons may also be started by the scilab function pvm_addhosts

OPTIONS :
The following options may be specified on the command line when starting the master pvmd or PVM console:

`-` dmask Set pvmd debug mask. Used to debug the pvmd or libpvm (not intended to be used to debug application programs). Mask is a hexadecimal number which is the sum of the following bits: Bit Information

`1` Packet routing
`2` Message routing and entry points
`4` Task management
`8` Slave pvmd startup
`10` Host table updates
`20` Select loop (below packet layer)
`40` IP network
`80` Multiprocessor port debugging
`100` Resource manager interface
`200` Application (messages with no destination, etc.)
`-nname` Specify an alternate hostname for the master pvmd to use. Useful when gethostname() returns a name not assigned to any network interface.

The following options are used by the master pvmd when starting slaves and are only of interest to someone writing a hoster. Don't just go using them, now.

`-s` Start pvmd in slave mode. Hostfile cannot be used, five additional parameters must be supplied: master pvmd index, master IP, master MTU, slave pvmd index, and slave IP.
`-S` Same as -s, but slave pvmd doesn't wait for its stdin to be closed after printing its parameters. Used for manual startup.

`-f` Slave doesn't fork after configuration (useful if the slave is to be controlled or monitored by some process).

Lines beginning with a splat (#), optionally preceded by whitespace, are ignored.
A simple host file might look like:

```
# my first host file
thud
fred
wilma
barney
betty
```

This specifies the names of five hosts to be configured in the virtual machine.

The master pvmd for a group is started by hand on the localhost, and it starts slaves on each of the remaining hosts using the rsh or rexec command. The master host may appear on any line of the hostfile, but must have an entry.

The simple format above works fine if you have the same login name on all five machines and the name of the master host in your .rhosts files on the other four.

There are several host file options available:

`lo=NAME` Specifies an alternate login name (NAME) to use.
`so=pw` This is necessary when the remote host cannot trust the master. Causes the master pvmd to prompt for a password for the remote host in the tty of the pvmd (note you can't start the master using the console or background it when using this option) you will see: Password (honk.cs.utk.edu:manchek): you should type your password for the remote host. The startup will then continue as normal.
`dx=FILE` Specifies the path of the pvmd executable. FILE may be a simple filename, an absolute pathname, or a path relative to the user's home directory on the remote host. This is mainly useful to aid in debugging new versions of PVM, but may have other uses.
`ep=PATH` Specifies a path for the pvmd to search for executable program components when spawning a new process. The path may have multiple elements, separated by colons (:).
`wd=PATH Specifies a working directory in which all spawned tasks on` this host will execute.
`sp=VALUE` Specifies the relative computational speed of this host compared to other hosts in the configuration. VALUE is an integer in the range [1 - 1000000]
`bx=PATH` Specifies the debugger program path. Note: the environment variable PVM_DEBUGGER can also be set.
`so=ms` Rarely used. Causes the master pvmd to request user to manually perform the startup of a pvmd on a slave host when rsh and rexec network services are disabled but IP connectivity exists. See section "MANUAL STARTUP".

A dollar sign ($) in an option introduces a variable name, for example $PVM_ARCH. Names are expanded from environment variables by each pvmd.

Each of the flags above has a default value. These are:
`lo` The loginname on the master host.
`so` Nothing
`dx` $PVM_ROOT/lib/pvmd (or environment variable PVM_DPATH)
`ep` pvm3/bin/$PVM_ARCH:$PVM_ROOT/bin/$PVM_ARCH
`wd` $HOME
`sp` 1000
`bx` $PVM_ROOT/lib/debugger

You can change these by adding a line with a star (*) in the first field followed by the options, for example:
`* lo=afriend so=pw` This sets new default values for 'lo' and 'so' for the remainder of the host file, or until the next '*' line. Options set on the last '*' line also apply to hosts added dynamically using pvm_addhosts().

Host options can be set without starting the hosts automatically. Information on host file lines beginning with '&' is stored, but the hosts are not started until added using pvm_addhosts().
Example hostfile:

```
# hostfile for testing on various platforms fonebone
refuge
# installed in /usr/local/here

sigi.cs
     dx=/usr/local/pvm3/lib/pvmd # borrowed accts, "guest", don't trust fonebone

* lo=guest so=pw sn666.jrandom.com cubie.misc.edu # really painful one, must start it by
```
hand and share a homedir & igor.firewall.com lo=guest2 so=ms ep=bob/pvm3/bin/$PVM_ARCH

MANUAL STARTUP :

When adding a host with this option set you will see on the tty of the pvmd:

```
*** Manual startup ***
```

Login to "honk" and type:

```
$PVM_ROOT/lib/pvmd -S -d0 -nhonk 1 80a9ca95:0cb6 4096 2 80a95c43:0000 Type response:
```

after typing the given command on host honk, you should see a line like:

```
ddpro<2312> arch<ALPHA> ip<80a95c43:0a8e> mtu<4096>
```

type this line on the tty of the master pvmd. You should then see:

```
Thanks
```

and the two pvmds should be able to communicate.

Note you can't start the master using the console or background it when using this option.

STOPPING PVMD3 :

The preferred method of stopping all the pvmds is to give the halt command in the PVM console. This kills all pvm tasks, all the remote daemons, the local daemon, and finally the console itself. If the master pvmd is killed manually it should be sent a SIGTERM signal to allow it to kill the remote pvmds and clean up various files.

The pvmd can be killed in a manner that leaves the file /tmp/pvmd.uid behind on one or more hosts. Uid is the numeric user ID (from /etc/passwd) of the user. This will prevent PVM from restarting on that host. Deletion of this file will fix this problem:

```
rm `( grep $user /etc/passwd || ypmatch $user passwd ) | \\
awk -F: `{print "/tmp/pvmd."$3; exit}'`
```

18.0.145 pvm_delhosts ______ deletes hosts from the virtual machine.

CALLING SEQUENCE :

```
infos = pvm_delhosts(hosts)
```

PARAMETERS :

`hosts` : row of strings, containing the names of the machines to be deleted.
`infos` : row of integers, contains the status code returned by the routine for the individual hosts.

DESCRIPTION :
`pvm_delhosts` deletes the computers of hosts from the existing configuration of computers making up the virtual machine. All PVM processes and the pvmd running on these computers are killed as the computer is deleted. The array infos can be checked to determine the status of each host. Values less than zero indicate an error, while zero values indicate a success.

If a host fails, the PVM system will continue to function and will automatically delete this host from the virtual machine. It is the responsibility of the application developer to make his application tolerant of host failure.

