MLAB

A Mathematical Modeling Laboratory

MLAB Reference Manual

by

Gary Knott and Daniel Kerner

Revision Date: November 18, 2015

Civilized Software, Inc.
12109 Heritage Park Circle
Silver Spring, MD 20906, USA
Tel: (301)962-3711
Email: csi@civilized.com
URL: http://www.civilized.com

Table of contents

Table of MLAB Builtin Functions and Operators

MLAB consists of the base MLAB system together with twelve *extension packages*. These extension packages are:

1. ordinary-differential equation (ODE) solving
2. curve-fitting
3. optimization
4. advanced linear algebra
5. special functions
6. signal analysis functions
7. cluster analysis functions
8. density functions, distribution functions, and inverse distribution functions
9. statistical computations and tests
10. splines and smoothing
11. random number generators
12. 3D graphics and contour maps

The functions throughout this index are tagged with the number (1-12) of the extension package to which they belong. The functions of the base MLAB system are tagged with the value 0.

1. Introduction to MLAB

MLAB is a computer program whose name is an acronym for "Modeling LABoratory". It is a tool for experimentation with and exploration of mathematical models and for their evaluation. A mathematical model is a collection of functions potentially defined by algebraic and/or differential equations. MLAB also incorporates a variety of special mathematical, statistical, matrix manipulation, input/output, and graphics operators.

MLAB was originally designed and programmed by Gary Knott and Douglas Reece at the National Institutes of Health. It was inspired by earlier work done by Richard Shrager and John Fletcher at the NIH. It was significantly enhanced from 1973 until 1984, with continuing contributions by Richard Shrager, Marvin Shapiro, George Hutchinson, and others. In 1985, Gary Knott and Barry Bunow formed Civilized Software Inc. At CSI, MLAB has been rewritten in C and substantially revised and enhanced by Barry Bunow, Gary Knott, Daniel Kerner, Judy Graves, Pedja Bogdanovich, George Pick, Zhiping You, Jerome Eastham, Glenn Pearson, and Charles Hultcher. MLAB now runs on IBM PC's and compatibles under DOS and MS-WINDOWS 3.1/95/98/Me/2000/NT/XP . MLAB is also available for Macintosh II and Power-PC systems running the Macintosh operating systems through OS 9.x. MLAB also runs with NeXTstep on Motorola, Intel, and HP PA-RISC machines, and with X-windows on LINUX-Intel machines, and on Silicon Graphics Iris machines. MLAB is marketed by Civilized Software, Inc. 12109 Heritage Park Circle, Silver Spring MD, 20906 USA. Telephone: (301) 962-3711. URL: `www.civilized.com`

MLAB is an interactive system for mathematical modeling. The major components of the system are:

- a curve-fitting procedure which will adjust the parameters of a model function to minimize the sum of the pth power of the weighted absolute errors
- a robust differential equation system solving procedure
- an extensive repertoire of mathematical and statistical functions and operators
- a collection of routines for the construction and display of pictures
- mechanisms for saving information between sessions

The combination of these facilities with an interactive interpretive shell provides a powerful and convenient environment for data manipulation and analysis, mathematical and statistical calculations, and for building and testing models.

The user communicates with MLAB by typing commands or by executing prepared scripts of commands called do-files. Most commands are executed at once, but a few, such as the save command, may prompt the user for additional information. Should the user have questions, typing HELP will provide access to a text database about MLAB.

One of MLAB's main uses is to fit a model to data. Curve-fitting is a useful analytical tool in many diverse disciplines. The basic notion is easily described. Given data, say various points in the plane $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, and a function $y = f(x)$ where f involves some parameters, say a and b, as for example in $f(x) = ax^b + 1$, we may wish to calculate values for the parameters a and b so that the function f well-predicts the observed data, that is, so that $f(x_i) \approx y_i$ for $1 \leq i \leq n$. In this case, we say we have fit the *model* f to the data. The end result is merely the values obtained for the initially unknown parameters. The same principles apply in higher dimensions. The notion of well-prediction which is generally used is that of minimizing the sum of squares,

$$S(a, b) = \sum_{i=1}^{n} (f(x_i) - y_i)^2$$

by appropriately choosing values for the parameters a and b. This is because least-squares minimization is a well-studied method which allows various theorems involving error and uniqueness to be invoked.

There are various algorithms which may be employed to estimate the parameters which minimize a sum-of-squares value. One of the most robust is the carefully-tuned Marquardt-Levenberg method used in MLAB.

When curve-fitting is considered, there are various problems which must somehow be dealt with. First, there must exist some way to carry out the calculations. This generally means that a digital computer must be used, and that some programming is needed. Curve-fitting is generally a complex iterative activity. The modeler will usually wish to try various models with various initial guesses applied to various sets of data. Different weights may be tried, and data may need to be scaled or otherwise transformed, as may the model. Generally the most convenient way to view the results is in the form of a graph of the best-fit model (the theoretical curve) and the plotted data points (the observed curve). Often the computer programs employed are neither sufficiently general nor easy to use, and the curve- fitting process becomes a tedious exercise. MLAB is designed to overcome these difficulties. MLAB is particularly suited for quick and convenient curve-fitting, moreover MLAB provides powerful facilities for data manipulation and graphics as well.

Models are expressed as real-valued functions of arbitrary numbers of independent real variables and parameters. These functions may be expressed in terms of the built-in operators and functions, other functions defined by the user, and recursively in terms of themselves.

The curve-fitting routine requires the derivatives of the model with respect to the parameters to be adjusted. If these derivatives are not specified by the user, a symbolic differentiation subroutine computes them when possible. Derivatives are themselves functions and may be dealt with and used as such in computing, curve-fitting, and drawing operations.

Systems of differential equations may be specified and solved or used to implicitly define functions to be used as models in curve- fitting.

The graphics facilities embedded in MLAB provide a means of rapidly examining the results of a curve fit on the computer display screen, as well as of preparing an elaborate graph for publication. A finished graph may be reproduced on a laser printer.

The most salient feature of MLAB is its civilized interface with human users. The statements are simple and direct and all unnecessary details and trivia relating to pure programming have been made invisible to the user. Functions are evaluated interpretively in order to avoid the necessity for user programming.

A comprehensive set of assumptions or "defaults" are in force which frequently allow command components to be neglected when the most often desired alternatives are called for. The use of files is relatively clean and simple.

Such a civilized interface is quite costly in programming terms. A flexible input notation with error checking and default handling requires a large and intricate "shell" of procedures, which, although simple in principle, are tedious and tricky to construct. Indeed, this shell is nearly the same size as the routines which perform the core operations which are the raison d'etre of MLAB.

This manual is a reference manual for MLAB. Most users will find the other MLAB manuals more accessible. For those who wish to dive right into the formal documentation, the following overview section is provided, but after that, *lasciate ogni speranza, voi ch'entrate.* This is the bible of MLAB. Everything is defined in its own place. In other places, however, those things appear as unexplained terms.

There is an on-line "help" facility within MLAB. In order to view a screen of text about any MLAB function, operator, or statement, type help "topic-name". If you type just help, an initial menu of help choices will appear. This manual, however, is the preferred and definitive source for information about MLAB. The help facility material is often incomplete, and some topics are missing.

Published descriptions of the original MLAB system include:

Knott, G. D., and Reece, D. K., "MLAB: A Civilized Curve-Fitting System", Proceedings of the ONLINE '72 International Conference, Vol. 1, pp. 497:526, Brunel University, England, Sept., 1972.

Knott, G. D., and Shrager, R. I., "On-Line Modeling By Curve–Fitting", Computer Graphics: Proceedings of the SIGGRAPH Computers in Medicine Symposium, Vol. 6, No. 4, pp. 138:151, ACM, SIGGRAPH Notices, Winter 1972.

Knott, G.D., "MLAB–A Mathematical Modeling Tool", Computer Programs in Biomedicine, Vol. 10, No. 3, pp. 271:280, 1979.

It is preferable to cite the web-site *www.civilized.com* in publications referring to MLAB; it contains the most up-to-date information about MLAB.

2. Running MLAB

Initially, of course, you must carry-out the appropriate steps needed to install MLAB. You will need to obtain a registration key to register MLAB after it is run for the first time. For Unix or DOS versions, you should have modified the PATH shell environment string to allow the MLAB executable file to be located.

MLAB reads, writes, and creates files. Some of these files must be located in specific directories; others may reside in arbitrary directories of your choice. There are MLAB control string variables whose values determine where some files will be sought. Other files are always assumed to be in certain fixed directories.

There are three directories of interest used when running MLAB: (1) The MLAB-executable directory, (2) The current directory, and (3) The home directory (The concept of a home directory is primarily a Unix construct; for other operating systems, your home directory is the same as your current directoru.)

