Chapter 4: Basic functions

Beside Scilab functions, you can find in GROCER a few functions, which are not strictly speaking econometric functions, but which can be useful either for the use of GROCER as a mere toolbox or, and this is the majority of the functions presented in this chapter, to program new econometric functions: these functions are indeed used extensively in GROCER functions.

In the first category, you find functions for importing data from or exporting data to EXCEL: this is the subject of the part 2 in this chapter. In the second category, you find functions transforming matrices or vectors, most of them are gauss compatibility functions (very useful if you want to transform a Gauss function into a Scilab one) and functions used in a VAR context: these functions need no further description than the basic one, which you will find, as for every chapter, at the end of the chapter (part 4). In the category of functions used in numerous GROCER functions, you will also find functions written to make the interface between the user and the core of the econometric functions, which work only with vectors and matrices: they are the subject of part 3 of this chapter.

There are also one basic econometrics tools: newey_west, which provides the Newest-West variance estimator, designed to estimate the variance of a series in the presence of autocorrelation and heteroskedasticty and some miscellaneous functions, among which dblist, which returns the content of a scilab database and joinstr which concatenates altogether strings and string vectors. Part 1 gives the list of all basic functions, classed along these lines.

1. List of basic functions.

- Interface with EXCEL: impexc2bd and expbd2exc;
- Gauss compatibility functions: cols, dmult, findcoef, invpd, matmul, matdiv, meanc, miss, packr, polymult, reshape_gauss, rev, rows, seqa, shiftr, sortc, stcd, trimr, vech_gauss and xpnd1; these functions either mimic gauss functions that do not exist in scilab or grocer or that exist but behave differently than in scilab (case of the gauss function vech whose equivalent in grocer is vech_gauss)
- Matlab compatibility functions: repmat and str2mat;
- Matrix manipulation functions for VARs: commutation, dnplus, duplication, elimination and vech, invvech;
- Other matrix manipulation functions: com_size, crlag, dtrend, extraptab, fusvect, is_scalar, lag, lagser, mlag, mlagb, ptrend, transdif and vec;
- Functions for the interface with the user: associate, explol, explots, explouniv, explovars, explon, explone and vec2col;
- Basic econometric tools: mak, newey_west, undrift, spencer;
- Miscellaneous: autocum, dblist, fuslist and joinstr.

2. Interface with EXCEL.

In the version available for download, Scilab does not contain functions to import data from EXCEL or export data from Scilab to EXCEL. The Scilab group has kindly provided me the basic tools to do that, and I have adapted them so that they can deal with timeseries and, as regards the importation, improved the speed of the basic tool.

2.1 The function impexc2bd

This interface remains limited with respect to the EXCEL formats it can deals: imported EXCEL file must be in the csv format and files exported to EXCEL are in a text format. This feature imposes some easy manipulations in EXCEL. If you want to import an EXCEL file, you must open your EXCEL file, then save it in csv format. Take for example the file bdhendreric.xls in the sublibrary macros/grocer/bdexamples of your scilab directory. Open it, in the menu « file », click on « save as », and select in the « file type » box « CSV (separator: semi-column) », then on save. When you close the file, answer no when EXCEL asks you if you want to save the file. Your file is ready to be imported. If you type the following instruction:

```
(°)-->impexcel('macros/bdexamples/bdhenderic.csv',sep,'mylibrary/bdhenderic.dat')
```
where sep is the csv separator is (typically ';' or ',')

For instance with the separator ';':
```
-->impexcel('macros/bdexamples/bdhenderic.csv',';','mylibrary/bdhenderic.dat') (°])
```

Then the file bdhenderic will be saved in the file bdhenderic.dat under the sublibrary "mylibrary" of your workspace library (provided that you have created it before; otherwise the function will return you the following error:
```
!--error   240
File bdhenderic.dat already exists or directory write access denied).
```

The file 'mylibrary/bdhenderic.dat' can then be loaded in the workspace by the instruction (see scilab manual for more details about the function load):
```
-->load('mylibrary/bdhenderic.dat')
```

When this is done, the 5 ts: cte, ly, lp, lm1 and rnet are loaded and can be used in the workspace. To check the content of the database, you can type:
```
-->dblist('mylibrary/bdhenderic.dat')
```

The result should be:
```
ans  =
!cte   !
!rnet  !
!ly    !
!lp    !
!lm1   !
```

The function impexc2bd allows one importing either numerical data or timeseries. In any case, your series can be stored in EXCEL in rows or columns, but the first cell of the row or the column must be the name of the series. In the second case, the first series must be named « dates » (or « DATES »: since Scilab makes a difference between lower and upper case letters, impexc2bd transforms « DATES » in « dates ») and it must contain the dates of the series, in GROCER format: see again bdhenderic.xls for an example. If you want, you can import from the same EXCEL file series with different date spans or even periodicities: you just have to enter before each block of series a row (if your series are stored in rows in EXCEL) or a column (if your series are stored in columns in EXCEL) beginning with « dates » and containing the new dates (see the file « macros/bdexamples/doubledata.xls ») in your Scilab library.

(*) The function impexc2bd allows also the importation of panel data. Several constraints in the .csv file must be respected: first, there must be a column or a row of dates whose name must be 'dates' and a

column or a row that identifies the individuals, whose name must be 'id'; second, and consequently, there must be only one dataset in the file. (**)

2.2 The function expbd2exc.

The exportation of data from Scilab to Excel is easier. It is performed by the function expbd2exc. This function calls the function sci2excel stored in the sub-library macros/grocer/basic of your Scilab library. This function, written by the Scilab group, is not presented here.

To export data to Excel, you have 2 possibilities: either you export the whole content of a Scilab database and it is then not necessary to load it; or you export a list of data called by their names between quotes and these data must be present in the environment. Data that can be exported are either vectors or ts.

In the first case, the syntax is the following: expbd2exc('mydb','myexcelfile.txt'). For instance, if you want to export the file bdhenderic.dat, then type the following command:

```
--> expbd2exc('macros/grocer/bdexamples/bdhenderic.dat','c:/mydb/mydata.txt')
```

In this example, the five ts cte, ly, lp, lm1 and rnet will be exported to the file 'c:/mydb/mydata.txt' which Excel can read. This file will have 6 rows: the first one corresponding to the dates (in Scilab format) of the series and the 5 following ones to the corresponding values. The first cell of each row will be the name of the series or, as regards the first row, the name 'dates'. If the data exported are not ts, then the row dates will obviously not be present in the Excel file.

If you want to export only a subset of a database, or series coming from more than one database, then the syntax will be the following: expbd2exc(list('name1';'name2',…,'namen'),'myexcelfile.txt'). For instance, to export all the data in bdhenderic.dat except the constant, type:

```
--> load('macros/grocer/bdexamples/bdhenderic.dat'); expbd2exc(list('rnet','ly', 'lp','lm1'), 'c:/mydb/mydata.txt')
```

You should note than the decimal separator is by default supposed to be the dot '.'. If you want to change it, the you must enter the value of the decimal separator as third argument. For instance:

```
--> load('macros/grocer/bdexamples/bdhenderic.dat'); expbd2exc(list('rnet','ly', 'lp','lm1'), 'c:/mydb/mydata.txt',',')
```

Lastly, note that the library in which you create your Excel file must already exist; that the file if it exists is replaced. If the library does not exist or if it exists but is open, then Scilab returns the following error:

```
!--error  9999
File u:/aarbre/eric/mydata.txt cannot be opened for writing
```

3. Functions for the interface with the user. (*)

Although it will never attain the user-friendliness of a click and mouse software, GROCER intends to ease the user's life. In particular, GROCER allows the user to use the same econometric functions to deal with a great variety of objects: timeseries, vectors, matrices… The interface with the matrix calculations is done with explovars, explots, explolist, explol and explouniv, which are used intensively throughout GROCER[1]. GROCER allows also, whenever there cannot be any ambiguity, the user entering vectors in rows or in columns, even though the program uses only the column format: the -very easy!- transformation of a row or column vector into a column vector is done with vec2col[2]. Lastly there is a function str2vec which allows the user to enter names of variables as 'name=[name1,name2,…, namen]' and to create the matrix of names: ['name1';'name2';…;namen'][3]. Programming your own econometric functions should involve these functions…

Let us first examine the functions explovars, explots and explolist in detail, then the functions explol and explouinv.

3.1. The functions explovars, explots and explolist

The functions explovars and explots are the basic blocks that can be combined, for instance in functions explol and explouniv, that are directly used in high level econometric function such as ols, var,….