EXAMPLE :

```
info = pvm_delhosts(["isostar","loop"])
```

SEE ALSO : `pvm_addhosts` 692

18.0.146 pvm_exit ___ tells the local pvmd that this process is leaving PVM.

CALLING SEQUENCE :

```
[info] = pvm_exit()
```

PARAMETERS :

`info` : integer

DESCRIPTION :
`pvm_exit` tells the local pvmd that this process is leaving PVM. This routine does not kill the process, which can continue to perform tasks just like any other serial process.

pvm_exit should be called by all PVM processes before they stop or exit for good. It must be called by processes that were not started with pvm_spawn.

EXAMPLE :

```
pvm_exit()
```

SEE ALSO : `pvm` 701

18.0.147 pvm_getinst ___ returns the instance number in a group of a PVM process.

CALLING SEQUENCE :

```
[inum] = pvm_getinst(group, tid)
```

PARAMETERS :

`group` : string, string group name of an existing group.
`tid` : integer, task identifier of a PVM process.
`inum` : integer, instance number returned by the routine.

DESCRIPTION :
`pvm_getinst` takes a group name group and a PVM task identifier tid and returns the unique instance number that corresponds to the input. It can be called by any task whether in the group or not. If pvm_getinst is successful, inum will be >= 0. If some error occurs then inum will be < 0.

EXAMPLE :

```
inum = pvm_getinst( "worker", pvm_mytid() )
```

SEE ALSO : `pvm_joingroup` 699, `pvm_gettid` ??

18.0.148 pvm_get_timer Gets the system's notion of the current time.

CALLING SEQUENCE :

```
[time] = pvm_get_timer()
```

PARAMETERS :

`time` : scalar

DESCRIPTION :
`pvm_get_timer` returns the time elapsed since the last call of pvm_get_timer or the last call of pvm_set_timer. The time is expressed in elapsed microseconds. The resolution of the system clock is hardware dependent; the time may be updated continuously or in clock ticks. timer will be > 0. If some error occurs then timer will be -1.

EXAMPLE :

```
B = rand(100,100);
A = rand(100,100);
pvm_set_timer();C=A*B;t=pvm_get_timer()
```

SEE ALSO : `pvm_set_timer` 703

18.0.149 pvm_gsize _ returns the number of members presently in the named group.

CALLING SEQUENCE :

```
[nb] = pvm_gsize(group)
```

PARAMETERS :

`group` : string, string group name of an existing group.
`nb` : integer, returning the number of members presently in the group.

DESCRIPTION :
`pvm_gsize` returns the size of the group named group. If there is an error nb will be negative.

Since groups can change dynamically in PVM 3.0, this routine can only guarantee to return the instantaneous size of a given group.

EXAMPLE :

```
nb_worker = pvm_gsize( "worker" )
```

SEE ALSO : `pvm_joingroup` 699

18.0.150 pvm_halt ________________ stops the PVM daemon

CALLING SEQUENCE :

```
[info] = pvm_halt()
```

PARAMETERS :

`info` : integer, status code returned by the routine. Values less than zero indicate an error.

DESCRIPTION :

`pvm_halt` kills all PVM tasks, all the remote daemons, and the local daemon. If the master pvmd is killed manually it should be sent a SIGTERM signal to allow it to kill the remote pvmds and clean up various files.

The pvmd can be killed in a manner that leaves the file /tmp/pvmd.uid behind on one or more hosts. Uid is the numeric user ID (from /etc/passwd) of the user. This will prevent PVM from restarting on that host. Deletion of this file will fix this problem:

rm '(grep $user /etc/passwd || ypmatch $user passwd) | awk -F: '{print "/tmp/pvmd."$3; exit}''

For example:

```
-->pvm_halt()
 ans  =

    0.

-->pvm_halt()
 ans  =

  - 14.
```

Error -14 means: pvm_halt(): Can't contact local daemon

SEE ALSO : `pvm_start` 705, `pvm_addhosts` 692, `pvm_config` 693

18.0.151 pvm_joingroup enrolls the calling process in a named group.

CALLING SEQUENCE :

```
[inum] = pvm_joingroup(group)
```

PARAMETERS :

`group` : string, string group name of an existing group.
`inum` : integer, instance number returned by the routine.

DESCRIPTION :

`pvm_joingroup` enrolls the calling task in the group named group and returns the instance number inum of this task in this group. If there is an error inum will be negative.

Instance numbers start at 0 and count up. When using groups a (group, inum) pair uniquely identifies a PVM process. This is consistent with the PVM 2.4 naming schemes. If a task leaves a group by calling pvm_lvgroup and later rejoins the same group, the task is not guaranteed to get the same instance number. PVM attempts to reuse old instance numbers, so when a task joins a group it will get the lowest available instance number. A task can be a member of multiple groups simultaneously.

EXAMPLE :

```
inum =  pvm_joingroup( "worker" )
```

SEE ALSO : `pvm_lvgroup` 700

18.0.152 pvm_kill Terminates a specified PVM process.

CALLING SEQUENCE :

```
[infos] = pvm_kill(tids)
```

PARAMETERS :

`tids` : row of integer, task identifier of the PVM process to be killed (not yourself).
`infos` : row of integer, status code returned by the routine. Values less than zero indicate an error.

DESCRIPTION :

`pvm_kill` sends a terminate (SIGTERM) signal to the PVM process identified by tids. In the case of multiprocessors the terminate signal is replaced with a host dependent method for killing a process. If pvm_kill is successful,

The array infos can be checked to determine the status for each process. Values less than zero indicate an error, while zero values indicate a success.

pvm_kill is not designed to kill the calling process. To kill yourself in C call pvm_exit() followed by quit().

EXAMPLE :

```
info = pvm_kill(262153)
```

SEE ALSO : `pvm_exit` 697

18.0.153 pvm_lvgroup Unenrolls the calling process from a named group.

CALLING SEQUENCE :

```
[info] = pvm_lvgroup(group)
```

PARAMETERS :

`group` : string, group name of an existing group.
`info` : integer, status code returned by the routine.

DESCRIPTION :

`pvm_lvgroup` unenrolls the calling process from the group named group. If there is an error info will be negative.

If a process leaves a group by calling either pvm_lvgroup or pvm_exit, and later rejoins the same group, the process may be assigned a new instance number. Old instance numbers are reassigned to processes calling pvm_joingroup.

EXAMPLE :

```
info = pvm_lvgroup( "worker" )
```

SEE ALSO : `pvm_joingroup` 699

18.0.154 pvm . communications with other applications using Parallel Virutal Machine

DESCRIPTION :
PVM is a software system that enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational resource.