It is generally convenient to create a specific working directory for a project where your MLAB-related data-files and do-files for the project will be kept. Generally, this specific working directory should be the current directory when MLAB is run to work with the project associated with that directory.

MLAB commands may be directly typed into the MLAB console text window. (In the DOS version, the MLAB console text window is the entire screen.) Currently, aside from *backspace*, no special line-editing facilities are present in DOS, Windows, or Mac MLAB; do not use the arrow keys while typing commands. In the Unix version, you may use the left and right arrow keys to move the cursor around the string being entered, and the up and down arrow keys to scroll through commands previously entered.

On a DOS or Windows PC, all entered text will be converted to upper-case. For Unix systems or Macintoshes, this conversion to upper-case will only occur if the control variable *casesw* is set to 0. Lower case file names may be used on Unix and Macintosh systems. You may do this either by setting *casesw* to 1, or by enclosing the filename in quotes.

When an input error occurs, DOS, Windows, and Macintosh MLAB will invoke a special single-line editor which will allow you to correct the erroneous command. Within the single-line editor, you may use the left and right arrow keys to move the cursor appropriately. This feature is not yet supported in Unix MLAB.

For one-time exploratory computations, it is acceptable to directly type MLAB commands. **For more complex work, it is more convenient to construct a do-file with the text-editor facility in MLAB (under DOS, Windows, or Macintosh), or with a parallel-running editor program such as *emacs* in Unix environments.** The do-file created may be executed repeatedly and revised as desired.

Files used in MLAB

MLAB accesses or creates the following types of files, all of which are discussed below: log-files, startup-do-files, save-files, plot-files, print-files, read-files, do-files, and MLAB-control-files.

Save-files, print-files, plot-files, read-files, and do-files are all accessed in the directory specified by the MLAB control string FILEDIR, or in the current directory if FILEDIR is not set. On Unix systems, for example: if FILEDIR="/Users/George/" then "PRINT mat" will create the file /Users/George/mat.lst. If FILEDIR is not specified, the current directory is used. Be aware that the special pattern ~ may not be used in defining FILEDIR; MLAB has no understanding of this shorthand notation.

Do-files are handled with an added option; if the MLAB control string DOFILEDIR is set, the path specified therein will be used instead of FILEDIR to determine the directory where MLAB do-files will be sought. (These variables do not affect the automatic execution of mlabinit.do on startup.)

Log-files: `mlab.log`, `mlab0.log`, ...

When MLAB is run, it will create a log-file in the current directory. This log-file is always named mlab.log. If there is a previously-existing file named mlab.log in that directory, it will be renamed to `mlab0.log`; any file named `mlab0.log` will first be shifted to be `mlab1.log`; any file named `mlab1.log` will first be shifted to be `mlab2.log`; and any file named `mlab2.log` that is replaced by `mlab1.log` will be deleted. MLAB log-files are standard ASCII files; they may be edited, printed, and/or deleted using the facilities provided by your operating system or other software. One common use of a log-file is to modify it to build a do-file for repetitive use in MLAB. See the EDIT FILE command elsewhere in this manual for directions on such editing. See the PRINT command elsewhere in this manual for directions on printing an ASCII file.

Startup-do-files: `mlabinit.do`, `$HOME/.mlabrc`

In Unix, if the do-file `.mlabrc` is present in your home directory, it will be interpreted during startup when MLAB is run. Immediately thereafter, the do-file called `mlabinit.do` will be interpreted if it is present in the current directory.

In DOS, Windows, and Macintosh MLAB, the do-file `mlabinit.do` will be interpreted during startup when MLAB is run, if it is present in the current directory.

Save-files: `*.sav`

MLAB can create binary files holding MLAB values (scalars, matrices, windows, functions, initials, strings) via the SAVE command. These files are called save-files and have the extension *sav*. The values stored in a save-file can be read into MLAB by means of the USE command. Save-files are created and accessed in the directory specified by FILEDIR, or in the current directory if FILEDIR is not specified.

Plot-files: `mlabp#.ps`, `mlabp#.lj`

MLAB can create PostScript and HP LaserJet plot-files by using the PLOT command. The file names for PostScript plot-files have an *mlabp* prefix followed by a number and a *ps* extension (e.g. /tt mlabp1.ps
. The file names for LaserJet plot-files have an *mlabp* prefix followed by a number and an *.lj* extension. Plot-files are created and accessed in the directory specified by FILEDIR or in the current directory if FILEDIR is not specified.

Print-files: `mlab.lst`, `*.lst`, `*`

MLAB can create print files by using the PRINT command. Print-files have default name *mlab.lst* if no name is specified, and subsequent print operations will be appended to this file. A print-file has the default extension *.lst* if the file name is given with no extension. The full print-file name can also be given in quotes. Print-files are created and accessed in the directory specified by FILEDIR, or in the current directory if FILEDIR is not specified.

Read-files: `*.dat`

The READ and READON operators are used in MLAB to read numbers from a text-file. Such text-files are accessed in the directory specified by FILEDIR, or in the current directory if FILEDIR is not specified.

Do-files: `*.do`

Do-files are text-files containing MLAB commands, they are effectively MLAB programs and function as scripts. A do-file `z.do` is executed by typing `do z`. Do-files are accessed in the directory specified by DOFILEDIR. If DOFILEDIR is not specified, then do-files are accessed in the directory specified by FILEDIR; if neither of these path strings are set, do-files will be accessed from the current directory.

MLAB control files: `MLAB.HLP`, `gxfonts.0`, etc.

MLAB consists of an executable file and a number of control files. These files are found in the MLAB executable directory, and possibly in certain ancillary directories. These directories are fixed at the time MLAB is installed.

Running MLAB using Windows 95 and later

Like Windows 3.1, Windows 95, 98, ME, NT, 2000 and XP do not have a simply-accessible command language, and the notion of "changing directories" has been obscured. You must use various complex sequences to create directories and to specify a directory as the "current" directory. The following directions assume some understanding of the Windows GUI user facilities and jargon.

To create a directory using Windows 95, either double click on the My Computer icon in the "desktop" or run "Windows Explorer". Select the logical drive where the directory is to be created. Navigate to the parent directory you wish to create the new directory in, then create a new directory by selecting [NEW FOLDER] from the [FILE] menu. (Folders and directories are the same thing.)

To specify a desired current directory for MLAB using Windows 95, you will need a "shortcut" icon for MLAB. The installation program should have created this for you; if it for some reason it did not, or if you have lost the shortcut somehow, it may be recreated as follows; Using My Computer or Windows Explorer, navigate to the directory where you have installed MLAB. Right click on the MLAB.EXE file icon, and select [CREATE SHORTCUT] from the resulting menu. A shortcut file will appear in that folder; you may drag it to your "desktop" where it will remain thereafter.

After the MLAB shortcut icon has been created, proceed as follows. Select the icon for the MLAB Shortcut and click the *right* mouse button; select [PROPERTIES] from the resulting menu. Then select the [SHORTCUT] tab in the resulting "SHORTCUT TO MLAB PROPERTIES" window. Change the [START-IN] field to the fully-qualified directory name of to the directory that you wish to serve as a current directory. (i.e. C:\MLAB\PROJECT1)

If you intend to work on multiple projects, you may find it helpful to create several shortcuts to the MLAB program, each with an unique working directory.

If you do not specify a current directory, then the current directory by default will be the MLAB-executable directory "\MLAB\". Using this directory is not a good option, as over time, it will become clogged with many do-files and data-files; moreover, you will not be able to keep your projects separated.

To run MLAB in Windows, double-click on the MLAB shortcut icon that you have placed in your ""desktop" area. Alternately, you may click on the START-button icon, and then on the [PROGRAMS] menu entry, and then on the [MLAB] menu entry, and then on the [MLAB.EXE] menu entry. Running MLAB by either of these methods will cause a movable, scrollable, and horizontally resizable MLAB Console text window to appear in the lower-left corner of your display screen. This Console window is used to echo the MLAB commands you type or otherwise specify to be echoed from a do-file via setting the ECHODO control variable, and also to display any computational results or messages produced by MLAB.

To quit MLAB, type "exit" at the MLAB prompt, or click the X button in the top right corner of the MLAB Console text window.

Windows Tips

How to make the text font large for Windows.

Generally, the font used is the same for all programs and is a global property. Some programs, however, set their own font without changing the global font.

If the text in your top-level MLAB window is too small to be easily legible, there are two options you may pursue to make it larger.

One solution is to reduce your display resolution. Fonts displayed in Windows are so-called bitmap fonts whose size is a function of the number of pixels per inch that are displayed. To change your resolution in Windows, you will need to do the following:

1) Click to open the Start Menu, go to Settings and click on the Control Panel entry in the pop-up submenu.