A call to explovars takes the form:
[namexos,listts,vect,indts,indvec,indcte,nvar]=explovars(names,defname)
or:
[namexos,listts,vect,indts,indvec,indcte,nvar]=explovars(names)

Explovars explodes a list of variables of any kind (ts, vectors, matrices, lists and strings representing the name of such variables) into:

- a vector *namexos* of strings, representing the names of all variables (if all variables have not been entered between quotes, then a default name applies, either given by the user as the second input argument of the function or endogenous if the user has not entered any second argument);
- a matrix *vec* representing the values of the matrices or vectors in the list and a vector *indvec* of indexes collecting the indexes of the corresponding variables in the names;
- a list *listts* containing all the ts in the list *names* and a vector *indts* of indexes collecting the indexes of the corresponding variables in the names;
- a vector *indcte* containing the indexes of all constant variables in the list names (there should be at most one…);
- a scalar equal to the number of variables (equal to the sum of the sizes of *indvec* and *indts*).

Assume for instance that your variables are in a matrix x of size (15x3) generated by grand(15,1,'nor',0,1). Then the following calls of function explovars with values of x will always return:

```
nvar  =

   3.
 indcte  =
```

[1] There is also a function explox, that was used in the 1.0 version, but that is no more used in version 1.1; I have kept it in the package for the users that were using it…
[2] This could also be done in scilab by the instruction x(:), but in that case it would also transform any matrix into a column, which would not allow to detect errors stemming from the entry of a matrix instead of a vector.
[3] For the expert Scilab user: this allows mimicking Scilab capability of passing named arguments, but with extending it to the case where there is a variable number of arguments.

```
    [ ]
 indvec  =

!   1. !
!   2. !
!   3. !
 indts  =

    [ ]
 vec  =

! - 0.8360876  - 0.5535018    0.5134568 !
!   0.3049811  - 1.8070101  - 1.3918619 !
! - 1.2272839    0.7840963  - 0.4279470 !
! - 0.9114889    2.47202    - 0.4206175 !
!   0.2824555    0.2448657    1.1876406 !
! - 1.7601598  - 1.278992   - 0.6492768 !
!   0.8148300  - 0.3163533  - 0.5971669 !
!   0.2065607  - 0.7602869  - 0.6048548 !
! - 0.0791338  - 1.0884885    1.6910222 !
!   0.7153849  - 1.5671516    1.1510432 !
!   0.4467040  - 2.857505     1.3220197 !
! - 0.1176301  - 0.7777774    0.7921972 !
! - 1.6305629    0.0503151  - 0.4962734 !
! - 1.6635418    1.013792   - 0.7781436 !
!   0.3631770  - 0.9658108    0.8430475 !
 listts  =

    ()
```

But namexos will take values varying with the way x is entered:

```
--> [namexos,listts,vect,indts,indvec,indcte,nvar]=explovars('x','exogenous');namexos
```

will result in:

```
namexos  =
!x_1  !
!x_2  !
!x_3  !
```

```
--> [namexos,listts,vect,indts,indvec,indcte,nvar]=explovars(x,'exogenous');namexos
```

will result in:

```
namexos  =
!exogenous # 1  !
!exogenous # 2  !
!exogenous # 3  !
```

```
-->dog=x(:,1);cat=x(:,2);mouse=x(:,3);[namexos,listts,vect,indts,indvec,indcte,nvar]=explovars(
['dog';'cat';'mouse'], 'exogenous'); namexos
```

will result in:

```
namexos  =
!dog    !
!cat    !
!mouse  !
```

```
-->[namexos,listts,vect,indts,indvec,indcte,nvar]=explovars(list('dog','cat','mouse'),'exogenous'); namexos
```

will result in:

```
!dog    !
!cat    !
!mouse  !
```

```
-->[namexos,listts,vect,indts,indvec,indcte,nvar]=explovars(list(dog,cat,mouse),'exogenous'); namexos
```

will result in:

```
namexos  =
!exogenous # 1  !
!exogenous # 2  !
!exogenous # 3  !
```

```
-->[namexos,listts,vect,indts,indvec,indcte,nvar]=explovars(list('dog','cat',mouse),'exogenous'); namexos
```

will result in:

```
namexos  =
!dog    !
!cat    !
!exogenous # 3  !
```

```
-->[namexos,listts,vect,indts,indvec,indcte,nvar]=explovars(['x(:,1:2)';'mouse'],'exogenous'); namexos
```

will result in:

```
namexos =

!x(:,1:2)_1  !
!x(:,1:2)_2  !
!mouse       !
```

Now take the variables lm1, ly, lp and rnet in the database bdenderic.dat:

```
-->load('SCI\macros\grocer\db\bdhenderic.dat');[namexos,listts,vect,indts,indvec,indcte,nvar]=
explovars(['lm1' 'ly' 'lp']
```

Then the result is:

```
nvar  =

   3.
 indcte  =
```

```
   [ ]
 indvec  =

   [ ]
 indts  =

!   1. !
!   2. !
!   3. !
 vect  =
[ ]
```

And listts contains the 3 ts lm1, ly and lp.

If there is a variable named ‘const’ in the list of the input variables, then the vector indvec will be filled with the index of the cte. For instance:

```
-->load('SCI\macros\grocer\db\bdhenderic.dat');[namexos,listts,vect,indts,indvec,indcte,nvar]=
explovars(['lm1' 'ly' 'lp' 'cte'])
```

results in:

```
nvar  =

   4.
 indcte  =

   4.
 indvec  =

   [ ]
 indts  =

!   1. !
!   2. !
!   3. !
 vect  =

   [ ]
```

If the const is in the third place, then indcte and indts are modified: indvec becomes 3 and indts [1 ; 2; 4]. So:

```
-->load('SCI\macros\grocer\db\bdhenderic.dat');[namexos,listts,vect,indts,indvec,indcte,nvar]=
explovars(['lm1' 'ly' 'cte' 'lp'])
```

results in:

```
nvar  =

   4.
 indcte  =
```

```
   3.
 indvec  =

    [ ]
 indts  =

!   1. !
!   2. !
!   4. !
 vect  =

    [ ]
```

In function explouniv, explovars is for instance used twice: to explode the endogenous variable and to explode the list of exogenous variables.

A call to explots takes the form:
[x,boundsvarb]=explots(listts)

The function explots determines the best range of bounds *boundsvarb* for the list of ts *listts* and the corresponding matrix of values *x*: the best range is defined as the longest time period over which the ts in the list *listts* are all simultaneously defined.

In function explouniv, explots is used if the function has detected the presence of ts and that there were no active bounds.

The function explolist is an old and much less useful function but which is uses sometimes in GROCER (for instance in ols). This function takes as input a first list of arguments and a list of commands, expressed as strings, which represent a condition to check on the first arguments of the function. It returns 2 lists: the list of arguments of the first input list which verify and the list of arguments of the first input list which does not verify those conditions.

For instance in ols, you will find the following command:
[grocer_lx,grocer_lnp]=explolist(grocer_lx,list('grocer_lx(i)=="noprint"'))

This command returns the initial list lx less the element 'noprint' if such an element belongs to the list lx and otherwise the initial list lx: in ols lx contains after the command is executed only the list of exogenous variables, whether the user has entered 'noprint' or not. The second argument is then either the list with one element 'noprint' or an empty list.

(°)3.2. The functions explone, explouniv and explon

The functions epxlovars and explots are the basic bricks used to build three functions that transform the data entered in any econometric function in grocer: the function explone is used when the econometric function delas with only one type of data (a var model without exogenous variables for instance); explouniv is used when you have two different types of data (endogenous and exogenous for instance as in an ols estimation) and explon when you have more types of data (for instance lhs endogenous variables, rhs endogenous variables, exogenous variables and instruments in an iv estimation). Since grocer allows different types of data (matrices, vectors, ts, between quotes or not), these functions are needed to:

- transform these data into matrices and vectors that are necessary for any estimation;
- recover the names of the variables for the printings;
- if there are ts, recover the bounds of the estimation, knowing that they can have been provided by the user of left to be determined by grocer program.

The function explone takes therefore as entry a unique list of variables.

A simple call to explol takes the form:
[y, namey, prests, b]=explol(ly)

ly can, in principle, be: a (nxk) matrix of real values; a (px1) matrix of names, each name being the name of a (nx1) vector, a (nxq) matrix or a ts; a list mixing any of the preceding object. Provided that the dimensions of the underlying vectors of values are compatible, then explone will return the *y* matrix of values (in general for the purpose of an estimation), the corresponding names *namey* (one name per column of matrix y), a boolean *prests* indicating if there were ts in the list of variables and a matrix *b*, equal to the matrix of bounds if they were given before the call to explone, to the greatest time span over which the ts in the entry were defined or to [] if they was no ts in the regression.

For instance:

```
-->load('SCI/macros/grocer/db/bdhenderic.dat'); bounds('1978q1','1978q4') ; [y, namey, prests, b]=explone(['lm1' 'ly' 'lp'])
```

results in:

```
b  =

!1978q1  !
!        !
!1978q4  !
 prests  =

  T
 namey  =

!lm1  !
!     !
!ly   !
!     !
!lp   !
 y  =

!  10.147021   11.503844  - 0.6580086 !
!  10.171461   11.510934  - 0.6346614 !
!  10.206368   11.514813  - 0.611669  !
!  10.238731   11.514245  - 0.5808162 !
```

You can enter 3 options to explone: the second argument can be a vector of bounds if you want to impose the bounds inside the function without resorting to the function bounds ([] if you do not want to); the third one is the name used to called variables not entered as strings (‘endogenous’ by default); the fourth one

is a boolean that indicates whether the program will test the existence of na's values in the matrix *y* (the default is true, but it can be useful to overcome this test, for instance for drawing graphs).