The individual computers may be shared- or local-memory multiprocessors, vector supercomputers, specialized graphics engines, or scalar workstations, that may be interconnected by a variety of networks, such as ethernet, FDDI.

Daemon programs (pvmd3) provide communication and process control between computers (see PVM manpage and manual for more details).

Most important functions of the PVM communication library are included in Scilab.

WARNING :
PVM must be installed in your envriromnent before using it in Scilab. PVM scilab have been developped using the version 3.3.7 of the PVM library.

AUTHORS :
PVM have been developped by A. L. Beguelin, J. J. Dongarra, G. A. Geist, W. C. Jiang, R. J. Manchek, B. K. Moore, V. S. Sunderam (see http://www.netlib.org/pvm3)

SEE ALSO : `pvm_barrier` ??, `pvm_mytid` 701, `pvm_bcast` 693, `pvm_parent` ??, `pvm_config` 693, `pvm_delhosts` 696, `pvm_recv` 702, `pvm_exit` 697, `pvm_send` 702, `pvm_getinst` 697, `pvm_spawn` 704, `pvm_gettid` ??, `pvm_spawn_independent` 703, `pvm_gsize` 698, `pvm_tasks` ??, `pvm_joingroup` 699, `pvm_tidtohost` 706 pvm_kill, pvm_lvgroup, pvm_start, pvm_halt

18.0.155 pvm_mytid ________ returns the tid of the calling process.

CALLING SEQUENCE :

```
[tid] = pvm_mytid()
```

PARAMETERS :

`tid` : integer, the task identifier of the calling PVM process. Values less than zero indicate an error.

DESCRIPTION :
`pvm_mytid` enrolls this process into PVM on its first call. It also generates a unique tid if this process was not created by pvm_spawn. pvm_mytid returns the tid of the calling process and can be called multiple times in an application.

Any PVM system call (not just pvm_mytid) will enroll a task in PVM if the task is not enrolled before the call.

The tid is a 32 bit positive integer created by the local pvmd. The 32 bits are divided into fields that encode various information about this process such as its location in the virtual machine (i.e. local pvmd address), the CPU number in the case where the process is on a multiprocessor, and a process ID field. This information is used by PVM and is not expected to be used by applications. Applications should not attempt to predict or interpret the tid with the exception of calling tidtohost()

If PVM has not been started before an application calls pvm_mytid the returned tid will be < 0.

EXAMPLE :

```
tid = pvm_mytid()
```

SEE ALSO : `pvm_tidtohost` 706, `pvm_parent` ??

18.0.156 pvm_recv ________ receive a message.

CALLING SEQUENCE :

```
[buff, info] = pvm_recv(tid, msgtag)
```

PARAMETERS :

`tid` : integer, task identifier of sending process supplied by the user.
`msgtag` : `integer, message tag supplied by the user. msgtag should be` >= 0.
`buff` : scilab variable, where the received message will be stored.
`info` : integer, status code returned by the routine. Values less than zero indicate an error.

DESCRIPTION :
`pvm_recv` blocks the process until a message with label msgtag has arrived from tid. pvm_recv then places the message in buff.

A -1 in msgtag or tid matches anything. This allows the user the following options. If tid = -1 and msgtag is defined by the user, then pvm_recv will accept a message from any process which has a matching msgtag. If msgtag = -1 and tid is defined by the user, then pvm_recv will accept any message that is sent from process tid. If tid = -1 and msgtag = -1, then pvm_recv will accept any message from any process.

The PVM model guarantees the following about message order. If task 1 sends message A to task 2, then task 1 sends message B to task 2, message A will arrive at task 2 before message B. Moreover, if both messages arrive before task 2 does a receive, then a wildcard receive will always return message A.

info will be the status code returned by the routine. If some error occurs then info will be < 0.

pvm_recv is blocking which means the routine waits until a message matching the user specified tid and msgtag values arrives at the local pvmd. If the message has already arrived then pvm_recv returns immediately with the message.

Once pvm_recv returns, the data in the message can be unpacked into the user's memory using the unpack routines.

EXAMPLE :

```
[b,info] = pvm_recv(pvm_parent(),100)
g = pvm_recv(pvm_parent(),200)
```

SEE ALSO : `pvm_send` 702, `pvm_bcast` 693

18.0.157 pvm_send ________ immediately sends (or multicast) data.

CALLING SEQUENCE :

```
[info] = pvm_send(tids,buff,msgtag)
```

PARAMETERS :

`tids` : row of integers, contains the task IDs of the tasks to be sent to.
`buff` : scilab variable.
`msgtag` : `integer, message tag supplied by the user. msgtag should be` >= 0. It allows the user's program to distinguish between different kinds of messages .
`info` : integer, status code returned by the routine. Values less than zero indicate an error.

DESCRIPTION :
pvm_send sends (or multicasts) a message to the PVM process identified in the tids array. Note that the message is not sent to the caller even if listed in the array of tids. msgtag is used to label the content of the message. If pvm_send is successful, info will be 0. If some error occurs then info will be < 0.

The pvm_send routine is asynchronous. Computation on the sending processor resumes as soon as the message is safely on its way to the receiving processor. This is in contrast to synchronous communication, during which computation on the sending processor halts until the matching receive is executed by the receiving processor.

If a multicast is performed, pvm_send first determines which other pvmds contain the specified tasks. Then passes the message to these pvmds which in turn distribute the message to their local tasks without further network traffic.

The PVM model guarantees the following about message order. If task 1 sends message A to task 2, then task 1 sends message B to task 2, message A will arrive at task 2 before message B. Moreover, if both messages arrive before task 2 does a receive, then a wildcard receive will always return message A.

Terminating a PVM task immediately after sending a message or messages from it may result in those messages being lost. To be sure, always call pvm_exit() before stopping.

EXAMPLE :

```
A = rand(5,5)*(1+%i);
deff('[x]=f(y)','x = 1/y')
info = pvm_send([262150, 262152], A(1:2:5,:), 100)
pvm_send(262146,f,200)
```

SEE ALSO : pvm_recv 702, pvm_bcast 693

18.0.158 pvm_set_timer Sets the system's notion of the current time.

CALLING SEQUENCE :

```
[info] = pvm_set_timer()
```

PARAMETERS :

info : scalar

DESCRIPTION :
pvm_set_timer initialized the timer. info will be 0. If some error occurs then info will be -1.

EXAMPLE :

```
pvm_set_timer()
```

SEE ALSO : pvm_get_timer 698

18.0.159 pvm_spawn_independent Starts new PVM processes.