2) Double click the icon titled Display in the control panel to bring up the display properties window.

3) Click on the Settings tab to display the resolution and color depth settings window.

4) Drag the "Screen Area" slider there to a lower resolution setting. The selected resolution will be reported below the slider.

5) Click Apply when the selected resolution is acceptable.

6) Close the opened windows that are left on the screen as a result of modifying the resolution by clicking on the X buttons in the top right corners.

On some older computers you will need to reboot for the new resolution setting to take effect, but usually this will not be necessary.

Another option is to increase your Windows global font scale factor. This factor is applied to all fonts used by all applications, unless they specify their own scale factor. To do this, you can proceed as follows.

1) Click to open the Start Menu, go to Settings and click on the Control Panel entry in the pop-up submenu.

2) Double click the icon titled Display in the control panel to bring up the display properties window.

3) Click on the Settings tab to display the resolution and color depth settings window.

4) Click on the Advanced button in lower right corner to bring up the video adapter properties window.

5) Click on the General tab to display the font size control window, if it is not present by default.

6) Change the selected font size to 'Large Fonts'. This effectively sets the global font scale factor to 125whatever fonts are being used.

7) Click Apply to record the new font size if you are satisfied with it.

8) Windows may bring up a dialog window stating that you will need to reboot for the change(s) you have made to take effect, and give you the option to reboot immediately. You may also choose to defer rebooting to a later time. If the reboot dialog window does not appear, you will need to reboot for your changes to take effect. 9) If you do not reboot, close the opened windows that are left on the screen as a result of modifying the global font size by clicking on the X buttons in the top right corners.

With a large global font scale factor enabled, you may encounter problems with some poorly designed websites or applications because larger text may extend beyond the boundries pre-determined for it. Of course, reducing your font size back to the default setting of Small Fonts will resolve any such problems.

Running MLAB under MacOS

To run MLAB on a Macintosh system, locate the MLAB application icon. If you followed the installation instructions, this icon will be found in the folder titled *MLAB for Macintosh*. Double clicking on the MLAB application icon will launch MLAB. This will cause a scrollable and movable (but not resizeable) MLAB Console text window to be opened in the lower left portion of the primary display.

To quit MLAB for Macintosh systems, type "exit" at the MLAB prompt, by pressing Apple-Q, or by selecting Quit from the File Menu.

Running MLAB under Unix

To run MLAB on a standard Unix/X-windows system, you must first establish a set of X-server processes (for example, running "startx" in linux), and then run a terminal emulator process such as XTERM. Switch to that terminal window and change directory *(cd)* to the desired working directory. Then you can issue the command "mlab" to run MLAB.

To exit MLAB in Unix, type "exit" at the MLAB prompt.

Running MLAB using NeXTSTEP

In the NeXTSTEP operating system, MLAB may be run from a terminal emulator window; change directory *(cd)* to the desired working directory and issue the command "mlab". MLAB can also be run from the NeXTSTEP Workspace (i.e., "desktop") by either double-clicking on the MLAB icon, or by double-clicking on the name of an MLAB do-file in the file browser window. In this latter case MLAB will be automatically supplied with the do-command to execute the selected do-file. In either case, when MLAB is run from the NeXTSTEP Workspace, the current directory will be your home directory as determined when you logged in. If your home directory could not be found at login time, the system will have assigned the top-level root directory as your home directory. When MLAB is run from the NeXTSTEP workspace, a front-end MLAB interface process is run which, in turn, communicates with an MLAB "session" process. You may establish multiple MLAB sessions by clicking the Session→New menu item.

To exit MLAB on NeXTSTEP, type "quit" at the console to close individual MLAB sessions, or you may close all MLAB sessions as well as the front-end by typing Ctrl-Q, or selecting Quit from the MLAB window.

Running MLAB using DOS

To run MLAB, change directory *(cd)* to the current directory you wish to use, and then run MLAB by typing "mlab".

To quit MLAB when you are finished working, type the command "exit" at the prompt.

Running MLAB using Windows 3.1

Windows 3.1 does not have a command language, and the notion of "changing directories" has been obscured, so you must go through complex motions to create directories, and to specify a directory as the "current" directory. The following directions assume some understanding of the Windows 3.1 GUI user facilities and jargon.

To create a directory using Windows 3.1, select [CREATE DIRECTORY] from the [FILE] menu in the File Manager, and fill-in the directory name appropriately.

To specify the desired current directory for MLAB using Windows 3.1, click once on the MLAB Program icon in the MLAB Program Group that you display by running the Windows 3.1 Program Manager. From the Program Manager's [FILE] menu, select [PROPERTIES], and change the [WORKING DIRECTORY] field in the resulting dialog box to the path of the directory that you wish to serve as your current directory.

If you do not specify a current directory, then the current directory will be the MLAB-executable directory "\MLAB\". This is not a good option for everyday use since over time this directory will become clogged with many do-files and data-files. Moreover, you will not be able to keep your projects separated.

Now, to run MLAB in Windows 3.1, double-click on the MLAB Program icon in the MLAB Program Group window which is displayed using the Windows Program Manager. This will cause a movable, scrollable, and resizable MLAB Console text window to appear in the lower-left corner of your display screen. This Console window is used to echo the MLAB commands you type or otherwise specify to be echoed from a do-file via setting the ECHODO control variable, and also to display any computational results or messages produced by MLAB.

To quit MLAB in Windows 3.1, you may type "exit" at the MLAB prompt, double clicking the upper left corner of the MLAB window, or selecting Quit from the MLAB menu.

Using MLAB

When MLAB runs, it will create a log-file in your current directory. (You can later print this log-file, and even copy it and extract output or other text to be placed in some document, and/or modify it to form a do-file, if you choose to do so.) You may now enter commands interactively, or, for more substantial work, you may create, run and revise a do-file.

Note that you will need to respecify your current directory whenever it is to be different from the current directory that has been set via the processes specified above. There are several easy, but temporary ways to specify a current directory for MLAB. First, you can *always* explicitly specify the location of a do-file or data-file that you wish to read (or write) within MLAB by stating the full path-qualified name of the file. Thus, if you wish to execute the do-file `xd.do` in the `s1` sub-directory of the `ss0` directory in the `D:` logical drive, you may type the MLAB command DO "D:\SS0\S1\XD.DO". You may also use the FILEDIR (or DOFILEDIR) control string. If you type the command: FILEDIR = "D:\SS0\S1", then *all* subsequent file-accesses will refer to files in the `s1` sub-directory of the `ss0` directory in the `D:` logical drive. Essentially, the string FILEDIR is prepended to every filename used in MLAB commands; its initial value is a null string.

To create a do-file, use the MLAB EDIT FILE command; this can be done, for examplee, by typing EDIT FILE x. On Windows PCs, this will pop-up the Windows Notepad editor in a new window.. On Macintosh systems, a similar editor program called Simpletext will be run in a new window. After you have created a .do file, save it to your current directory.

In the DOS version of MLAB, the built-in MLAB editor will be brought up - this works similarly to the external editors of other versions, except you must quit the editor with the F2 or F3 key in order to return to the MLAB Console text window.

The EDIT FILE feature does not yet work in Unix. You should merely run your favorite editor in a separate X-terminal window.

On Windows and Macintosh systems, it is often convenient to leave the editor program running side by side with MLAB. You can then test the do-file `x.do` by typing `do x` in the MLAB Console text window; if the `x.do` do-file requires modification, you need only switch back to the suspended editor window, modify the file, resave it in your current directory, switch back to MLAB, and then execute `x.do` again. Under DOS, the MLAB built-in editor is run, and you may behave in the same way, however you *must* completely exit the editor by typing the F2 or F3 key before you go back to MLAB. This iterative process is often the most convenient way to construct and debug a do-file. The do-file will then be available for repetitive use, if this is desired.

3. An Overview of MLAB

using mlab as a calculator

Statements such as

```
TYPE 5 + 7*2/SQRT(3)
```

will cause the appropriate arithmetic to be performed and the answer to be typed. Lists of the built-in operators such as +, and functions such as the square root function SQRT, appears under the discussion of operators and functions. The keyword TYPE is optional.

defining scalars and matrices

Scalars may be defined by statements such as

```
E = 2.71828
```

The equal sign (=) is used for the assignment operator. The underscore character (_) may also be used if you prefer a symbol for assignment different from (=), which also serves for equality testing. After an assignment you may then use the assigned variable, e.g. E, in computations. After the above assignment statement, the symbol E will continue to have value 2.71828 until the end of your session unless you do one of the following: The statement `DELETE E` will cause the symbol E to be forgotten. Another assignment statement, e.g. `E = 4` will set E to the new value 4. The FIT command (discussed below) behaves as a kind of assignment statement, since it will adjust the value of one or more variables if you request it to do so.