For instance:

```
-->load('SCI/macros/grocer/db/bdhenderic.dat'); bounds() ; [y, namey, prests, b]=explol(list('lm1','ly',lp)
,['1979q1' ; '1979q4'],'prices')
```

results in:

```
b =

!1979q1  !
!        !
!1979q4  !
 prests =

  T
 namey =

!lm1     !
!        !
!ly      !
!        !
!prices  !
 y =

!  10.268052   11.510363  - 0.5570325 !
!  10.287929   11.569201  - 0.5256881 !
!  10.315017   11.552588  - 0.4745929 !
!  10.325176   11.55743   - 0.4306380 !
```

Function explouniv[4] takes as first entries 2 lists of variables[5] (typically a list of endogenous and of exogenous variables). There are then 3 optional entries as in explone: the first one is a matrix of bounds; the second one a (2x1) matrix of strings defining the default names for each of the list and the third one a boolean that indicates whether the program will test the existence of na's values in the 2 matrices of values extracted from the 2 lists.

The output is also very similar. The only difference is that after the matrix y and the corresponding vector of names, there is a matrix x and the corresponding vector of names.

Function explon works slightly differently. Because the number of different objects is undetermined, the input and output take a somehow different form:

- the first argument is a list, where each element in the list collects all of object of one type; for instance if you have endogenous, exogenous and instruments then the first element of the list will gather (typically in a list) all endogenous elements, the second element of the list will similarly gather all exogenous elements and the third element of the list all instruments;

[4] The name of the function dates from the previous version, where the function was less general and applied essentially to univariate regressions. I have kept the name to maintain ascending compatibility.

[5] A more general function should have accepted an undefined number of lists. The programming would have been much more complicated however, for a use which did not seem obvious to me….

- the second argument the generic names for variables which will not have been entered between quotes; these elements must be entered as a string vector (a list of strings would also work); with a list of entries that are respectively endogenous, exogenous or instruments such a vector could be ['endogenous';'exogenous';'instruments'] or ['endo';'exo';'inst'];
- the third and fourth arguments are as in explon or explouniv (optional) vector of bounds and a boolean indicating if the program must test the presence in na values and, if it is the case, provide an error message;
- the first output argument takes also the form of a list; each elements of the list corresponds to the matrix of values associated to each type of object; in the above example, the first element of this matrix will collect the values of the endogenous variables, the second one the values of the exogenous ones and the third one the values of the instruments;
- the first output argument takes also the form of a list; each elements of the list corresponds to the vector of names associated to each type of object; in the above example, the first element of this matrix will collect the names of the endogenous variables, the second one the names of the exogenous ones and the third one the names of the instruments;

You can find examples of the use of this function in functions iv and ms_reg:

- [grocer_mats,grocer_names,grocer_prests,grocer_b]=explon(list(grocer_namey0,grocer_endo, grocer_exo,grocer_ivar),['endogenous' 'rhs endogenous' 'exogenous' 'instruments'])
  in function iv;
- [grocer_mats,grocer_names,grocer_prests,grocer_b]=explon(list(grocer_endo,grocer_exo_co m,grocer_exo_idio),['endogenous' 'swithing exog.' 'non swithing exog.'])
  in function ms_reg;

Note that it thereafter necessary to recover each element of the lists grocer_mats and grocer_names. In ms_reg for instance, this is done as follows:

```
grocer_y=grocer_mats(1)
grocer_z=grocer_mats(2)
grocer_x=grocer_mats(3)

grocer_namey=grocer_names(1)
grocer_namez=grocer_names(2)
grocer_namex  =grocer_names(3)
```

(°°)

4. The basic functions and their specification.

autocum_____AR(1) cumulation

CALLING SEQUENCE

[seriein]=autocum(seriein,rho)

PARAMETERS

INPUT:
* seriein = a (n x 1) or (1 x n) constant vector or a timeseries
* rho = a constant

OUTPUT:
* seriein = the vector (n x 1) or ts y such as:
* y(t) = rho*y(t-1) + x(t)
* y(t0) = x(t0) for the first observation t0

DESCRIPTION
Provides the vector (n x 1) or ts y such as:
* y(t) = rho*y(t-1) + x(t)
* y(t0) = x(t0) for the first observation t0

Example:
y=autocum(x,0.9)

cols_____# of columns in a matrix

CALLING SEQUENCE

[c]=cols(x)

PARAMETERS

INPUT:
x = input matrix

OUTPUT:
c = # of columns in x

DESCRIPTION
Returns the # of columns in a matrix x (a Gauss compatibility function).

com_size_____replaces values of matrix x

CALLING SEQUENCE

[ret,x,aout,bout]=com_size(x,a,b)

PARAMETERS

INPUT:
* x = a matrix or vector
* a = a scalar or matrix
* b = a scalar or matrix

OUTPUT:
* ret = an indicator 0 if common_size, 1 if not
* x = input matrix
* a = matrix size(x) or input matrix a if already size(x)
* b = matrix size(x) or input matrix b if already size(x)

DESCRIPTION
Makes a,b scalars equal to constant matrices size(x) or leaves them unchanged if they are already of common size.

Examples:
1) [r,x,ao,bo]=com_size(ones(4,3),5,2)
2) [r,x,ao,bo]=com_size(ones(2,2),[3,2;1,0],[2,4;5,3])
3) [r,x,ao,bo]=com_size(ones(2,2),[3,2],[2,4;5,3])

Example 1 returns:
bo =

```
!  2.   2.   2. !
!  2.   2.   2. !
!  2.   2.   2. !
!  2.   2.   2. !
 ao =

!  5.   5.   5. !
!  5.   5.   5. !
!  5.   5.   5. !
!  5.   5.   5. !
 x =

!  1.   1.   1. !
!  1.   1.   1. !
!  1.   1.   1. !
!  1.   1.   1. !
 r =

   0.
```

Example 2 returns:

```
bo  =

!   2.    2.    2. !
!   2.    2.    2. !
!   2.    2.    2. !
!   2.    2.    2. !
 ao  =

!   5.    5.    5. !
!   5.    5.    5. !
!   5.    5.    5. !
!   5.    5.    5. !
 x  =

!   1.    1.    1. !
!   1.    1.    1. !
!   1.    1.    1. !
!   1.    1.    1. !
 r  =

    0.
```

Example 3 returns:

```
bo  =

!   2.    4. !
!   5.    3. !
 ao  =

!   3.    2. !
 x  =

!   1.    1. !
!   1.    1. !
 r  =

    1.
```

commutation_____commutation of a matrix

CALLING SEQUENCE

[K]=commutation(m,n)

PARAMETERS

INPUT:
* m = # of rows of matrix A.
* n = # of columns of matrix A.

OUTPUT:
K = commutation matrix

DESCRIPTION
Given any (m X n) matrix A, returns the (mn x mn) K matrix such that: vec(A')=K*vec(A).

Example:
Used for the impulse functions calculated from vars (see irf_asy).

crlag_____circular lag function

CALLING SEQUENCE

[y]=crlag(x,n)

PARAMETERS

INPUT:
* x = input vector (tx1) or (1xt)
* n = # of values to return (optional; default=t)

OUTPUT:
y = a (nx1) (if x is (1xt)) or a (1xn) (if x is (1xt))
vector with:
* y(1) = x(n)
* y(2) = x(1)
.
* y(n) = x(n-1)

DESCRIPTION
Circular lag function.

Example:
y=crlag([1:5]) gives y=[5 1 2 3 4]

dblist_____gives the content of a database

CALLING SEQUENCE

[l]=dblist(filein)

PARAMETERS

INPUT:
filein = name of the database

OUTPUT:
l = column vector containing the names of variables in the database

DESCRIPTION
Gives the content of a database (simple but not elegant and, I presume, not efficient)

Example:
dblist('SCI/macros/grocer/db/bdhenderic.dat')

Gives the content of Hendry and Ericsson database. The results is:

```
!rnet  !
!      !
!lm1   !
!      !
!lp    !
!      !
!ly    !
```

defaultcoef_____finds default coefficients in equations

CALLING SEQUENCE

coef=defaultcoef(defaultname,speccarb,speccara,arg1,...,argn)

PARAMETERS

INPUT:
* defaultname = a string (default prefix of the coefficients)
* speccarb = the characters that must be before a coefficient to be sure that defaultname is not part of the name of another object
* speccara = the characters that must be after a coefficient to be sure that defaultname is not part of the name of another object
* argi = a string representing the text of an equation

OUTPUT:
coef=a (nx1) vector of coefficients names

DESCRIPTION
Determines the list of default coefs names in a set of equations

Example:
1) defaultcoef('co',['=' ; '+' ; '(' ; '-' ; '*'],['+' ; '-' ; '*' ; '/' ; ')'],'y=co1+co2*x1+(co3-co2)*x2')
2) grocer_namecoef=defaultcoef('a',grocer_speccarb,grocer_speccara,varargin(:))

In example 1, the coefficients are prefixed by co; they are asumed to be preceded by '=', '+', '(', '-' or '*' and followed by '+', '-', '*', '/', or ')'; the equation is 'y=co1+co2*x1+(co3-co2)*x2'.