CALLING SEQUENCE :

```
[tids, numt] = pvm_spawn_independent(task, ntask, [where])
```

PARAMETERS :

task: string, which is the executable file name of the PVM process to be started. The executable must already reside on the host on which it is to be started. The name may be a file in the PVM search path or an absolute path. The default PVM search path is $HOME/pvm3/bin/$PVM_ARCH/ .

ntask: integer, specifying the number of copies of the executable file to start.

where: string (optional), can be a host name such as "tequila.ens-lyon.fr" or a PVM architecture class such as "SUN4".

numt: integer, the actual number of tasks started. Values less than zero indicate a system error. tids: row of integers, array of the tids of the PVM processes started by this pvm_spawn_independent call.

DESCRIPTION :
pvm_spawn_independent starts ntask copies of the executable named task. On systems that support environment, spawn passes selected variables from parent environment to children tasks. If set, the envar PVM_EXPORT is passed. If PVM_EXPORT contains other names (separated by ':') they will be passed too. This is useful for e.g.:

```
setenv DISPLAY myworkstation:0.0
setenv MYSTERYVAR 13
setenv PVM_EXPORT DISPLAY:MYSTERYVAR
```

The hosts on which the PVM processes are started are determined by the where arguments. On return the array tids contains the PVM task identifiers for each process started.

If pvm_spawn_independent starts one or more tasks, numt will be the actual number of tasks started. If a system error occurs then numt will be < 0. If numt is less than ntask then some executables have failed to start and the user should check the last ntask - numt locations in the tids array which will contain error codes (see below for meaning). The first numt tids in the array are always valid.

When the argument where is omitted a heuristic (round-robin assignment) is used to distribute the ntask processes across the virtual machine.

In the special case where a multiprocessor is specified by where, pvm_spawn_independent will start all ntask copies on this single machine using the vendor's underlying routines.

EXAMPLE :

```
[tids, numt] = pvm_spawn_independent("a.out",2)
```

SEE ALSO : pvm 701, pvm_spawn 704

18.0.160 pvm_spawn Starts new Scilab processes.

CALLING SEQUENCE :

```
[tids, numt] = pvm_spawn(task, ntask, [where])
```

PARAMETERS :

task : string, which is the file name of the scilab script (see exec) to be started. The Scilab script must already reside on the host on which it is to be started. The name must an absolute path.

ntask : integer, specifying the number of copies of the scilab script to start.

where : string (optional), can be a host name such as "tequila.ens-lyon.fr" or a PVM architecture class such as "SUN4".

numt : integer, the actual number of tasks started. Values less than zero indicate a system error.

tids : row of integers, array of the tids of the PVM processes started by this pvm_spawn call.

DESCRIPTION :
`pvm_spawn` starts ntask copies of the scilab script task. On systems that support environment, spawn passes selected variables from parent environment to children tasks. If set, the envar PVM_EXPORT is passed. If PVM_EXPORT contains other names (separated by ':') they will be passed too. This is useful for e.g.:

```
setenv DISPLAY myworkstation:0.0
setenv MYSTERYVAR 13
setenv PVM_EXPORT DISPLAY:MYSTERYVAR
```

The hosts on which the PVM processes are started are determined by the where arguments. On return the array tids contains the PVM task identifiers for each process started.

If pvm_spawn starts one or more tasks, numt will be the actual number of tasks started. If a system error occurs then numt will be < 0. If numt is less than ntask then some executables have failed to start and the user should check the last ntask - numt locations in the tids array which will contain error codes (see below for meaning). The first numt tids in the array are always valid.

When the argument where is omitted a heuristic (round-robin assignment) is used to distribute the ntask processes across the virtual machine.

In the special case where a multiprocessor is specified by where, pvm_spawn will start all ntask copies on this single machine using the vendor's underlying routines.

EXAMPLE :

```
// create an exec file (script)
write(TMPDIR+'/foo.sce',['a=1';'plot2d()'])
// start a new Scilab on the same host to execute the script
[tids, numt] = pvm_spawn(TMPDIR+'/foo.sce',1)
pvm_kill(tids) //terminate the new scilab
```

SEE ALSO : `pvm` 701, `pvm_spawn_independent` 703

18.0.161 pvm_start ______ Start the PVM daemon

CALLING SEQUENCE :

```
[info] = pvm_start()
```

PARAMETERS :

`info` : integer, status code returned by the routine. Values less than zero indicate an error.

DESCRIPTION :
`pvm_start` starts the Pvmd3 which is a daemon process which coordinates unix hosts in a virtual machine. One pvmd3 must run on each host in the group. They provide the communication and process control functions needed by the user's PVM processes. The local and remote pvmds can also be started from the PVM console program pvm.

For example:

```
pvm_start()
 ans  =

    0.
-->pvm_start()
 ans  =

  - 28.
```

Error -28 means: pvm_start_pvmd(): Duplicate host

SEE ALSO : `pvm_halt` 698, `pvm_addhosts` 692, `pvm_config` 693

18.0.162 pvm_tidtohost ______ returns the host of the specified PVM process.

CALLING SEQUENCE :

```
[dtid] = pvm_tidtohost(tid)
```

PARAMETERS :

`tid` : integer, task identifier of the PVM process in question.
`dtid` : integer, the tid of the host's pvmd3 or a negative value if an error.

DESCRIPTION :

`pvm_tidtohost` returns the host id on which the process identified by tid is located.

EXAMPLE :

```
dtid = pvm_tidtohost(pvm_mytid())
```

SEE ALSO : `pvm_config` 693, `pvm_tasks` ??

18.0.163 Communications ______ communications with other applications using GeCi

DESCRIPTION :
This the beta version of the Communications Toolbox.

GeCi manages communications between Scilab and other applications (included Scilab itself) by using GeCi.

GeCI is an interactive communication manager created in order to manage remote executions of softwares and allow exchanges of messages between those softwares. It offers the possibility to exploit numerous machines on a network, as a virtual computer, by creating a distributed group of independent softwares.

In order to communicate, the other applications must have been prepared for, by including a communication library in them. The way to do this is described in the Communication Toolbox documentation.

SEE ALSO : `CreateLink` 706, `DestroyLink` 707, `ExecAppli` 707, `GetMsg` 708, `SendMsg` 709, `WaitMsg` 709

18.0.164 CreateLink ______ creates a link between two applications

CALLING SEQUENCE :

```
CreateLink(appli1,appli2)
```

PARAMETERS :

`appli1, name of an application` : string
`appli2` : string, name of an application

DESCRIPTION :
CreateLink creates a link from appli1 to appli2. Note that this link is directed. This link being created, appli1 can send messages to appli2 and appli2 can receive messages from appli1.