There are several ways to define matrices. Consider the following 3×2 matrix:

$$\begin{bmatrix} 1 & 5 \\ 1 & 13 \\ 3 & 47 \end{bmatrix}$$

A simple way to define this matrix to MLAB and give it the name M is to use the following two statements:

```
M COL 1 = LIST(1,1,3)
M COL 2 = LIST(5,13,47)
```

To change the element in row 1 column 2 of M to the value 76, type `M[1,2] = 76`. You may also use statements like this to define matrices. Any elements which are not defined will have the value 0. In defining matrices, it is often more convenient to create a corresponding file of numbers outside of MLAB using a text editor. The numbers can then be entered into MLAB using the READ operator. This method is less error-prone for large matrices than using the LIST operator as shown here.

functions and models

If you wish to do curve-fitting, you must define your model to MLAB as one or more functions. Suppose you wish to fit a straight line to some data. You might then choose the function $F(X) = A * X + B$ as your model and ask MLAB to find the best values of A and B.

To define the function F, type `FUNCTION F(X)= A*X+B`. You must give values to A and B (perhaps just guesses) before F can be evaluated. Type in statements like `A = 3.5` and `B = 4`. Then you may ask MLAB to evaluate F on some argument by typing a statement like `TYPE F(5)`.

curve-fitting

If you wish to do a fit, you must define at least two things to MLAB: the function to be fit and the data to be used. The function is defined as illustrated above. The data is entered in the form of a matrix. If your data and model have N independent variables, then the matrix will have N+1 columns. Columns 1 through N contain the values of the independent variables. Column N+1 contains the corresponding value of the dependent variable. There should be one row for each observation.

Suppose for example we have the following situation: the model to be fit is $H(X1, X2) = AX1 + BX2 + C$, and we have the following observations to be used in the fit.

```
[X1=2    X2=5      H=47]
[X1=13   X2=8      H=28]
[X1=19   X2=3.4    H=16]
```

Our best guesses for A, B, and C are 10, 4, and 16 respectively. The following sequence of commands will do the appropriate fit operation:

```
DATA COL 1 = (2&13&19)
DATA COL 2 = (5&8&3.4)
DATA COL 3 = (47&28&16)
FUNCTION H(X1,X2) = A*X1 + B*X2 + C
A = 10
B = 4
C = 16
FIT (A,B,C), H TO DATA
```

The first 3 statements above use the row concatenation operator (&) to define a matrix named DATA having the experimental values of the variables in the correct format. The function H which constitutes the model is then defined and the initial guesses for A, B, and C are established. The FIT statement given above requests MLAB to vary the parameters named A, B, and C in the function H and to fit to the data described by the matrix named DATA. The FIT statement will assume default values for various control parameters to the fit routine, unless you specify other values. The fitting process will then begin, and informative messages will be typed out during the process.

You may interrupt the fit by typing the letter Q (either upper or lower case will work). When the current iteration is completed the fitting process will be stopped if Q has been typed.

solving ordinary differential equations

The INTEGRATE operator is used to solve systems of first order differential equations. Suppose we have the following two coupled, first order differential equations with initial conditions.

$$\frac{dx}{dt}(t) = y;\ x(0) = x_0$$

$$\frac{dy}{dt}(t) = -x + \epsilon(1 - ky^2);\ y(0) = y_0$$

In MLAB these differential equations and initial conditions may be defined using function statements and initial statements. The derivative function $dx/dt(t)$ is written as `X DIFF T(T)` or `X' T (T)`. For the example above, we have:

```
FCT X'T(T) = Y;
INITIAL X(0) = X0;
FCT Y'T(T) = -X+EPS*(1-K*Y*Y);
INITIAL Y(0) = Y0;
```

Once X0, Y0, EPS, and `K` have been given values with appropriate assignment statements, this pair of differential equations may be solved using the integrate operator. The assignment statement

```
M = INTEGRATE(X'T,Y'T,TL)
```

where TL is a 1-column matrix of independent variable values, solves the differential equation system, and assigns the table of solution values at the points in TL to the matrix M. For example, you may compute TL as a 1-column matrix of 400 equally-spaced increasing values between 0 and 10 with the following assignment statement.

```
TL = 0:10!400
```

Upon completion of the integrate statement, M COL 1 is a copy of TL; M COL 2 contains the values of the solution X(T) at the values of T given in M COL 1; M COL 3 contains the values of X′ T(T) at the values of T given in M COL 1; M COL 4 contains the values of the solution Y(T) at the values of T given in M COL 1; and M COL 5 contains the values of Y′ T(T) at the values of T given in M COL 1.

Columns of the result matrix M can be selected for plotting. The rows of M COL (1,2) are points which lie on the solution curve for X(T). The rows of M COL(1,4) are points which line on the solution curve for Y(T). Plotting M COL (2,4) will show the phase plane trajectory of the system.

creating pictures

The DRAW command is used for constructing pictures to be shown on the computer display screen.

The DRAW command accepts a 2-column matrix. The first column holds the x-coordinates of the points to be drawn, and the second column holds the corresponding y-coordinates. If you wish to draw several curves, a separate DRAW command is used for each curve.

To draw graphs of functions of one variable, the POINTS function may be used. To draw the SIN function for the arguments 1,2,3,4,5 one might say DRAW POINTS(SIN,1:5). POINTS(F,V) is the matrix whose rows are points on the graph of F.

To draw the data with x-coordinates given by column 2 of the matrix M and y-coordinates given by column 4 of the matrix M, type DRAW (M COL 2)&′ (M COL 4). Each row of the matrix (M COL 2)&′ (M COL 4) is a pair of coordinates, i.e. a point to be plotted.

This form of the DRAW command will draw solid lines connecting the specified points. It is possible to draw symbols at each point, to omit the line, or to draw dashed lines.

The form of the DRAW command just described causes data to be drawn in the default window, which is named W. To delete all the displayed information associated with W, type DELETE W. Each time a new curve is drawn, it is given a name such as C1. A list of curve names in the window W will be given if you type TYPE W. You may delete just the curve named C1 by typing DELETE C1.

viewing pictures on the display

The DRAW command creates pictures but does not show them on the display screen. In order to view these pictures, the VIEW command is used. The window W will be drawn on the display. With DOS, it will remain there until the *ENTER* key is pressed. With Windows, Macintosh, NeXTstep or X-windows, it will remain there until it is explicitly removed by an UNVIEW command.

hardcopy plots

To obtain a file which is suitable for producing a hardcopy plot of a picture on a laser printer, the PLOT command is used. Type PLOT. A file named mlabpn.ps, or mlabpn.lj, depending on the plot

device selected, (the options are Postscript or HP-Laserjet) where n is a unique small integer, will be created. This file may be printed later.

command files and command strings

A text file on disk, called a do-file, may be prepared containing a sequence of one or more MLAB commands, separated by semicolons (;). Similarly, such a sequence of one or more MLAB commands may be entered into a string variable S. The sequence of commands may then be performed by typing DO D, or DO S, where D is the name of the do-file and S is the name of the string variable. Any MLAB statement may appear in such a sequence, including another do statement. In this way, elaborate and general algorithms may be programmed.

saving entities on disk

The MLAB command SAVE IN X will save a copy of the all extant MLAB entities on your disk area in a file whose name is X.sav.

The command USE X will make available, within the current MLAB session, copies of all the named data entities which were saved earlier (perhaps during a previous session) in the file X.sav. Matrices, scalars, functions, constraints, and windows may be saved in this way.

4. Scalars and Matrices

Scalar and matrix values may be created and manipulated by MLAB. An assignment statement may be used to create a scalar or matrix. For example the statement A = 15 will define a scalar with name A and value 15. Scalar values are real numbers represented in 64 bit floating point form. For a PC, a Macintosh, or a Sun SPARC, positive floating point values lie in the approximate range 10^{-308} to 10^{308}. Integers are just a special case of real numbers with no fractional part.

A matrix is a two-dimensional table of scalar elements with one or more rows, and one or more columns. Matrix elements have two subscripts. All subscripts must be positive integers. The first element of a matrix M is M[1,1]. The first subscript specifies the row and the second subscript specifies the column. An element of a matrix with a single column may be referred to as M[E], where E is the row number, using only a single subscript. In this case the implicit second subscript is 1. Thus vectors are just single column matrices.

The statement A[1,1] = 5 will define a matrix A and set its [1,1] element to 5. A is now a 1×1 matrix. If one now types A[3,3] = 25, A will be enlarged to a 3×3 matrix; the [1,1] element is 5, the [3,3] element is 25, and all other elements are 0.