The result is:

```
!co1  !
!    !
!co2  !
!    !
!co3  !
```

Example 2 is taken from function sur.

detrend_____detrend a matrix

CALLING SEQUENCE

[resid]=detrend(y,p)

PARAMETERS

INPUT:
* y = input matrix (or vector) of time-series (nobs x nvar)
* p = trend degree
  . p = 0, subtracts mean
  . p = 1, constant plus trend model
  . p > 1, higher order polynomial model
  . p = -1, returns y

OUTPUT:
resid = residuals from the detrending regression

DESCRIPTION
Detrends a matrix y using regression of y against a polynomial time trend of order p.

Examples:
1) xdt=detrend(x,0)
2) xdt=detrend(x,1)
3) r0t = detrend(trimr(dx,grocer_k,0),f) (see johansen.sci)

Example 1 simply demeans x, example 2 detrends it by a linear trend, example 3 taken from the function johansen detrends variable trimr(dx,grocer_k,0) by a trend of order f.

dmult _____gauss dmult function

CALLING SEQUENCE

[M]=dmult(A,B)

PARAMETERS

INPUT:
* A = a matrix
* B = a matrix

OUTPUT:
m = diag(A) times B

DESCRIPTION
Computes the product of diag(A) and B (a Gauss compatibility function).

dnplus _____vec to vech function

CALLING SEQUENCE

B = dnplus(n)

PARAMETERS

INPUT:
n = a scalar, equal to the size of the symmetric matrix

OUTPUT:
B = a ((n*(n+1)/2) x (n*n)) matrix

DESCRIPTION
Realize the passing from a vecctorized matrix by vec to a vectorized matrix by vech. Equivalent to duplication()'.

duplication _____Magnus and Neudecker's duplication matrix

CALLING SEQUENCE

[d]=duplication(n)

PARAMETERS

INPUT:
* n = dimension of the underlying var-cov matrix

OUTPUT:
* d = (n*n x n*(n+1)/2) duplication matrix

DESCRIPTION
Returns Magnus and Neudecker's duplication matrix of size n.

Example:
dup=duplication(N) (see irf_asy)

elimination_____Magnus and Neudecker's elimination matrix

CALLING SEQUENCE

[d]=elimination(n)

PARAMETERS

INPUT:
n = the size of the underlying var_cov matrix

OUTPUT:
d = the elimination matrix

DESCRIPTION
Returns Magnus and Neudecker's elimination matrix of size n.

Example:
dup=duplication(N) (se irf_asy.sci)

expbd2exc_____exportation to Excel

CALLING SEQUENCE

[]=expbd2exc(bd,output)

PARAMETERS

INPUT:
* bd = the name of a database or of a list of ts loaded in the environment
* output = the excel file where to save the data

OUTPUT:
nothing

DESCRIPTION
Exports the content of a data base or a list of variables bd to a file that Excel can read.

Examples:
1) expbd2exc('macros/grocer/bdexamples/bdhenderic.dat','c:/mydb/mydata.txt')
2) load('SCI/macros/grocer/bdexamples/bdhenderic.dat'); expbd2exc(list('cte','rnet','ly', 'lp','lm1'), 'c:/mydb/mydata.txt')

Examples 1 and 2 do exactly the same thing: save on a file 'c:/mydb/mydata.txt' the variables 'cte','rnet','ly', 'lp' and 'lm1' contained in the scilab file 'SCI/macros/grocer/bdexamples/bdhenderic.dat'.

explolist_____splitting of a list

CALLING SEQUENCE

[l1,l3]=explolist(l1,l2)

PARAMETERS

INPUT:
* l1 = the list to split
* l2 =the list of conditions

OUTPUT:
* l1 = the initial list without the elements verifying the conditions
* l3 = the initial list with the elements verifying the conditions

DESCRIPTION
Splits the list l1 in 2 lists containing the elements that do (l3) or do not (l1) verify the conditions contained in list l2 (used in ols but not very interesting).

Example:
[grocer_lx,grocer_lnp]=explolist(grocer_lx,list('grocer_lx(i)=="noprint"'))
Taken from function ols.

explots_____explosion of a list of ts

CALLING SEQUENCE

[x,boundsvarb]=explots(listts)

PARAMETERS

INPUT:
listts = a list of ts

OUTPUT:
* x = a (T x k) real matrix
* boundsvarb = a (2 x 1) string matrix (of dates)

DESCRIPTION
From a list of ts find the greatest time span over which the series are simultaneously defined and non NA and store in a vector the corresponding values.

Example:
load('SCI/macros/grocer/db/bdhenderic.dat');[x,boundsvarb]=explots(list(lm1,ly,lp,rnet))

Determines the admissible bounds for a regression involving the variables m1,ly,lp,rnet of the database bdhenderic.dat: 1963q1 to 1989q2 and stores the values in the series on this period in matix x.

explouniv_____explosion of a sequence of variables

CALLING SEQUENCE

[y,namey,x,namexos,prests,boundsvarb]=explouniv(ly,lx,b,named,presna)

PARAMETERS

INPUT:
* ly (lx) = list of variables:
  each element could be
  - a timeseries, a real vector, a real matrix or a string (the name of a variable with one of the types cited above, between quotes)
  - a matrix of strings, each one being the name of a variable
  - the string 'cte' or 'const' if the user wants a constant to be included automatically
* b = a (px1) string vector (of dates) (optional: if not given the function either takes the existing bounds or determines the bounds suitable to the given series)
* named = a (2x1) vector of strings representing the name of variables not entered between quotes (optional; default = ['endogenous' ; 'exogenous'])
* testna = a booelan indicating whether the program will test the existence of na's values in the matrices of values y and x

---

OUTPUT:
* y = a (Tx1) real vector or a ts
* namey = a string
* x = a (T x k) real matrix
* namexos = a (1 x k) string vector
* prests = a boolean indicating whether there is a ts in x
* boundsvarb = a (2 x 1) string matrix (of dates)

DESCRIPTION
From a series -that can be supposed to be an endogenous variable- and a list of series -that can be supposed to be exogenous variables- retrieves their names or give them names if they don't have (when an element is not string), store the values of these series in a vector and a matrix, and, if necessary, update the admissible estimation bounds.

Example:
[y,namey,x,namexos,prests,boundsvarb]=explouniv('delts(lm1)',list('delts(ly)','delts(lp)','const'))

Provides:

boundsvarb =

!1963q2 !
! !
!1989q2 !
prests =

T
namexos =

!delts(ly) !
! !
!delts(lp) !
! !
!const !
x =

! 0.0390034 0.0047424 1. !
! 0.0030870 0.0080163 1. !
! 0.0209885 0.0138010 1. !
! 0.0157089 0.0007787 1. !
! 0.0137243 0.0135667 1. !
! - 0.0008068 0.0116560 1. !
! 0.0229969 0.0083836 1. !
! - 0.0128250 0.0142634 1. !
! 0.0068007 0.0083169 1. !
! 0.0098162 0.0112869 1. !
! 0.0112467 0.0060656 1. !
! 0.0022278 0.0101358 1. !
! 0.0016613 0.0131184 1. !
! 0.0061426 0.0104237 1. !
! - 0.0007133 0.0085061 1. !
! 0.0258513 - 0.0013986 1. !
! 0.0005722 0.0128876 1. !
! 0.0090065 0.0024707 1. !
! - 0.0002298 0.0074279 1. !
! 0.0392895 0.0175068 1. !
! - 0.0156918 0.0192279 1. !
! 0.0248690 0.0123366 1. !
! 0.0084429 0.0134938 1. !
! - 0.0106325 0.0116452 1. !
! 0.0074253 0.0079896 1. !
...
namey =

delts(lm1)
y =

! 0.0219412 !
! 0.0050507 !
! 0.0304794 !
...

explovars_____explosion of a sequence of variables

CALLING SEQUENCE

[namexos,listts,vecy,indts,ndvec,indcte,nvar]=explovars(names,defname)

PARAMETERS

INPUT:
* names = either
  - a list of variables; each element could be a timeseries, a real vector, a real matrix or a string (the name of a variable with one of the types cited above, between quotes)
  - a matrix of strings, each one being the name of a variable
  - the string 'cte' or 'const'
* defname = a string indicating the prefix used by default to name the variables (default = 'endogenous')

OUTPUT:
* namexos = a (k x 1) string vector
* listts = a list of ts
* vec = a (n x k) matrix of real values
* indts = a (k1 x 1) vector of integers that indicates the indexes of the ts in the original list
* indvec = a (k2 x 1) vector of integers that indicates the indexes of the real vectors in the original list
* indcte = a (k3 x 1) vector of integers that indicates the indexes of the terms 'cte' or 'const' in the original list
* nvar = # of variables in the list

DESCRIPTION
From a list of series retrieve their names or give them names if they don't have (when an element is not string), store the ts in a list, store the values of the vectors and matrices in a unique matrix, store the index of the ts, the index of the vectors and the index of the 'cte' or 'const' string.