If the name of appli1 or appli2 is "SELF", it corresponds to the name of the application where we execute CreateLink.

If the name of appli1 or appli2 is "XGeCI", it corresponds to the name of the first Scilab application started.

SEE ALSO : DestroyLink 707, GetMsg 708, SendMsg 709

18.0.165 DestroyLink destroys the link between two applications

CALLING SEQUENCE :

```
DestroyLink(appli1,appli2)
```

PARAMETERS :

appli1, name of an application : string
appli2 : string, name of an application

DESCRIPTION :
DestroyLink destroys the link from appli1 to appli2.

If the name of appli1 or appli2 is "SELF", it corresponds to the name of the application where we execute DestroyLink.

If the name of appli1 or appli2 is "XGeCI", it corresponds to the name of the first Scilab application started.

SEE ALSO : CreateLink 706

18.0.166 ExecAppli executes an application

CALLING SEQUENCE :

```
ExecAppli(command,h,appli)
```

PARAMETERS :

command : string, command of the application
h : string, host name
appli : string, name of the application

DESCRIPTION :
ExecAppli executes the application described by command on the host h and gives it the name appli. Arguments of the application can be also given in the string command.

After executing ExecAppli, it is necessary to create links with CreateLink to be able to send messages between applications.

EXAMPLE :

```
h=unix_g("hostname")
ExecAppli(SCI+"/bin/scilex",h,"Scilab2")
CreateLink("SELF","Scilab2")
ExecAppli(SCI+"/bin/scilex -ns",h,"Scilab3")
```

SEE ALSO : CreateLink 706, ExecScilab 708, ExeclScilab 708

18.0.167 ExeclScilab executes another linked local Scilab

CALLING SEQUENCE :

```
ExeclScilab(appli)
```

PARAMETERS :

appli : string, name of new Scilab

DESCRIPTION :
ExecScilab executes a new Scilab application on the same host and creates links between them.

Use ExecAppli to execute a new Scilab application on a remote host.

SEE ALSO : CreateLink 706, ExecAppli 707, ExecScilab 708

18.0.168 ExecScilab executes another local Scilab

CALLING SEQUENCE :

```
ExecScilab(appli)
```

PARAMETERS :

appli : string, name of new Scilab

DESCRIPTION :
ExecScilab executes a new Scilab application on the same host.

After executing ExecScilab, it is necessary to create links with CreateLink to be able to send messages to new Scilab.

Use ExecAppli to execute a new Scilab application on a remote host.

SEE ALSO : CreateLink 706, ExecAppli 707, ExeclScilab 708

18.0.169 GetMsg gets a pending message

CALLING SEQUENCE :

```
[type,msg,appli] = GetMsg()
```

PARAMETERS :

type : string
msg : string
appli : string, name of an application

DESCRIPTION :
GetMsg gets, in an asynchronous way, a message sent by another application. The type of the message is string and the message itself is msg. The name of the application which has sent the message is appli,

SEE ALSO : SendMsg 709, WaitMsg 709

18.0.170 SendMsg ____________ sends a message

CALLING SEQUENCE :

```
SendMsg(type,msg)
```

PARAMETERS :

`type` : string
`msg` : string

DESCRIPTION :
`SendMsg` sends a message to ALL applications linked to this application.

SEE ALSO : `CreateLink` 706, `SendMsg` 709, `WaitMsg` 709

18.0.171 WaitMsg ____________ waits for a message

CALLING SEQUENCE :

```
[type,msg] = WaitMsg(appli)
```

PARAMETERS :

`appli` : string, name of an application
`type` : string
`msg` : string

DESCRIPTION :
`WaitMsg` waits for a message sent by another application. As long as any message has not been sent, `WaitMsg` waits. This is a way to send and get messages in a synchronous way. The type of the message is `string` and the message itself is `msg`.

By default the name of the first application (the one executed by GeCi) is "XGeCI".

SEE ALSO : `GetMsg` 708, `SendMsg` 709

Index

Symbols
%asn, 362
%helps, 42
%k, 381
%sn, 394

A
abcd, 254
abinv, 254
abort, 6
abs, 180
ABSBLK_f, 523
acos, 181
acosh, 180
acoshm, 180
acosm, 181
AdCommunications, 692
add_edge, 466
add_node, 466
addcolor, 114
addf, 181
addinter, 6
addmenu, 182
adj2sp, 183
adj_lists, 467
aff2ab, 426
AFFICH_f, 523
alphagifs, 584
amell, 184
analpf, 362
analyze, 556
and, 6
ANDLOG_f, 524
ANIMXY_f, 524
ans, 7
apropos, 7
arc_graph, 468
arc_number, 468
argn, 7
arhnk, 257
arl2, 257
arma, 402
arma2p, 401
armac, 401
armax, 403
armax1, 403
arsimul, 404
artest, 572
articul, 469
ascii, 684
asin, 185
asinh, 184
asinhm, 184
asinm, 185
atan, 186
AtanH, 584
atanh, 186
atanhm, 186
atanm, 187
augment, 308

B
backslash, 7
balanc, 427
balreal, 258
bandwr, 469
bbch, 585
bdiag, 427
besseli, 187
besselj, 188
besselk, 188
bessely, 189
best_match, 470
bezout, 408
bifish, 572
BIGSOM_f, 524
bilin, 259
binom, 586
black, 114
bloc2exp, 189
bloc2ss, 191
bode, 115
bool2s, 8
boolean, 8
boucle, 573
brackets, 9
break, 9
bstap, 309
buttmag, 363
bvode, 328