MLAB provides many operations useful in building matrices. For example, the colon (:) notation $a : b$ denotes the list of values $a, a+1, \ldots, a+\lfloor b-a \rfloor$ (i.e. $\lfloor b-a \rfloor$) when $a \leq b$. Thus, 1:4 is used to define the following column matrix:

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

The double colon (::) notation $a : b : c$ is used to specify a sequence of numbers with a specified spacing between the successive numbers. The double colon notation $a : b : c$ denotes a list of values as (initial value):(final value):(increment). Thus, 1:2:.5 is the following column matrix:

$$\begin{bmatrix} 1 \\ 1.5 \\ 2 \end{bmatrix}$$

Another way to specify a 1-column matrix is by using the LIST operator. LIST(1,2:2.5:.25,4) is the following 1-column matrix:

$$\begin{bmatrix} 1 \\ 2 \\ 2.25 \\ 2.5 \\ 4 \end{bmatrix}$$

The colon-exclamation notation $a : b!c$ denotes a list of values as (initial value):(final value)!(number of points in the list). Thus, 0:10!5 is the 5×1 matrix with elements: 0, 2.5, 5, 7.5, 10.

Matrices may be created and extended in parts by using the ROW and COL operators. For example, if Y is a 4-column matrix, then X = Y COL 3 creates X as a 1-column matrix equal to the third column of Y. X = Y COL 4:1 creates X as a 4-column matrix whose ith column is equal to the $(5-i)$th column of Y.

A matrix may be created by typing, say, M = LIST(1,2,4) which makes M the appropriate column matrix. M COL 2 = 1:3 will make M a 3×2 matrix with the specified second column. For constructing a matrix with input data, the READ operator, which reads from a disk file, is often the best method.

5. Strings and String Variables

A list of characters is called a string. String constants are specified by enclosing the characters of the string within double quote characters ("). For example, "EGAC" is a string constant. String constants may be assigned to string variables, just as scalars are assigned. Strings have several uses in MLAB.

- Titles in graphs are strings.
- A string may be typed out in a TYPE statement. This may provide informative commentary associated with numerical output.
- Some MLAB control variables are strings. One of these is the FILEDIR string. This string is prefixed to file names in the DO, SAVE, USE, PRINT, and PLOT statements, and in the builtin function READ. The FILEDIR string enables the user to specify the disk drive and/or a directory path for files to be read or written.
- A string entered as a statement from the keyboard or in a do-file is treated as a comment. It is ignored. Such comments are useful in annotating the MLAB session log file or a do-file.
- A string may be created as an internal do-file, which may be subsequently executed in a DO statement.
- In the builtin function READ, the filename may be a string constant, or it may be the name of a string variable, in which case the contents of that string will be used as the file name. Strings may be read in with the builtin function KSREAD. In preprogramed applications, this is a good way to obtain the name of the file to be used subsequently by read.

Arrays of strings can be constructed in MLAB with direct assignment statements, e.g. S[1] = "A"; S[2] = "AB". Such string arrays are used as arguments to the MENUCHOICE and GETSTRINGS functions.

6. Functions and Derivatives

MLAB has various built-in functions such as SIN (the trigonometric sine function, sine(x)), EXP (the exponential function ex), and SQRT (the square root function $\sqrt{x}$), as well as user-defined functions. It also has various operators such as ^(the power operator), / (the divide operator), + (the addition operator), - (the subtraction operator), etc. which are written in Fortran-style infix notation. User-defined functions can be explicitly specified by using the FUNCTION statement. Moreover, functions can be created from existing functions by invoking the differentiation operation.

A user defined function can be defined in terms of other functions and scalar parameters, such as `FUNCTION F(X,Y) = A*SIN(X)+2*G(Y)`. In this case F is a function of two arguments, X and Y, and has one parameter, A. A parameter "belongs" to a function if it occurs in the body of the function, or in the body of a referenced function as a free variable. A free variable is one which appears in the body of a function but is not an argument given in the formal argument list. Thus, if the function G, referenced in the body of F above is defined as `FUNCTION G(X) = B*X/A`, then F has the parameters A and B.

Although various built-in functions explicitly yield matrices, user functions are always scalar-valued functions. The arguments of a user function may be scalars, real-valued functions of real arguments, or matrices. Matrices may occur in subscripted form in a function body. No operations which produce a matrix result are allowed, however.

Derivatives are, of course, just functions. For a function F of two arguments, X and Y, the derivative of F with respect to a parameter A which may occur directly or indirectly in the body of F is again a function of X and Y and has the name F DIFF A. Thus DIFF is an operator which takes a function name and a symbol name and produces another function name. F DIFF X is the partial derivative of F with respect to X with arguments X and Y, and F DIFF X(1,A) is the value of F DIFF X with X = 1 and Y = A.

Derivatives, as well as differential or difference equations can be defined explicitly by using the DIFF notation in naming the function. The symbol DIFF may be abbreviated by the single quote symbol ($'$). Thus F$'$ X is equivalent to F DIFF X.

Derivatives are specified by using the DIFF operator and computed symbolically when required.

A function may be defined as a piecewise combination of other functions by using the conditional form: IF B THEN P ELSE Q, where P and Q are expressions and B is a boolean expression. Moreover, recursive functions such as `FUNCTION F(X) = IF X = 0 THEN 1 ELSE (X-1)*F` are perfectly acceptable.

All function calls must normally have an argument list (possibly empty). However, a special convention is established in MLAB which defines the value of a function name occurring in a function body without an argument list to be the value of the function on the actual arguments used in the previous evaluation of the function . This convention is useful for sequentially evaluating recursive functions; it is also consistent with the methods used for solving differential and difference equations. However, functions which use this convention cannot always be automatically differentiated. The previous value of a function may be set by direct assignment. Thus, if F has been defined as a function, the assignment F = 3 sets the previous value of F to 3.

Systems of algebraic, differential, or difference equations may be specified but the consistency of such systems is not checked. The evaluation of such incorrectly-specified systems may lead to infinite loops.

MLAB computes the derivatives of functions symbolically. The differentiator handles arithmetic operators such as "+" and most of the built-in mathematical functions, such as cosine, which have derivatives. The chain rule is applied when necessary. Some simplifications are performed.

Standard rules are used for differentiating algebraic operators. These rules are given below. In the rules below, the operator D differentiates with respect to some unspecified variable of interest. Dx

denotes differentiation with respect to the symbol X. Di denotes the derivative of a function, treating it as a function of its ith argument, with respect to some unspecified variable of interest.

```
D(A+B) = D(A) + D(B)
D(A-B) = D(A) - D(B)
D(A*B) = A*D(B) + B*D(A)
D(A/B) = D(A)/B + A*D(B^-1)
D(A^B) = B*A^(B-1)*D(A)+LOG(A)*A^B*D(B)
```

The differentiation rules for some operators are somewhat arbitrary and may not give the result you wish. These special cases are the following:

```
D(A=B) = (A = B)
D(A<B) = (A = B)
D(A>B) = (A = B)
D(A<=B) = (A = B)
D(A>=B) = (A = B)
D(A AND B) = D(A) AND D(B)
D(A OR B) = D(A) OR D(B)
D(NOT A) = (A = 0)
D(IF A THEN B ELSE C) = IF A THEN D(B) ELSE D(C)
D(LOOKUP(M,A)) = D(A)*[LOOKUP(M,1.01*A+.000001)-LOOKUP(M,.99*A-.000001)]/
                     (.02*A+.000002)
```

Various built-in functions which operate on symbolic expressions and return scalars are differentiable. The rules for differentiating these functions are given below.

```
D(SUM(I,A,B,E)) = SUM(I,A,B,D(E))
D(PRODUCT(I,A,B,E))=SUM(I,A,B,D(E)*PRODUCT(J,A,B,IF I=J THEN 1 ELSE E(J))
D(INTEGRAL(X,A,B,E(X))) = E(B)*D(B) - E(A)*D(A)+INTEGRAL(X,A,B,D(E))
D(ROOT(X,A,B,E(X))) = EVAL(X,ROOT(X,A,B,E(X)),-D(E)/Dx(E))
D(EVAL(X,A,E)) = EVAL(X,A,D(E))+D(A)*EVAL(X,A,Dx(E))
```

When user-defined functions are to be differentiated, the chain rule is used. If F is defined as: F(X,Y,...) = E, then D(F(A,B...)) = F DIFF X(A,B...)*D(A)+F DIFF Y(A,B...)*D(B)+....

Most of the scalar-valued builtin functions may be differentiated symbolically. The complex-valued functions with the same names may not be differentiated. The following table lists such functions and their symbolic derivatives.