Example:
[grocer_namey,grocer_listtsy,grocer_y,grocer_indtsy,grocer_indvecy,grocer_indctey]=explovars(grocer_namey,'endogenous')

Example taken from function explouniv. Grocer_namey is here the name of an endogenous variable. If it has not been entered between quotes, then the variable is simply called 'endogenous'. If it is a ts, then grocer_listtsy is the list containing this ts, grocer_y is empty, grocer_indtsy is equal to 1, grocer_indvecy is empty, grocer_indctey is empty. If it is a vector, then grocer_listtsy is empty, grocer_y is the corresponding vector, grocer_indtsy is empty, grocer_indvecy is equal to 1, grocer_indctey is empty and grocer_indctey is equal to 1. If it has been entered as 'cte', then grocer_listtsy, grocer_y, grocer_indtsy and grocer_indvecy are all empty and grocer_indctey is equal to 1.

explox_____explosion of a sequence of variables

CALLING SEQUENCE

[x,namexos,boundsvar,prests]=explox(l,defname,boundsvar)

PARAMETERS

INPUT:
* l = either
  - a list of variables, each element could be a timeseries, a real vector, a real matrix or a string (the name of a variable with one of the types cited above, between quotes)
  - a matrix of strings, each one being the name of a variable
  - the string 'cte' if the user wants a constant to be included automatically
* defname = default name of variables
* boundsvar = a (2x1) string matrix (of dates) equal to the maximum period over which to take the series (optional)

OUTPUT:
* x = a (T x k) real matrix
* namexos = a (1 x k) string vector
* b = a (2 x 1) string matrix (of dates)
* prests = a boolean indicating whether there is a ts in x

DESCRIPTION
From a list of series retrieve their names or give them names if they don't have (when an element is not string), store the values of these series in a matrix, and, if necessary, update the admissible estimation bounds.

Examples:
1)[grocer_x,grocer_namexos,grocer_boundsvarb]=explox(grocer_lx,'exogenous',grocer_boundsvarb) (taken from ols)
2)[grocer_y,grocer_namey,grocer_boundsvar,grocer_prests]=explox(grocer_endo,'endogenous') (taken from var)

explol_____explosion of a sequence of variables

CALLING SEQUENCE

[y,namey,prests,b]=explol(ly,b,named,presna)

PARAMETERS

INPUT:
* ly = list of variables:
  each element could be
  - a timeseries, a real vector, a real matrix or a string (the name of a variable with one of the types cited above, between quotes)
  - a matrix of strings, each one being the name of a variable
  - the string 'cte' or 'const' if the user wants a constant to be included automatically
* b = a (px1) string vector (of dates) (optional: if not given the function either takes the existing bounds or determines the bounds suitable to the given series)
* named = a (2x1) vector of strings representing the name of variable not entered between quotes (optional; default = ['endogenous' ; 'exogenous'])

* testna = a booelan indicating whether the program will test the existence of na's values in the matrices of values y and x

---

OUTPUT:
* y = a (Tx1) real vector or a ts
* namey = a string
* prests = a boolean indicating if y is or is not a ts
* b = a (2x1) string matrix (of dates)

DESCRIPTION
From a list of variables, store the values of the corresponding series in a matrix, their names in a vector and, if necessary, define the admissible estimation bounds.

Example:
[grocer_y,grocer_namey,grocer_prests,grocer_boundsvarb]=explol(grocer_namey) (taken from ols)

extraptab_____extrapolation of statistical tables

CALLING SEQUENCE

[critical]=extraptab(tab,n)

PARAMETERS

INPUT:
* tab = a (pxq) table where
  - the first row is not used (it is supposed to be filled with the size levels
  - each row has the form [n v1 ... vn] where n is a # of observations (filled from the lowest to the highest in col 1, starting from row 2, since row 1 is not used)
* n = a scalar (the # of observations for which critical are calculated by extrapolation)

OUTPUT:
critical = a (1xq) vector of critical values for all sizes present in the table tab

DESCRIPTION
From a table (implicitly of stats...) filled for some values for the # of observations, calculates the corresponding values for n observations.

Example:
critical1=extraptab(tab1,T) (see schmiphi_tab)

findcoeff_____find coefficients in an equation

CALLING SEQUENCE

[coef]=findcoef(str,car)

PARAMETERS

INPUT:
* str = the equation analyzed
* car = the default car for the beginning of the coefficients
---
OUTPUT:
* coef = the names of the coefficients

DESCRIPTION
In an equation where the coefficients are given by default (car1,...,carn) find the last coefficient in this equation and infer the names of the coefficients.

Example:
findcoef('a1+a2*x1+(a3-a2)*x2+(a4-a3)*a5b','a')

This example gives ['a1';'a2';'a3';'a4']. The main usage is in econometric functions where the user must give the text of an equation (nls, twosls, threesls,…).

fuslist_____merge vectors of a list

CALLING SEQUENCE

[vecout]=fuslist(listin)

PARAMETERS

INPUT:
listin = the list of vectors to fusion

OUTPUT:
* vecout= the fusion

DESCRIPTION
Makes the fusion between the vectors of a list. Each component of the list must be a (n,1) vector.

Example:
numx3=fuslist(listx1_2)

This example, taken from automatic, realises the fusion of the vectors of indexes resulting from the search of econometric specifications.

fusvect_____sorted fusion of vectors

CALLING SEQUENCE

[ind]=fusvect(arg1,...,argn)

PARAMETERS

INPUT:
arg1,...,argn = (n x1) vectors

OUTPUT:
ind = a (n x1) vector

DESCRIPTION
Makes the fusion of (n x1) vectors and sort them in increasing order

Example:
fusvect([1 ;4 ; 5],[3 ; 2 ;4])

Returns:
```
!   5. !
!   4. !
!   4. !
!   3. !
!   2. !
!   1. !
```

impexc2bd_____importation of a csv excel file

CALLING SEQUENCE

impexc2bd(filein,sep,fileout)

PARAMETERS

INPUT:
* filein = name of the file to be imported (between quotes)
* sep = separator used in the filein (between quotes)
* fileout = name of the scilab file where to save the imported data (between quotes)

OUTPUT: nothing; the imported series are saved in the file named fileout

DESCRIPTION
Importation of an excel file saved under csv format.
* this is at the very beginning an original scilab function, but improved to be more efficient and extended to deal with ts
* if one data is named dates or DATES (scilab distinguish capitals from small letters) then the data that follow are saved as timeseries
* if a value is lacking or #N/A in a timeseries, it is given a NA value

Example:
impexcel('macros/bdexamples/bdhenderic.csv',';','mylibrary/bdhenderic.dat')

This example provides the importation of date in file macros/bdexamples/bdhenderic.csv into the Scilab file mylibrary/bdhenderic.dat. The separator is here ‘ ;’.

invpd_____mimic Gauss function invpd

CALLING SEQUENCE

[xi]=invpd(x)

PARAMETERS

INPUT:
x = a square matrix

OUTPUT:
xi = inverse of mat

DESCRIPTION
A dummy function to mimic Gauss invpd simply returns the inverse, with no checking for positive definiteness.

Example:
Y=invpd(x)

invvech_____reverses vech

CALLING SEQUENCE

[xi]=invvech(x)

PARAMETERS

INPUT:
* v = a (nrx1) vector
* nr = # of rows of the original matrix

OUTPUT:
* mat = the original -vectorized by vech- matrix

DESCRIPTION
Retrieves from a matrix vectorized by vech the original matrix.

Example:
a=grand(5,2,'nor',0,1);b=vech(a'*a);c=invvech(b,2)

C is equal to a'*a.

is_scalar_____check if a matrix is a scalar

CALLING SEQUENCE

[ret]=is_scalar(x)

PARAMETERS

INPUT:
x = a (nxm) matrix

OUTPUT:
ret0 = 1 if x is scalar, 0 in other cases

DESCRIPTION
Checks if a matrix is a scalar.

Examples:
1) is_scalar(5)
2) is_scalar(ones((4,1))

Example 1 returns 1, example 2 returns 0.

is_empty_____check if an object is empty

CALLING SEQUENCE

answ=is_empty(m)

PARAMETERS

INPUT:
m = a matrix, a list or a ts

OUTPUT:
answ = a boolean: %t if the object is empty, %f if not

DESCRIPTION
Checks if an object is empty. This is Scilab function isempty extended to time series.