C

c_link, 193
cainv, 259
calerf, 192
calfrq, 260
call, 9
canon, 261
casc, 363
case, 11
ccontrg, 309
cdfbet, 560
cdfbin, 560
cdfchi, 561
cdfchn, 561
cdff, 562
cdffnc, 563
cdfgam, 563
cdfnbn, 564
cdfnor, 565
cdfpoi, 565
cdft, 566
ceil, 12
cepstrum, 364
chain_struct, 471
chaintest, 573
champ, 116
champ1, 116
chart, 117
chdir, 12
cheb1mag, 364
cheb2mag, 365
check_graph, 472
chepol, 365
chfact, 428
chol, 428
chsolve, 429
circuit, 473
clean, 408
clear, 13
clearfun, 12
clearglobal, 13
CLINDUMMY_f, 525
CLKIN_f, 525
CLKINV_f, 525
CLKOUT_f, 525
CLKOUTV_f, 526
CLKSOM_f, 526
CLKSOMV_f, 526
CLKSPLIT_f, 526
CLOCK_f, 526
close, 670
CLR_f, 527
cls2dls, 262
CLSS_f, 527
cmb_lin, 193
cmndred, 409
code2str, 13
coeff, 14
coff, 429
coffg, 409
colcomp, 430
colcompr, 409
colinout, 310
colnew, 331
colon, 14
colormap, 118
colregul, 262
comma, 15
comments, 15
Communications, 706
comp, 15
companion, 430
con_nodes, 474
cond, 431
conj, 194
connex, 473
CONST_f, 527
cont_frm, 263
cont_mat, 263
contour, 119
contour2d, 118
contr, 264
contract_edge, 475
contrss, 264
contwt, 589
contwtmir, 591
contwtspec, 593
convex_hull, 475
convol, 366
convstr, 194
copfac, 310
corr, 366
cos, 195
COSBLK_f, 528
cosh, 194
coshm, 195
cosm, 195
cotg, 196
coth, 196
cothm, 196
CreateLink, 706
csim, 265
cspect, 368
ctr_gram, 265
cumprod, 197
cumsum, 197
curblock, 552
CURV_f, 528
cwt, 594
cwtspec, 595
cwttrack, 597

cwttrack_all, 596
cycle_basis, 476
czt, 369

D
dasrt, 331
dassl, 333
datafit, 335
date, 15
dbphi, 266
dcf, 310
ddp, 266
debug, 198
dec2hex, 198
deff, 16
degree, 16
DELAY_f, 528
DELAYV_f, 528
delbpt, 17
delete_arcs, 476
delete_nodes, 477
delip, 198
delmenu, 199
demos, 199
DEMUX_f, 529
denom, 410
derivat, 410
des2ss, 311
des2tf, 268
DestroyLink, 707
det, 431
determ, 411
detr, 411
dft, 370
dhnorm, 311
diag, 200
diary, 17
dilate, 598
dimR2d, 600
diophant, 411
disp, 17
dispbpt, 17
dlgamma, 200
DLR_f, 529
DLRADAPT_f, 529
DLSS_f, 529
dmt, 601
dot, 18
double, 18
driver, 120
dscr, 268
dsimul, 269
dt_ility, 269
dtsi, 312
dwtspec, 602

E
edge_number, 478
edit, 201
edit_curv, 120
ell1mag, 370
else, 19
elseif, 19
empty, 19
emptystr, 201
end, 19
eqfir, 371
eqiir, 371
equal, 20
equil, 271
equil1, 270
ereduc, 432
erf, 202
erfc, 202
erfcx, 202
errbar, 121
errcatch, 20
errclear, 20
error, 21
eval, 203
eval3d, 121
eval3dp, 122
evans, 122
EVENTSCOPE_f, 530
evstr, 21
EVTDLY_f, 530
EVTGEN_f, 530
Example, 692
excel2sci, 684
exec, 22
ExecAppli, 707
ExeclScilab, 708
ExecScilab, 708
execstr, 23
exists, 23
exit, 24
exp, 432
EXPBLK_f, 531
expm, 433
external, 24
extraction, 24
eye, 27

F
fac3d, 123
factors, 412
faurre, 372
fbmfwt, 603
fbmlevinson, 604
fchamp, 123
fcontour, 124

fcontour2d, 124
fec, 125
feedback, 271
feval, 27
ffilt, 372
fft, 373
fft1d, 605
fgrayplot, 125
figure, 670
file, 28
filter, 374
find, 29
find_freq, 374
find_path, 478
findm, 375
findobj, 671
findWTLM, 606
fit_dat, 336
fix, 29
floor, 30
flt, 607
flts, 272
for, 30
format, 31
fort, 31
fourplan, 312
fplot2d, 126
fplot3d, 127
fplot3d1, 126
fprintf, 33
fprintfMat, 34
frep2tf, 273
freq, 274
freson, 275
frfit, 375
frmag, 375
fscanf, 34
fscanfMat, 35
fsfirlin, 376
fsolve, 337
fspecg, 312
fstabst, 313
fstair, 433
full, 203
fullrf, 434
fullrfk, 434
funcprot, 35
function, 36
funptr, 36
fusee, 574
FWT, 609
FWT2D, 608

G
G_make, 205
g_margin, 276
GAIN_f, 531
GAINBLK_f, 531
gainplot, 127
gamitg, 313
gamma, 204
gammaln, 204
gauss, 610
gcare, 314
gcd, 413
gcf, 672
gen_net, 478
GENERAL_f, 531
GENERIC_f, 532
GeneWei, 611
genfac3d, 128
genlib, 37
GENSIN_f, 532
GENSQR_f, 532
geom3d, 128
get, 673
getblocklabel, 553
getcolor, 129
getcwd, 37
getd, 38
getdate, 37
getenv, 38
getf, 39
getfield, 39
GetMsg, 708
getpid, 40
getscicosvars, 553
getsymbol, 129
getvalue, 204
gfare, 314
gfrancis, 275
gifs2wave, 612
gifseg, 613
girth, 479
givens, 435
glever, 435
glist, 480
global, 40
gpeche, 574
gr_menu, 133
graduate, 129
grand, 566
graph-list, 482
graph_2_mat, 480
graph_center, 481
graph_complement, 481
graph_diameter, 482
graph_power, 485
graph_simp, 485
graph_sum, 486

graph_union, 487
Graphics, 130
graycolormap, 132
grayplot, 133
group, 377
gschur, 436
gsort, 206
gspec, 437
gstacksize, 41
gtild, 315

H
h2norm, 316
h_cl, 316
h_inf, 317
h_inf_st, 317
h_norm, 318
halt, 206
HALT_f, 533
hamilton, 487
hank, 377
hankelsv, 316
hat, 41
havewindow, 207
help, 41
hermit, 413
hess, 438
hex2dec, 207
hilb, 378
hist3d, 134
histplot, 134
holder2d, 614
horner, 414
host, 43
hotcolormap, 135
householder, 438
hrmt, 414
htrianr, 415
hypermat, 43
hypermatrices, 43