For the functions ABS, SIGN, CEILING, FLOOR, MAX, MIN, and MOD, the "correct" derivative involves Dirac delta-functions (d- functions), so the following definitions are somewhat arbitrary.

```
D[ABS(X)] = D(X)*SIGN(X)
D[SIGN(X)] = D(X)*(X=0)
D[CEILING(X)] = D(X)*(X=CEILING(X))
D[FLOOR(X)] = D(X)*(X = FLOOR(X))
```

```
D[MAX(X1,...,XN)] = IF MAX(X2,...,XN) < X1 THEN D(X1) ELSE D[MAX(X2,...,XN)]
D[MIN(X1,...,XN)] = IF MIN(X2,...,XN) < X1 THEN D(X1) ELSE D[MIN(X2,...,XN)]
D[MOD(X,Y)] = D(X)*FLOOR(X/Y)=(X/Y)*Y-D(Y)*FLOOR(X/Y)
D[SQRT(X)] = D(X)/(2*SQRT(X))
D[SIN(X)] = D(X)*COS(X)
D[SIND(X)] = D(X)*(PI/180)*COSD(X)
D[COS(X)] = -D(X)*SIN(X)
D[COSD(X)] = -D(X)*(PI/180)*SIND(X)
D[TAN(X)] = D(X)/COS(X)^2
D[TAND(X)] = D(X)*(PI/180)/COSD(X)^2
D[COSEC(X)] = -D(X)*COSEC(X)*COTAN(X)
D[COSECD(X)] = -D(X)*(PI/180)COSEC(X)*COTAN(X)
D[SEC(X)] = D(X)*SEC(X)*TAN(X)
D[SECD(X)] = D(X)*SEC(X)*(PI/180)*TAN(X)
D[COTAN(X)] = -D(X)/SIN(X)^2/SIN(X)^2
D[COTAND(X)] = -D(X)/SIN(X)^2*(PI/180)/SIN(X)^2
D[ASIN(X)] = D(X)/SQRT(1-X*X)
D[ACOS(X)] = -D(X)/SQRT(1-X*X)
D[ATAN(X)] = D(X)/(1+X*X)
D[ATAN2(Y,X)] = D(Y)*X/(X*X+Y*Y)-D(X)*Y/(X*X+Y*Y)
D[ACOTAN(X)] = -D(X)/(1+X*X)
D[ASEC(X)] = D(X)/(X*SQRT(X*X-1))
D[ACOSEC(X)] = -D(X)/(X*SQRT(X*X-1))
D[EXP(X)] = D(X)*EXP(X)
D[LOG(X,Y)] = D(X)/(X*LOG(Y))-D(Y)*LOG(X)/(Y*LOG(Y)^2)
D[LOG2(X)] = D(X)*(1/LOG(2))/X = D(X)*(0.693147181.../X)
D[LOG10(X)] = D(X)*(1/LOG(10))/X= D(X)*(0.434294482.../X)
D[SINH(X)] = D(X)*COSH(X)
D[COSH(X)] = D(X)*SINH(X)
D[TANH(X)] = D(X)/COSH(X)^2
D[ASINH(X)] = D(X)/SQRT(X*X+1)
D[ACOSH(X)] = D(X)/SQRT(X*X-1)
D[ATANH(X)] = D(X)/(1-X*X)
D[ACTNH(X)] = D(X)/(1-X*X)
D[ASECH(X)] = -D(X)/(X*SQRT(1-X*X))
D[ACSCH(X)] = -1/(X*SQRT(1-X*X))
D[GAMMA(X)] = D(X)*DIGAMMA(X)*GAMMA(X)
D[LOGGAMMA(X)] = D(X)*DIGAMMA(X)
D[ERF(X)] = 2*D(X)/SQRT(PI)*EXP(-X*X)
D[ERFC(X)] = -2*D(X)/SQRT(PI)EXP(-X*X)
D[GAUSSF(X,Y,Z)] = GAUSSD(X,Y,Z)*[D(X)-D(Y)-D(Z)*(X-Y)/(2*Z)]
D[GAUSSD(X,Y,Z)] =GAUSSD(X,Y,Z)* [(X-Y)*(D(Y)-D(X))+((X-Y)^2/(2*Z)-1/2)*D(Z)]
D[GAUSSI(X,Y,Z)] = D(X)/GAUSSD(GAUSSI(X,Y,Z),Y,Z)+D(Y)+D(Z)*(GAUSSI(X,Y,Z)-Y)/(2*Z)
D[CHISQF(X,Y,0)]  = (1/(2*GAMMA(Y/2))*{D(X)*(X/2)^(Y/2-1)*EXP(-X/2)]+
   D(Y)*[JGAMMA(X/2,Y/2)-CHISQF(X,Y,0)*JGAMMA(INFINITY,Y/2)]}
D(CHISQF(X,Y,Z) = D(Z)*(CHISQF(X,Y+2,Z)-CHISQF(X,Y,Z))/2+
   EXP(-Z/2)*SUM(K,0,INFINITY,(Z/2)^K*D(CHISQF(X,Y+2*K,0))/FACT(K))
D[CHISQD(X,Y,0)] = CHISQD(X,Y,0)*{D(X)*(1/X-1/2)+
   D(Y)*[LOG(X)-LOG(2)/2-JGAMMA(INFINITY,Y/2)/(2*GAMMA(Y/2)]}
D1[STUTF(X,Y,Z)] = D(X)*STUTD(X,Y,Z)
D1[QFF(X,Y,Z,W)] = D(X)*QFD(X,Y,Z,W)
D1[BESSJ(X,Y)] = IF Y = 0 THEN -D(X)*BESSJ(X,1) ELSE
   D(X)*(BESSJ(X,Y-1)-BESSJ(X,Y+1))/2
D1[BESSY(X,Y)] = IF Y = 0 THEN -D(X)*BESSY(X,1) ELSE
```

```
   D(X)*(BESSY(X,Y-1)-BESSY(X,Y+1))/2
D1[BESSI(X,Y)] =  IF Y = 0 THEN D(X)*BESSI(X,1) ELSE
   D(X)*(BESSI(X,Y-1)+BESSI(X,Y+1))/2
D1[BESSK(X,Y)] = IF Y = 0 THEN -D(X)*BESSK(X,1) ELSE
   -D(X)*(BESSK(X,Y+1)+ BESSK(X,Y-1))/2
D1[LEG(X,Y)] = D(X)*(Y*(Y+1)LEG(X,Y-1)/SQRT(X^2-1))
 D1[ALEG(X,Y,Z)] = D(X)*((Y+Z)*(Y-Z+1)*ALEG(X,Y,Z-1)/SQRT(X*X-1)-Z*X*ALEG(X,Y,Z))
D1[EL2(X,Y,Z,W)] = D(X)*((Z+W*X*X)/SQRT(1+Y*Y*X*X)-Z)/((1+X*X)*SQRT(1+X*X)
D1[LE2(X,Y)] = D(X)*(SQRT(1+Y^2*SIN(X)^2)-1)
D1[.IJACOBISN(X,Y)] = D(X)*JACOBICN(X,Y)*JACOBIDN(X,Y)
D1[JACOBICN(X,Y)] = -D(X)*JACOBISN(X,Y)*JACOBIDN(X,Y)
D1[JACOBIDN(X,Y)] = D(X)*Y*JACOBISN(X,Y)*JACOBICN(X,Y)
D1[IBETA(X,A,B)] = (D(X)*X^(A-1)*(1-X)^(B-1) -
IBETA(X,A,B)*D(BETA(A,B)) +
  (D(A)*JBETA(X,A,B)+D(B)*(JBETA(1,B,A)-JBETA(1- X,B,A)))/BETA(A,B)
D[IGAMMA(X,Y)]  =  D(Y)*Y^(X-1)*EXP(-Y)-D(GAMMA(X))*IGAMMA(X,Y)
+  D(X)*JGAMMA(Y,X)
D1[EXPINTN(X,Y)] = -D(X)*EXPINTN(X,Y-1)
D[EXPINT(X)] = -D(X)*EXP(-X)/X
D[SININT(X)] = D(X)*SIN(X)/X
D[COSINT(X)] = D(X)*(-2+COS(X))/X
D[TYPEOUT(X)] = TYPEOUT(D(X))
D[BETA(X,Y)] = D(X)*JBETA(1,X,Y)+D(Y)*JBETA(1,Y,X)
D[BETAF(X,P,Q,0,A,B)] = 1/BETA(P,Q)*
   {(X-A)^(P-1)*(B-X)^(Q-1)*[(B-A)*D(X)+(X-B)*D(A)-(X-A)*D(B)]/(B-A)^(P+Q) -
   BETAF(X,P,Q,0,A,B)*D(BETA(P,Q))/BETA(P,Q) +
   D(P)*JBETA((X-A)/(B-A),P,Q)+D(Q)*(JBETA(1,Q,P)-JBETA((B-X)/(B-A),Q,P))}
D[BETAF(X,P,Q,C,A,B)] = D(C)*[BETAF(X,P+1,Q,C,A,B)-
BETAF(X,P,Q,C,A,B)]/2+
   EXP(-C/2)*SUM[K,0,INFINITY,(C/2)^K*D(BETAF(X,P+K,Q,0,A,B))/K!)
D[BETAD(X,P,Q,0,A,B)] = BETAD(X,P,Q,C,A,B)*
   {[(P-1)/(X-A)-(Q-1)/(B-X)]*D(X)+[(P+Q-1)/(B-A)-(P-1)/(X-A)]*D(A)+
   [(Q-1)/(B-X)-(P+Q+1)/(B-A)]*D(B)+[LOG(X-A)-LOG(B-A)]*D(P)+
   [LOG(B-X)-LOG(B-A)]*D(Q)+D(BETA(P,Q)/BETA(P,Q)}
D[BETAD(X,P,Q,C,A,B)] = 0.5*[BETAD(X,P+1,Q,C,A,B)-BETAD(X,P,Q,C,A,B)]*D(C)
   +EXP(-C/2)*SUM(K,0,INFINITY,(C/2)^K*D[BETAD(X,P+K,Q,0,A,B)]/K!)
D[GAMMAF(X,P,A,B)] = IF X<B THEN 0 ELSE 1/GAMMA(P)*{[D(X)-D(B)-((X-A)/B)*D(A)]*
   EXP(-(X-A)/B)/A+D(P)*[JGAMMA((X-A)/B,P)-GAMMAF(X,P,A,B)*JGAMMA(INFINITY,P)]}
D[GAMMAD(X,P,A,B)] = IF X<B THEN 0 ELSE GAMMAD(X,P,A,B)*{D((X-A)/B)*[(P-1)/((X-A)/B)-1]-
   D(A)/A+D(P)*LOG((X-A)/B)-D(GAMMA(P))/GAMMA(P)}
```