Examples:
1) is_empty(reshape([1:5],'1a'))
2) is_empty(reshape([],'1a'))
3) is_empty([])

Example 1 returns %f, example 2 and 3 return %t.

joinstr_____concatenation of strings and strings vectors

CALLING SEQUENCE

car=joinstr(varargin)

PARAMETERS

INPUT:
* objects which can be strings or column vectors of strings; each vector must have the same size
---
OUTPUT:
* car = a string

DESCRIPTION
(similar to the one with the same name in portable troll) Concatenates elements which can be strings or matrix of strings; when some arguments are vectors of string, then the function creates a string for each element of the vector by concatenating it with the arguments of size 1 (the last one notwithstanding), in the order they are given by the user; then the function concatenates these strings with the last element as a separator. Very useful, but for experts!

Examples:
1) namex=['x1' 'x2']; namec=['a1' '(a2-a1)'];str=joinstr('namec','*','namex','+')
2) joinstr('(',res('namex'),')*(',string(res('beta')),')+')+')'

Example 1 gives str='a1*x1+(a2-a1)*x2'. Example 2 (taken from prevstat) takes the names of variables from, say, an ols regression, multiply each name by the value of the corresponding estimated coefficient and them adds all.

lag_____creation of a lagged matrix

CALLING SEQUENCE

[z]=lag(x,n,v)

PARAMETERS

INPUT:
* x = input matrix or vector, (nobs x k)
* n = order of lag
* v = (optional) initial values (default=0)

OUTPUT:
z = matrix (or vector) of lags (nobs x k)

DESCRIPTION
Creates a matrix or vector of lagged values.
* if n <= 0, z = [] is returned. While you may find this perverse, it is sometimes useful.

* for ts use lagts (see chapter 3)

Examples:
1) y = lag([1:8]',2)
2) x1=ones(7,1);x2=[1:7]';y = lag([x1 x2],1,4)

Example 1 gives the column matrix:
```
!  0. !
!  0. !
!  1. !
!  2. !
!  3. !
!  4. !
!  5. !
!  6. !
```

Example 2 gives the (7,2) matrix:
```
!  4.   4. !
!  1.   1. !
!  1.   2. !
!  1.   3. !
!  1.   4. !
!  1.   5. !
!  1.   6. !
```

lagser_____creation of a lagged matrix

CALLING SEQUENCE

[yl,ys]=lagser(y,ll)

PARAMETERS

INPUT:
* y = (nxm) matrix
* ll= a (nx1) or (1xn) vector of lags (>0 lags, <0 leads)

OUTPUT:
* yl = (n-ml)x(mxl) lagged and/or lead series, with ml=maxlag+abs(maxlead).
* ys = (n-ml)x(m) original series with the last ml observations suppressed.

DESCRIPTION
Generates lags and leads from a data matrix.

Examples:
1) y = lagser([1:8]',2)
2) x1=ones(7,1);x2=[1:7]';y = lag([x1 x2],-2)

Example 1 gives the column matrix:
```
!  0. !
!  0. !
```

```
!  1. !
!  2. !
!  3. !
!  4. !
!  5. !
!  6. !
```

Example 2 gives the (7,2) matrix:

```
!  4.   4. !
!  1.   1. !
!  1.   2. !
!  1.   3. !
!  1.   4. !
!  1.   5. !
!  1.   6. !
```

mak_____filters a series with a moving average

CALLING SEQUENCE

xma=mak(x,k)

PARAMETERS

INPUT:
* x = original time series
* k = size of the moving average fliter

OUTPUT:
xma = filtered timeseries

DESCRIPTION
Calculates a 2 sided k-period centered moving average with equal weights (k odd) or equal weights except for the first and last term whose weight is half the others (k even).

Examples:
lm1_sp=mak(series(lm1),10)

Takes a 10 terms moving average of the values vector of series lm1 taken from the database bdhenderic.dat.

matdiv_____matrix quotient

CALLING SEQUENCE

[out]=matdiv(x,y)

PARAMETERS

INPUT:
x,y = two matrices (not of the same dimension but that are row or column compatible)

OUTPUT:
out = x./y where x and y are row or column compatible

DESCRIPTION
Performs matrix quotient even if matrices are not of the same dimension, but are row or column compatible.

Examples:
1) x1=ones(7,1);x2=[1:7]';y = lag([x1 x2],1,4);z=0.5*ones(7,1);matdiv(x,z)
2) w = matdiv(x,resid);

Example 1 gives the matrix:
```
!  8.   8.  !
!  2.   2.  !
!  2.   4.  !
!  2.   6.  !
!  2.   8.  !
!  2.   10. !
!  2.   12. !
```

Example 2 (taken from lad) makes the term by term division of each column of matrix x ((nxk) matrix of exogenous variables) by the (nx1) vector of absolute residuals from the ols regression of y on x.

matmul_____matrix elementwise multiplication

CALLING SEQUENCE

[out]=matmul(x,y)

PARAMETERS

INPUT:
x,y = two matrices (not of the same dimension but are row or column compatible)

OUTPUT:
out = x.*y where x and y are row or column compatible

DESCRIPTION
Performs matrix multiplication even if matrices are not of the same dimension, but are row or column compatible.

Examples:
1) x1=ones(7,1);x2=[1:7]';y = lag([x1 x2],1,4);z=0.5*ones(7,1);matmul(x,z)
2) xstar = matmul(x,sqrt(w));

Example 1 gives the matrix:

```
!  2.    2.  !
!   .5    .5 !
!   .5   1.  !
!   .5   1.5 !
!   .5   2.  !
!   .5   2.5 !
!   .5   3.  !
```

Example 2 (taken from robust) makes the term by term multiplication of each column of matrix x ((nxk) matrix of exogenous variables) by the (nx1) vector of weights for the ongoing weighted least squares estimation.

meanc_____mean of the rows of a matrix

CALLING SEQUENCE

v=meanc(mat)

PARAMETERS

INPUT:
mat = a (nxp) matrix

OUTPUT:
v = a (px1) vector

DESCRIPTION
Function that mimics gauss function meanc: takes the mean of each column of a matrix and stores it in a column. Equivalent to Scilab instruction mean(mat,'r').

Examples:
1) x1=ones(7,1);x2=[1:7]';y = meanc([x1 x2])

Example 1 gives the matrix:

```
!  4.   4.   4.   4. !
!  1.   4.   1.   4. !
!  1.   1.   2.   1. !
!  1.   1.   3.   2. !
!  1.   1.   4.   3. !
!  1.   1.   5.   4. !
!  1.   1.   6.   5. !
```

Example 2 (taken from var1) takes the residuals from a regression and stores all lags from 1 to nobse/6 to prepare the calculation of the Box-Pierce statistics.

miss_____replace a specified value by %nan

CALLING SEQUENCE

v=miss(mat,val)

PARAMETERS

INPUT:
* mat = a (nxp) matrix
* val = a scalar

OUTPUT:
mat = a (nxp) vector

DESCRIPTION
Function that mimics gauss function miss: converts to NA prespecified values in a matrix

Example:
a = miss(eye(4,4),1)
returns:

Example 1 gives the matrix:

```
!  4.   4.   4.   4. !
!  1.   4.   1.   4. !
!  1.   1.   2.   1. !
!  1.   1.   3.   2. !
!  1.   1.   4.   3. !
!  1.   1.   5.   4. !
!  1.   1.   6.   5. !
```

Example 2 (taken from var1) takes the residuals from a regression and stores all lags from 1 to nobse/6 to prepare the calculation of the Box-Pierce statistics.

mlag_____lag of a matrix

CALLING SEQUENCE

[xlag]=mlag(x,n,init)

PARAMETERS

INPUT:
* x = a matrix (or vector), nobs x nvar
* n = # of contiguous lags for each vector in x
* init = (optional) scalar value to feed initial missing values (default = 0)

OUTPUT:
* xlag = a matrix of lags (nobs x nvar*nlag)

* x1(t-1), x1(t-2), ... x1(t-nlag), x2(t-1), ... x2(t-nlag),...

DESCRIPTION
Generates a matrix of n lags from a matrix (or vector) containing a set of vectors (for use in var routines).

Examples:
1) x1=ones(7,1);x2=[1:7]';y = mlag([x1 x2],2,4)
2) elag = mlag(rols('resid'),nobse/6);

Example 1 gives the matrix:
```
! 4.  4.  4.  4. !
! 1.  4.  1.  4. !
! 1.  1.  2.  1. !
! 1.  1.  3.  2. !
! 1.  1.  4.  3. !
! 1.  1.  5.  4. !
! 1.  1.  6.  5. !
```

Example 2 (taken from var1) takes the residuals from a regression and stores all lags from 1 to nobse/6 to prepare the calculation of the Box-Pierce statistics.

mlagb _____lag a matrix

CALLING SEQUENCE

[xlag]=mlagb(x,n,init)

PARAMETERS

INPUT:
* x = a matrix (or vector), nobs x nvar
* nlag = # of contiguous lags for each vector in x
* init = (optional) scalar value to feed initial missing values (default = 0)

OUTPUT:
* xlag = a matrix of lags (nobs x nvar*nlag)
* x(t-1),x(t-1) ... ,x(t-nlag)

DESCRIPTION
Generates a matrix of n lags from a matrix (or vector) containing a set of vectors (For use in var routines).