I
icontwt, 614
iconvert, 44
idmt, 616
ieee, 44
if, 45
IFTHEL_f, 533
iir, 379
iirgroup, 378
iirlp, 379
im_inv, 439
imag, 45
impl, 338
imrep2ss, 277
IN_f, 533
input, 207
insertion, 46
int, 49
int16, 48
int2d, 339
int32, 48
int3d, 340
int8, 48
intc, 342
intdec, 380
integ, 617
INTEGRAL_f, 533
integrate, 208
interp, 209
interpln, 208
intersci, 49
intersect, 209
intg, 342
intl, 343
intppty, 49
INTRP2BLK_f, 534
INTRPLBLK_f, 534
intsplin, 210
inttrap, 210
inttype, 50
inv, 439
inv_coeff, 51
INVBLK_f, 534
invr, 415
invsyslin, 277
is_connex, 488
isdef, 211
isempty, 618
iserror, 51
isglobal, 51
isinf, 211
isnan, 211
isoview, 135
isreal, 212
IWT, 619
IWT2D, 618

J
jmat, 380

K
kalm, 380
karmarkar, 343
kernel, 440
keyboard, 52
knapsack, 488
Koutrouvelis, 620
kpure, 278
krac2, 278

kron, 212
kroneck, 441

L
lambdak, 621
lattn, 381
lattp, 382
lcf, 318
lcm, 416
lcmdiag, 416
ldiv, 417
ldivf, 212
leastsq, 344
left, 52
length, 52
lepskiiap, 621
leqr, 318
less, 53
lev, 384
levin, 382
lex_sort, 213
lft, 319
lgfft, 385
lib, 53
lin, 279
lindquist, 385
line_graph, 489
linearlt, 623
lines, 54
linf, 320
linfn, 321
link, 54
linpro, 345
linsolve, 442
linspace, 213
list, 56
lmisolver, 347
lmitool, 348
load, 57
load_graph, 490
loadwave, 556
locate, 136
log, 214
log10, 214
LOGBLK_f, 534
logm, 215
logspace, 215
LOOKUP_f, 534
lotest, 575
lqe, 279
lqg, 281
lqg2stan, 280
lqg_ltr, 321
lqr, 281
lsslist, 57
lstcat, 57
ltitr, 282
lu, 444
ludel, 442
lufact, 443
luget, 443
lusolve, 445
lyap, 445

M
m_circle, 137
macglov, 322
macr2lst, 215
macro, 58
macrovar, 216
make_graph, 490
manedit, 216
mapsound, 556
markp2ss, 283
mat_2_graph, 491
Matplot, 136
matrices, 58
matrix, 59
max, 60
max_cap_path, 492
max_clique, 492
MAX_f, 535
max_flow, 493
maxi, 59
mbmlevinson, 624
McCulloch, 625
mcfg1d, 626
mclearerr, 60
MCLOCK_f, 535
mclose, 61
mdfl1d, 628
mdfl2d, 629
mdznq1d, 630
mdznq2d, 630
mean, 217
median, 217
meof, 61
mese, 386
mesh2d, 494
metanet, 496
metanet_sync, 496
mexhat, 631
MFCLCK_f, 535
mfft, 386
mfile2sci, 684
mfprintf, 62, 68
mfscanf, 63, 70
mget, 64
mgetstr, 65
milk_drop, 137

min, 66
MIN_f, 535
min_lcost_cflow, 497
min_lcost_flow1, 498
min_lcost_flow2, 499
min_qcost_flow, 500
min_weight_tree, 501
mine, 576
mini, 65
minreal, 284
minss, 284
minus, 66
mlist, 67
mode, 67
modulo, 218
monolr, 632
mopen, 68
morlet, 634
mprintf, 62, 68
mps2linpro, 218
mput, 69
mputstr, 70
mrfit, 386
mscanf, 63, 70
mseek, 71
msprintf, 62, 68
msscanf, 63, 70
mtell, 72
mtlb_load, 686
mtlb_mode, 72
mtlb_save, 686
mtlb_sparse, 218
mulf, 219
multim1d, 635
multim2d, 638
MUX_f, 536

N
names, 73
narsimul, 405
NEGTOPOS_f, 536
nehari, 322
neighbors, 502
netclose, 502
netwindow, 503
netwindows, 503
newfun, 73
nextpowQ, 641
nlev, 446
nnz, 219
node_number, 503
nodes_2_path, 504
nodes_degrees, 504
noisegen, 405
norm, 220
not, 74
null, 74
numer, 417
nyquist, 138

O
obs_gram, 287
obscont, 285
obscont1, 577
observer, 286
obsv_mat, 287
obsvss, 288
ode, 350
ode_discrete, 350
ode_root, 354
odedc, 348
odedi, 406
odeoptions, 353
oldload, 74
oldsave, 75
ones, 75
optim, 355
or, 76
orth, 446
oscillsing, 642
OUT_f, 536
overloading, 76

P
p_margin, 289
param3d, 139
param3d1, 138
parents, 78
parrot, 323
part, 79
path_2_nodes, 505
pause, 79
pbig, 447
pdiv, 417
pen2ea, 220
pencan, 447
penlaur, 448
percent, 80
perfect_match, 505
pertrans, 221
pfss, 288
phasemag, 289
phc, 387
pinv, 449
pipe_network, 506
plot, 147
plot2d, 142
plot2d1, 140
plot2d2, 140
plot2d3, 141

plot2d4, 141
plot3d, 144
plot3d1, 143
plot_graph, 507
plotframe, 146
plus, 80
plzr, 147
pol2des, 418
pol2str, 418
pol2tex, 687
polar, 449
polfact, 419
poly, 81
portr3d, 577
portrait, 578
POSTONEG_f, 536
POWBLK_f, 536
power, 81
ppol, 290
prbs_a, 406
predecessors, 508
predef, 82
prescrib, 642
print, 85
printf, 84
printf_conversion, 82
printing, 148
prod, 221
PROD_f, 537
proj, 450
projsl, 290
projspec, 450
pseudoAW, 644
psmall, 451
pspect, 388
pvm, 701
pvm_addhosts, 692
pvm_bcast, 693
pvm_config, 693
pvm_delhosts, 696
pvm_exit, 697
pvm_get_timer, 698
pvm_getinst, 697
pvm_gsize, 698
pvm_halt, 698
pvm_joingroup, 699
pvm_kill, 700
pvm_lvgroup, 700
pvm_mytid, 701
pvm_recv, 702
pvm_send, 702
pvm_set_timer, 703
pvm_spawn, 704
pvm_spawn_independent, 703
pvm_start, 705
pvm_tidtohost, 706
pvmd3, 694
pwd, 85