7. Planes, Windows, Frames and Images

All curves, titles, and axes which are to be shown on the display and/or eventually plotted must be drawn in a Cartesian *plane*. Many such planes may be in use simultaneously.

Within each plane there is exactly one rectangular region or *window* which may be displayed. Curves may be drawn anywhere in a plane but only those parts within the specified rectangular window of the plane can be seen.

Each such plane also has exactly one associated frame region and image region on the display screen, with the image region nested within the frame region. Thus the basic elements for 2D-graphics in MLAB are plane-window-frame-image quadruples.

The *image* of a window is a rectangular area on the display in which the content of the window is seen. The image of a window is established or changed with the IMAGE statement. The size and location of an image may be changed at any time. The contents of a window as seen in its image are always scaled (stretched or compressed) linearly with a suitable affine mapping so as to "fit" the window rectangle into the image rectangle. This automatic application of an affine transformation can often be used to advantage.

The *frame* of a window is a rectangular region on the display screen It contains the image region of the window. It may also contain various axis and title graphics objects. The frame of a window is established or changed with the FRAME statement. Various frames may overlap arbitrarily on the display screen. The image of a window lies wholly within its corresponding frame.

One establishes a plane and its window (or changes the window of an established plane) by using the WINDOW statement. A name may be given to a plane-window frame-image quadruple in a WINDOW statement (say, W1), and that name is used henceforth to specify the window, the frame, the image or the plane.

The size and location of a window within a plane may be changed at any time. Each window, say W, always has associated with it its single "image" region and single "frame" region, both of which which are also called W.

The AXIS statement is used to install a curve which may serve as an axis and to position numeric labels associated with points on the axis-curve in the frame, which may of course overlay the image.

The XAXIS statement and YAXIS statement are used to construct axes in which not only does the axis position remain fixed within the frame, but also the axis numbers remain in fixed relation to the bounds of the underlying window.

Thus, one may imagine a frame as a rectangular plate of glass upon which things may be inscribed, and through which one may see both axes-curves and the contents of the window underneath. Such plates of glass are are laid on the display screen to form the final composite picture to be seen.

In the case of a windowing system such as X-windows or NeXTstep, the screen is merely a window managed via the system window manager. In this case, the screen may be resized, and the frame and image of every existing MLAB window will be linearly scaled to occupy the same relative positions in the "new" (resized) screen that they occupied in the earlier (unresized) screen.

8. Curves, Axes, and Titles

A *curve* in MLAB is an ordered set of one or more points in the plane, which may be displayed individually using various point symbols, and optionally connected by lines of various sorts. The points of a curve are always stored and manipulated by the user as a 2-column matrix whose rows are the points (x_i, y_i), of the curve. The ordering of the points is the ordering of the rows in the curve matrix. In addition to the point set, a curve consists of a line type specification, a point type specification, a point size specification, (which may be a matrix), a color triple specification, and a numeric label matrix, with an associated font, size, angle, offset and placement specification.

A curve belongs to a window. It is drawn in the plane of a plane- window-frame-image quadruple by using the DRAW statement. The points may be labeled by numeric values.

A curve has a name which is assigned in the DRAW statement creating it. The curve name may be used in blanking, unblanking, redrawing, or deleting the curve.

A *title* in MLAB is a text string drawn in the plane of a window. In addition to the text string, a title has a size (with optional units), a font, a color, a position (with optional units), an angle, a centering value, and a placement pair. Within a string, sizes, fonts, colors, and angles may be locally altered.

An *axis* in MLAB is a graphics data object with the same properties as a curve. An axis may be associated with a specified window with the AXIS statement. An axis is really just a curve, which, however, may be imagined to reside in the frame rather than the window. Thus, when the window is changed, the axis does not move. Two special kinds of axes are the XAXIS axis and the YAXIS axis. An XAXIS axis is an axis whose point labels are readjusted whenever the window x-limits are changed. A YAXIS axis is an axis whose point labels are readjusted whenever the window y-limits are changed. The effect is that xaxis and yaxis axis numbers remain correct when the window is changed.

The TITLE statement is used to place a string of text, a title, which may be written anywhere in the frame. These strings may be used to label axes or to provide notes or comments. They may consist of several different fonts, with multiple lines, subscripts, superscripts, colors, and varying-size characters.

The location of a title may be given in screen, frame, or window coordinates. However, a title which is located outside the window is still displayed unless it would be off the frame. In this case, any portion of the string within the frame is displayed.

The most frequently used titles occur at the top, left or right side, and bottom of a picture. These titles are conveniently created using the TOP, LEFT, RIGHT, and BOTTOM qualifiers to the TITLE statement.

9. 3D-Spaces, Images, and Surfaces

Graphs of perspective projections of 3-dimensional objects can be produced by MLAB. To do this in MLAB one establishes a euclidean 3-dimensional space with an XYZ cartesian coordinate system and then draws a user-specified set of points in this 3D-space which may be connected by lines in various elaborate ways. Such graphs in 3D-space are called "surfaces" since this is generally what is desired, however an MLAB surface can be an arbitrary collection of points in 3D-space, including space curves for example. Once one or more surfaces have been drawn, they may be rotated, scaled, or translated and then viewed as if photographed by a camera in space. The imaginary camera can be positioned at will and pointed in any direction, and moreover the viewing angle can be set as desired.

A 3D-space is created by specifying a name for the space to be used in drawing a surface defined by a 3-column matrix with the 3D-DRAW statement (the 3D-DRAW statement is a special 3D form of the ordinary DRAW statement). If no such name is given, the name W3 is assumed.

Each 3D-space has an associated camera that can be controlled by 3D-VIEW CONTROL statements. The patch of screen in which all scenes are displayed is set and reset by the IMAGE and FRAME statements, just as with 2D-windows.

The clipping box of a 3D-space is a rectilinearly-oriented region such that lines or points drawn outside the clipping box will not be drawn. Unless explicitly specified with a WINDOW3D statement, the clipping box of a 3D-space is taken to be the entire space.

Thus a 3D-window is a named 3D-space with an embedded cartesian coordinate system together with a clipping box and a fictitious camera which views some four-sided pyramidal region of the coordinatized 3D-space called the viewing pyramid, and a rectangular image patch in which the camera view is seen.

The coordinate systems for 3D-windows are left-handed. If a right-handed coordinate system is desired, it suffices to scale Z by -1 (by executing [SPACE ZSCALE -1]) in the desired 3D-window.