Examples:
1) x1=ones(7,1);x2=[1:7]';y = mlagb([x1 x2],2,4)
2) xlag = mlagb(y,nlag);

Example 1 gives the matrix:
```
! 4.  4.  4.  4. !
! 1.  1.  4.  4. !
! 1.  2.  1.  1. !
! 1.  3.  1.  2. !
```

```
!  1.   4.   1.   3. !
!  1.   5.   1.   4. !
!  1.   6.   1.   5. !
```

Example 2 (taken from var1) takes all lags from 1 to nlag of endogenous variables in the var to form the matrix of regressors.

mvblockboot_____multivariate block-bootstrap

CALLING SEQUENCE

[bsdata,a] = mvblockboot(data,sb,B)

PARAMETERS

INPUT:
* data = matrix of data
* sb = size ob blocks
* B = number of draws

OUTPUT:
* bsdata = resampled data
* a= corresponding columns

DESCRIPTION
Computes multivariate block-bootstrap.

newey_west_____Newey_West variance estimator

CALLING SEQUENCE

nw=newey_west(resid,l)

PARAMETERS

INPUT:
* x = input matrix or vector, (nobs x k)
* l = order of lag (optional)

OUTPUT:
nw = matrix (or vector) of lags (nobs x k)

DESCRIPTION
Performs the Newey_West variance estimator; if no truncation lag (l) is provided by the user, then it is fixed at floor(5*nobs^0.25).
* if l <= 0, nw = [] is returned. While you may find this perverse, it is sometimes useful.
* should presumably be made more efficient

Example:
sigma2=newey_west(r('resid'),l)

This example (taken from kpss) calculates the Newey-West estimator for the residuals (r('resid')) of the regression of the tested variable on a trend, with truncation lag l.

packr_____delete rows containing Nas

CALLING SEQUENCE

mat=packr(mat)

PARAMETERS

INPUT:
mat = a (nxp) matrix

OUTPUT:
mat = a (mxp) vector with m <= n

DESCRIPTION
Function that mimics gauss function packr: Deletes the rows of a matrix that contain any missing values.

Example:
v=[ones(7,1) [1:7]'] ;v(3,2)=%nan;w= packr(v)

Example gives the matrix:
w =

```
!  1.   1. !
!  1.   2. !
!  1.   4. !
!  1.   5. !
!  1.   6. !
!  1.   7. !
```

polymult_____mimics gauss function polymult

CALLING SEQUENCE

vecout=polymult(vec1,vec2)

PARAMETERS

INPUT:
* vec1 = a (nx1) vector of coefficients representing a polynom
* vec2 = a (mx1) vector of coefficients representing a polynom
---

OUTPUT:
vecout = a ((n+m)x1) vector of coefficients representing a polynom

DESCRIPTION
Function that mimics gauss function polymult: polymult = Scilab function convol ! (but with greater precision).

Example:
polymult([1 2 1],[1 -2])

Example gives the matrix:
ans =

```
!   1.    0.  - 3.  - 2. !
```

repmat_____replicates and tiles an array

CALLING SEQUENCE

[B]=repmat(A,m,n)

PARAMETERS

INPUT:
* A = a matrix
* m = a scalar or a (1xk) vector
* n = a scalar (used only if m is a scalar)

OUTPUT:
B = the replication and tiling of A

DESCRIPTION
Function that mimics Matlab function repmat, that replicates and tiles an array.

Example:
A = [1 2 3 ; 4 5 6];B=repmat(A,2,3)

Example gives the matrix:
B =

```
!   1.    2.    3.    1.    2.    3.    1.    2.    3. !
!   4.    5.    6.    4.    5.    6.    4.    5.    6. !
!   1.    2.    3.    1.    2.    3.    1.    2.    3. !
!   4.    5.    6.    4.    5.    6.    4.    5.    6. !
```

reshape_gauss_____replicates and tiles an array

CALLING SEQUENCE

[B]=reshape_gauss(A,m,n)

PARAMETERS

INPUT:
* matin = a (pxq) matrix
* n = # of rows of the destination matrix
* m = # of cols of the destination matrix

OUTPUT:
matout = the (nxm) destinaion matrix

DESCRIPTION
Function that mimics gauss function reshape: replicates and tiles an array.
Note that:
1) function reshape exists in grocer, but it mimics troll, and not gauss, function reshape, that is performs the transformation of a vector into a ts.
2) is equivalent to scilab function matrix only when the destination matrix has the same number of elements that the original one.

Example:
A = 1:4;B=reshape_gauss(A,3,5)

Example gives the matrix:
B =

```
!  1.   2.   3.   4.   1. !
!  2.   3.   4.   1.   2. !
!  3.   4.   1.   2.   3. !
```

Example:
A = 1:4;B=reshape_gauss(A,3,5)

Example gives the matrix:
B =

```
!  1.   2.   3.   4.   1. !
!  2.   3.   4.   1.   2. !
!  3.   4.   1.   2.   3. !
```

rev_____reverses the order of the rows of a matrix

CALLING SEQUENCE

[B]=rev(A)

PARAMETERS

INPUT:
A = a (nxp) matrix

OUTPUT:
B = a (nxp) matrix

DESCRIPTION
Mimics gauss function rev: reverses the order of the rows of a matrix.

A = [1 2 ; 3 4 ; 5 6];B=rev(A)

Example gives the matrix:
B =

```
!   5.    6. !
!   3.    4. !
!   1.    2. !
```

seqa_____sequence of values

CALLING SEQUENCE

[seq]=seqa(a,b,c)

PARAMETERS

INPUT:
a = initial value in sequence
b = increment
c = number of values in the sequence

OUTPUT:
a sequence, (a:b:(a+b*(c-1)))' in scilab notation

DESCRIPTION
Produces a sequence of values (a Gauss compatibility function).

Example:
seqa(2,5,3)

Example gives the matrix:

```
!   2.  !
!   7.  !
!   12. !
```

search_cte_____finds the indexes of constant columns

CALLING SEQUENCE

indcte=search_cte(x)

PARAMETERS

INPUT:
x = a (n x k) real matrix

OUTPUT:
indcte= a (p x 1) vector of integers

DESCRIPTION
Finds the indexes of the constant columns in a matrix.

Example:
search_cte([grand(15,2,'nor',0,1) ones(15,1)])

Gives 3.

shiftr_____shifts rows of a matrix and replace with a prespecified value

CALLING SEQUENCE

[matout]=shiftr(matin,shifts,val)

PARAMETERS

INPUT:
* matin = input (mxn) matrix
* shifts = (mx1) vectors of shifting values
* val = value to fill the holes

OUTPUT:
matout = (mxn) transformed matrix

DESCRIPTION
Function that mimics gauss function shiftr: shifts rows of a matrix according to a vector of shifts and fills the holes with a prespecified value.

Example:
A = matrix([1:15],3,5);B=shiftr(A,[2 3 -2],0)

Example gives the matrix:
B =

! 0. 0. 1. 4. 7. !

```
!  0.   0.    0.    2.   5. !
!  9.   12.   15.   0.   0. !
```

sortc_____sorts a matrix according to the value of one of its columns

CALLING SEQUENCE

[matout]=sortc(matin,val)

PARAMETERS

INPUT:
* matin = input (nxp) matrix
* val = value to fill the holes

OUTPUT:
matout = (nxp) transformed matrix

DESCRIPTION
Mimics gauss function sortc: sorts a matrix according to the value of one of its columns.

Example:
A=[1 3 2 4; 5 1 0 2 ; 4 2 1 2];B=sortc(A,3)

Example gives the matrix:
B =

```
!  5.   1.   0.   2. !
!  4.   2.   1.   2. !
!  1.   3.   2.   4. !
```

spencer_____Filters a series using a 15-term two-sided Spencer filter

CALLING SEQUENCE

xsp = spencer(x)

PARAMETERS

INPUT:
x = a (n x 1) real vector

OUTPUT:
xsp = a (n x 1) real vector

DESCRIPTION
Filters a series using a 15-term two-sided Spencer filter.

Example:
lm1_sp=spencer(series(lm1))

str2mat_____fusion of string vectors

CALLING SEQUENCE

m=str2mat(arg1,...,argn)

PARAMETERS

INPUT:
argi = a string or a vector of strings

OUTPUT:
m = a (px1) vector of strings

DESCRIPTION
Function that mimics matlab function str2mat: forms the column vector fusion of input string (row or column) matrices.

Examples:
str2mat(['a1' 'a2'],'b1',['c1';'c2';'c3'])

Example gives the vector:
str2mat(['a1' 'a2'],'b1',['c1';'c2';'c3'])
ans =

```
!a1  !
!    !
!a2  !
!    !
!b1  !
!    !
!c1  !
!    !
!c2  !
!    !
!c3  !
```

stcd_____standard deviations of each column of a matrix

CALLING SEQUENCE

v=stcd(mat)

PARAMETERS

INPUT:
mat = a (nxp) matrix

OUTPUT:
v = a (px1) vector

DESCRIPTION
Function that mimics gauss function stcd: takes the standard deviations of each column of a matrix and stores them in a column.