Q
qassign, 508
qr, 451
QUANT_f, 537
quapro, 356
quaskro, 452
quit, 85
quote, 85

R
rand, 86
RAND_f, 537
randpencil, 453
range, 453
rank, 454
rat, 87
rational, 86
rcond, 454
rdivf, 222
read, 88
read4b, 87
readb, 88
readc_, 222
READC_f, 537
readmps, 222
real, 89
recur, 578
regdim, 645
REGISTER_f, 538
reglin, 406
RELAY_f, 538
remez, 390
remezb, 389
repfreq, 291
replot, 149
residu, 419
resume, 89
return, 90
reynitq, 647
RFILE_f, 539
ric_desc, 323
ricc, 292
riccati, 323
rlist, 90
roots, 420
rotate, 149
round, 90
routh_t, 420
rowcomp, 455
rowcompr, 420
rowinout, 324

rowregul, 293
rowshuff, 455
rpem, 391
rref, 456
rtitr, 293

S
salesman, 509
SAMPLEHOLD_f, 539
SAT_f, 539
save, 91
save_graph, 510
savewave, 557
SAWTOOTH_f, 539
sbinom, 647
scaling, 150
scanf, 92
scanf_conversion, 91
schur, 456
sci2exp, 224
sci2for, 687
sci2map, 225
sciargs, 93
scicos, 520
scicos_block, 546
scicos_cpr, 549
scicos_graphics, 547
scicos_link, 548
scicos_main, 545
scicos_menus, 520
scicos_model, 547
scicosim, 551
scifunc_block, 545
scilab, 93
ScilabEval, 673
scilink, 94
SCOPE_f, 540
SCOPXY_f, 540
sd2sci, 150
secto3d, 150
select, 94
SELECT_f, 541
semi, 95
semicolumn, 95
semidef, 358
SendMsg, 709
sensi, 325
set, 674
setbpt, 95
setfield, 95
setmenu, 225
setscicosvars, 554
sfact, 421
Sfgrayplot, 151
sgifs, 650
Sgrayplot, 151
sgrid, 152
shortest_path, 510
show_arcs, 511
show_graph, 512
show_nodes, 512
sign, 96
Signal, 392
signm, 96
sim_stable, 651
simp, 421
simp_mode, 422
sin, 226
SINBLK_f, 541
sinc, 394
sincd, 393
sinh, 226
sinhm, 226
sinm, 227
size, 96
slash, 97
sm2des, 295
sm2ss, 296
smooth, 227
smultim1d, 652
smultim2d, 654
solve, 227
SOM_f, 541
sort, 228
sortup, 656
sp2adj, 229
spaninter, 458
spanplus, 458
spantwo, 459
sparse, 229
spchol, 460
spcompack, 230
spec, 460
specfact, 296
speye, 231
spget, 232
splin, 232
split_edge, 513
SPLIT_f, 542
spones, 233
sprand, 233
sprintf, 97
spzeros, 233
sqroot, 461
sqrt, 234
sqrtm, 234
square, 152
squarewave, 235
srfaur, 395
srkf, 395

ss2des, 297
ss2ss, 297
ss2tf, 299
sscanf, 98
sskf, 395
ssprint, 235
ssrand, 235
st_deviation, 236
st_ility, 300
stabil, 299
stable_cov, 657
stable_sm, 657
stable_test, 658
stacksize, 99
standard_define, 550
standard_draw, 550
standard_input, 550
standard_origin, 551
standard_output, 551
star, 99
startup, 99
STOP_f, 542
str2code, 99
strcat, 237
strfbm, 659
strindex, 237
string, 100
strings, 100
stripblanks, 238
strong_con_nodes, 514
strong_connex, 514
strsubst, 238
subf, 239
subgraph, 515
successors, 516
sum, 239
SUPER_f, 542
supernode, 516
sva, 461
svd, 462
svplot, 301
sylm, 423
sylv, 462
symbols, 101
symcori, 660
synth2, 660
sysconv, 240
sysdiag, 240
syslin, 241
syssize, 301
system, 396
systems, 579
systmat, 423

T

tan, 243
TANBLK_f, 543
tangent, 580
tanh, 242
tanhm, 242
tanm, 243
TCLSS_f, 543
tdinit, 581
testmatrix, 101
texprint, 688
TEXT_f, 543
tf2des, 326
tf2ss, 302
then, 101
tilda, 102
TIME_f, 543
time_id, 302
timer, 243
titlepage, 153
TK_EvalFile, 675
TK_EvalStr, 676
TK_GetVar, 676
TK_SetVar, 677
tlist, 102
toeplitz, 244
trace, 463
trans, 396
trans_closure, 517
translatepaths, 689
TRASH_f, 544
trfmod, 244
trianfml, 244
tril, 245
trisolve, 245
triu, 246
trzeros, 303
type, 103
typename, 103
typeof, 246

U

ui_observer, 304
uicontrol, 678
uimenu, 680
uint16, 48
uint32, 48
uint8, 48
ulink, 104
union, 247
unique, 247
unix, 104
unix_g, 104
unix_s, 105
unix_w, 105
unix_x, 106

unobs, 305
unsetmenu, 248
user, 106

V

varargin, 106
varargout, 107
varn, 107
viewmat, 661
viewWTLM, 662

W

WaitMsg, 709
warning, 108
wave2gifs, 663
WFILE_f, 544
wfir, 396
what, 108
where, 109
whereami, 108
whereis, 108
while, 109
who, 109
whos, 110
wiener, 397
wigner, 397
window, 398
winsid, 153
writb, 110
write, 111
write4b, 110
WRITEC_f, 544
WT2Dext, 664
WT2DStruct, 665
WT2DVisu, 666
WTMultires, 666
WTStruct, 667

X

x_choices, 248
x_choose, 249
x_dialog, 249
x_matrix, 250
x_mdialog, 250
x_message, 251
xarc, 154
xarcs, 154
xarrows, 155
xaxis, 155
xbasc, 156
xbasimp, 157
xbasr, 157
xchange, 157
xclea, 158
xclear, 158
xclick, 159
xclip, 159
xdel, 160
xend, 160
xfarc, 161
xfarcs, 161
xfpoly, 162
xfpolys, 162
xfrect, 163
xget, 164
xgetech, 163
xgetfile, 249
xgetmouse, 164
xgrid, 165
xinfo, 166
xinit, 166
xlfont, 166
xload, 167
xname, 167
xnumb, 168
xpause, 168
xpoly, 168
xpolys, 169
xrect, 169
xrects, 170
xrpoly, 170
xs2fig, 171
xsave, 171
xsegs, 172
xselect, 172
xset, 174
xsetech, 173
xsetm, 175
xstring, 177
xstringb, 176
xstringl, 176
xtape, 178
xtitle, 178

Y

yulewalk, 398

Z

ZCROSS_f, 545
zeropen, 305
zeros, 252
zgrid, 178
zpbutt, 399
zpch1, 399
zpch2, 399
zpell, 400