Examples:
load('SCI/macros/grocer/db/bdhenderic.dat') ;
mat=explox(['delts(lm1)','delts(lp)','delts(ly)','rnet'],'',[]);v=stcd(mat)

Example gives the vector:
v =

! .0205913 !
! .0141534 !
! .0142649 !
! .0376221 !

str2vec_____string to vector of strings

CALLING SEQUENCE

[s]=str2vec(name)

PARAMETERS

INPUT:
name = a string

OUTPUT:
s = a (n x 1) vector of names

DESCRIPTION
Given a string 'xxx=[yy1,...,yyn]', define the vector ['yy1';...;'yyn'].

Examples:
1) str2vec(‘a=b,c,d’)
2) grocer_coef=str2vec(grocer_coef)

Example 1 gives the string (3,1) vector:
!b !
!c !

!d !

In example 2, taken from nls, str2vec is used to transform into a vector the list of coefficient names given by the user.

studentize_____studentize a vector or columnwise a matrix

CALLING SEQUENCE

[t]=studentize(x)

PARAMETERS

INPUT:
x = a vector or matrix

OUTPUT:
t = transformed matrix

DESCRIPTION
If x is a vector, substract its mean and divide by its standard deviation. If x is a matrix, do the above for each column.

Example:
y=studentize(x)

tdiff_____produce matrix differences

CALLING SEQUENCE

[dmat]=tdiff(x,k)

PARAMETERS

INPUT:
* x = input matrix (or vector) of length nobs
* k = lagged difference order

OUTPUT:
* dmat = matrix or vector, differenciated by k-periods, e.g. x(t) - x(t-k), of length nobs, (first k observations are zero)

DESCRIPTION
Produces matrix differences.

Examples:
1) x1=ones(7,1);x2=[1:7]';y = tdiff([x1 x2],2)
2) dep = tdiff(r,1);

Example 1 gives the matrix:
! 0. 0. !
! 0. 0. !
! 0. 2. !
! 0. 2. !
! 0. 2. !
! 0. 2. !
! 0. 2. !

In example 2, taken from cadf, vector r is differenciated once.

transdif_____differenciation and Box-Cox transformation

CALLING SEQUENCE

[yt]=transdif(y,lambda,d,ds,s)

PARAMETERS

INPUT:
* y = (nxk) data matrix
* lambda = a scalar, parameter of the Box-Cox transformation
* d = a scalar, the number of regular differences (1-B)^d
* ds = (Sx1) matrix containing the number of seasonal differences
* s = (rx1) matrix containing the seasonal periods

OUTPUT:
yt =(n-d-sum(ds*s))xk matrix of transformed data

DESCRIPTION
Takes the difference and the seasonal difference of the Box-Cox transformation of a vector.

Examples:
1. mtlb_load('SCI/scied/grocer/encours/e4sci/seriesa.dat'); elec_cons = transdif(seriesa,0,1,1,12);
2. mtlb_load('SCI/scied/grocer/encours/e4sci/seriesa.dat'); s = transdif(seriesa,-1,0,0,12);

Example 1 taken from varma function varma_d1(). Elec_cons is equal to (1-L)(1-L^12)log(seriesa).
Example 2 gives simply the box-cox transformation of seriesa with lambda = -1 (that is 1-1/seriesa)

trend_____exponentiated time trend

CALLING SEQUENCE

[xmat]=trend(p,nobs)

PARAMETERS

INPUT:
* p = exponent of the time-trend
* nobs = size of the matrix

OUTPUT:
xmat = (nobsx1) vector containing trend

DESCRIPTION
Produces a vector equal to a time-trend with exponentiation p.

Example:
ext=ptrend(2,10)

This example gives the matrix:
```
!  1.  !
!  4.  !
!  9.  !
!  16. !
!  25. !
!  36. !
!  49. !
!  64. !
!  81. !
!  100. !
```

trimr_____strip a matrix

CALLING SEQUENCE

[z]=trimr(x,n1,n2)

PARAMETERS

INPUT:
* x = input matrix (or vector) (n x k)
* n1 = first n1 rows to strip
* n2 = last n2 rows to strip

OUTPUT:
* z = x(n1+1:n-n2,:)

DESCRIPTION
Returns a matrix (or vector) x stripped of the specified rows (modelled after Gauss trimr function).

Examples:
1) trimr([1:10]',2,3)

This example gives the matrix:

! 3. !
! 4. !
! 5. !
! 6. !
! 7. !

undrift_____removes the drift or a linear time trend

CALLING SEQUENCE

[xun]=undrift(x)

PARAMETERS

INPUT:
x =
- a data matrix x where columns represent different variables, x is (T x # variables).
- or a ts

OUTPUT:
xun =
- if x is a matrix, then a data matrix of the same size as x with a different drift/trend
- a ts with drift/trend removed if x is a ts

DESCRIPTION
Function that removes the drift or a linear time trend.

Example:
undrift([1:15]')

Provides ones(15,1).

var2cor_____covariance to correlation

CALLING SEQUENCE

[c]=var2cor(s)

PARAMETERS

INPUT:
s = (n x n) covariance matrix

OUTPUT:
c = (n x n) correlation matrix

DESCRIPTION
Transforms a covariance matrix in a correlation matrix.

varcov0_____covariance matrix

CALLING SEQUENCE

m=varcov0(z)

PARAMETERS

INPUT:
z = a (nxk) matrix

OUTPUT:
m = the (kxk) variance-covariance

DESCRIPTION
Calculates the covariance matrix of z columns.

vec_____stacks columns of a matrix

CALLING SEQUENCE

[v]=vec(x)

PARAMETERS

INPUT:
x = an input matrix

OUTPUT:
v = output vector containing stacked columns of x

DESCRIPTION
create a column vector by stacking columns of x. V=vec(x) is equivalent to scilab command v=x(:)

vec2col_____vector into column vector

CALLING SEQUENCE

[x]=vec2col(x)

PARAMETERS

INPUT:
* x = (1 x n) or (n x 1) vector

OUTPUT:
* x = (n x 1) vector

DESCRIPTION
Transforms a row or col vector into a col vector.

Examples:
1) vec2col([3:7])
2) vec2col([3:7]')
2) grocer_init=vec2col(grocer_init)

Examples 1 and 2 give the matrix:
```
!  3. !
!  4. !
!  5. !
!  6. !
!  7. !
```

Example 3, taken from nls, transforms into a column vector the vector of starting values for the optimisation algorithm. This allows the user to give either a column or a row vector and therefore facilitates her life.

vec2row_____vector into row vector

CALLING SEQUENCE

[x]=vec2row(x)

PARAMETERS

INPUT:
x = (1 x n) or (n x 1) vector

OUTPUT:
x = (1 x n) vector

DESCRIPTION
Transforms a row or col vector into a row vector.

Examples:
1) vec2row([3:7])
2) vec2row([3:7]')

Examples 1 and 2 give the matrix:
```
! 3.   4.   5.   6.   7. !
```

vech_____stacks a symmetric matrix

CALLING SEQUENCE

[v]=vech(x)

PARAMETERS

INPUT:
x = an input matrix

OUTPUT:
v = output vector containing stacked columns of x

DESCRIPTION
Creates a column vector by stacking columns of x on and below the diagonal.

Example:
vech([1 0.5 0.2; 0.5 2 0.4 ; 0.2 0.4 3])

This example gives the vector:
1.
0.5
0.2
2.
0.4
3.

vech_gauss_____stacks a symmetric matrix as in gauss

CALLING SEQUENCE

[v]=vech_gauss(x)

PARAMETERS

INPUT:
x = an input matrix

OUTPUT:
v = output vector containing stacked columns of x

DESCRIPTION
Creates a column vector by stacking columns of x on and below the diagonal.

Example:
vech_gauss([1 0.5 0.2; 0.5 2 0.4 ; 0.2 0.4 3])

This example gives the vector:
1.
0.5
2.
0.2
0.4
3.

vecr_____stacks rowwise

CALLING SEQUENCE

[v]=vecr(x)

PARAMETERS

INPUT:
x = an input matrix

OUTPUT:
v = output vector containing stacked rows of x

DESCRIPTION
Creates a column vector by stacking rows of x.

xpnd1_____mimics gauss function xpnd

CALLING SEQUENCE

mat=xpnd1(v,nr)

PARAMETERS

INPUT:
* v = a (nr x 1) vector
* nr = # of rows of the original matrix

OUTPUT:
mat = the original -vectorized by vech_gauss- matrix

DESCRIPTION
Retrieves from a matrix vectorized by vech_gauss the original matrix.

Example:
xpnd1([1;0.5;2;0.2;0.4;3])

Provides the (3x3) matrix:

1.	0.5	0.2
0.5	2.	0.4
0.2	0.4	3.