A surface is a finite collection of zero or more points in 3D- space which are drawn and viewed in some particular 3D-window. Various line and/or point symbols may be used in graphing a surface. The lines and points which are drawn are restricted to those which lie inside the clipping box of the 3D-window when the surface is drawn. A surface object may in fact represent a space- curve or an arbitrary collection of points as well as a true surface.

There may be many surfaces in a single 3D-window, and there may be many 3D-windows in existence simultaneously.

A surface is established and drawn in a 3D-window by drawing a 3-column matrix with a 3D DRAW statement, which is the 3D form of the DRAW statement. This statement will also create a 3D-window if the window mentioned in the 3D DRAW statement does not currently exist. A surface may be redrawn by drawing it again; various parameters of a surface, including which 3D-window it is placed in, may be changed.

For each 3D-window, a 2D-perspective projection line drawing of the visible parts of the clipped surfaces defined by various 3-column matrices is displayed. The combined "object" (set of surfaces) is shown as though it were the result of a camera which recorded the scene from some point in space. The camera may be controlled by various 3D-VIEW CONTROL statements. The result is that the scene changes, thus reflecting the new state of the camera.

10. Built-In Operators and Functions

A large variety of builtin operators and functions are provided in MLAB. Some of these operations are defined on both scalars and matrices, or combinations thereof. The combinations are handled by the implicit conversion conventions which follow.

If a matrix argument is given where a scalar is required, the [1,1] element of the matrix is used in some cases where this seems natural. When a matrix is converted into a scalar by taking the [1,1] element, we say that the matrix is coerced to a scalar. In certain other cases coercion does not take place, and an error message will result.

In some cases, if a scalar argument is given where a matrix is required, the argument used is a 1×1 matrix whose [1,1] element is equal to the scalar argument value. When a scalar is converted into a 1×1 matrix, we say that the scalar is coerced to a matrix. In other cases coercion does not take place, and an error message will result.

Certain matrix operations require their arguments to be conformable, (i.e. to have certain size relationships). If a matrix argument A has too few rows or columns, then a copy of A with the appropriate number of rows and/or columns added is usually used in place of the original argument. The extra added rows are taken cyclicly from A, starting with row 1, and the extra added columns are taken cyclicly from A, starting with column 1. This is referred to as cyclic extension. Exceptions to this rule are stated below.

Note, operators and functions are used to compose various values into expressions. An expression is just a complex name for a value; it is not a statement. Expressions, of course, occur in statements. Thus, M = READ(FOO) is a statement, but READ(FOO) is an expression. Generally, only statements can be typed into MLAB. The one exception to this rule involves the type statement. The symbol TYPE may be omitted; nevertheless, the specified expression is "executed" as if it were a statement beginning with the symbol TYPE. Thus "1+2" as a statement is the same as "TYPE 1+2".

A list of the builtin infix form and functional form operators in MLAB follows.

MLAB consists of the base MLAB system together with twelve optionally-included *extension packages*. These extension packages are:

1. ordinary-differential equation (ODE) solving
2. curve-fitting
3. optimization
4. advanced linear algebra
5. special functions
6. signal analysis functions
7. cluster analysis functions
8. density functions, distribution functions, and inverse distribution functions
9. statistical computations and tests
10. spline smoothing
11. random number equations
12. 3D graphics and contour maps

The functions throughout this index are tagged with the number (1-12) of the extension package to which they belong. The functions of the base MLAB system are tagged with the value 0.

1. Basic Scalar and Matrix Arithmetic Operators

1·1	+,–	unary and binary addition and subtraction
1·2	*	multiplication
1·3	/	division
1·4	^	exponentiation

2. Relational Operators

2·1	<, <=, =, >=, >, NOT=	relational (comparison) operators

3. Boolean Operators

3·1	NOT, AND, OR	logical negation, and, or

4. Conditional Computation Operator

4·1	IF-THEN-ELSE	conditional operator

5. Matrix Manipulation Operators

5·1	:	sequence generator
5·2	::	fixed step sequence generator
5·3	:!	fixed-size sequence generator
5·4	[ ]	matrix subscripting
5·5	ROW	row submatrix construction
5·6	COL	column submatrix construction
5·7	’	transpose of a matrix
5·8	&	row-wise concatenation
5·9	&’	column-wise concatenation
5·10	^^	row-wise replication
5·11	^^’	column-wise replication
5·12	#	vector cross product
5·13	ON	function application operator
5·14	*’	element-by-element multiplication
5·15	/’	element-by-element division

6. String Operators

6·1	+	string concatenation
6·2	^^	string replication
6·3	<, <=, =, >=, >	relational (comparison) string operators

7. Integer Arithmetic Functions and Combinatorial Functions

7·1	SIGN^0	sign function
7·2	INT^0	integer part function
7·3	CEILING^0	ceiling function
7·4	FLOOR^0	floor function
7·5	ROUND^0	round to nearest integer
7·6	MOD^0	modulus (integer division remainder) function
7·7	BINCOEF^5	binomial coefficient
7·8	BERNUM^5	Bernoulli number

7·9	EULNUM[5]	Euler number
7·10	STIRL1[5]	Stirling number of the first kind
7·11	STIRL2[5]	Stirling number of the second kind
7·12	PARTU[5]	unrestricted partitions of an integer
7·13	PARTD[5]	distinct partitions of an integer
7·14	GCD[5]	greatest common divisor
7·15	EGCD[0]	extended greatest common divisor
7·16	FACTOR[0]	prime factors of an integer
7·17	PRIMES[0]	list of primes up to argument
7·18	NPRIME[5]	the X^{th} prime number
7·19	DIVISORS[0]	divisors of an integer
7·20	CATNUM	Catalan number
7·21	PARITY	parity of a permutation
7·22	BERPOL[5]	Bernoulli polynomial
7·23	EULPOL[5]	Euler polynomial
7·24	FACT[5]	factorial function
7·25	RFACT[5]	rising factorial function

8. Elementary Functions

8·1	ABS[0]	absolute value function
8·2	SQRT[0]	square root function
8·3	EXP[0]	exponential function
8·4	LOG[0]	logarithm function
8·5	LN[0]	natural (base e) logarithm function
8·6	LOG10[5]	base-10 logarithm function
8·7	LOG2[5]	base-2 logarithm function
8·8	MIN[0]	minimum of list of scalars
8·9	MAX[0]	maximum of list of scalars

9. Non-Elementary Functions

9·1	GAMMA[0]	Gamma function
9·2	LOGGAMMA[0]	logarithmic gamma function
9·3	DIGAMMA[0]	derivative of the logarithmic Gamma function
9·4	IGAMMA[5]	incomplete Gamma function
9·5	BETA[5]	Beta function
9·6	IBETA[5]	incomplete beta function
9·7	ERF[5]	error function
9·8	ERFC[5]	complementary error function
9·9	EXPINTN[5]	Yth order exponential integral function
9·10	SININT[5]	sine integral function
9·11	COSINT[5]	cosine integral function
9·12	EXPINT[5]	exponential integral function
9·13	LOGINT[5]	logarithmic integral function
9·14	SINHINT[5]	hyperbolic sine integral function
9·15	COSHINT[5]	hyperbolic cosine integral function
9·16	JBETA[5]	function used in derivative of beta distribution function
9·17	JGAMMA[5]	function used in derivative of gamma distribution function
9·18	SPENCE[5]	Spence's dilogarithm
9·19	DAWSON[5]	Dawson's integral

10. Trigonometric Functions

10·1	SIN[0]	trigonometric sine function
10·2	SIND[5]	trigonometric sine function (argument in degrees)
10·3	COS[0]	trigonometric cosine function
10·4	COSD[5]	trigonometric cosine function (argument in degrees)
10·5	TAN[0]	trigonometric tangent function
10·6	TAND[5]	trigonometric tangent function (argument in degrees)
10·7	SEC[0]	trigonometric secant function
10·8	SECD*	trigonometric secant function (argument in degrees)
10·9	COSEC[5]	trigonometric cosecant function
10·10	COSECD*	trigonometric cosecant function (argument in degrees)
10·11	COTAN[5]	trigonometric cotangent function
10·12	COTAND*	trigonometric cotangent function

11. Inverse Trigonometric Functions

11·1	ASIN[0]	inverse trigonometric sine function
11·2	ASIND[0]	inverse trigonometric sine function (result in degrees)
11·3	ACOS[0]	inverse trigonometric cosine function
11·4	ACOSD[0]	inverse trigonometric cosine function (result in degrees)
11·5	ATAN[0]	inverse trigonometric tangent function
11·6	ATAND[0]	inverse trigonometric tangent function (result in degrees)
11·7	ATAN2[0]	2-argument inverse trigonometric tangent function
11·8	ATAN2D[0]	2-argument inverse trigonometric tangent function (result in degrees)
11·9	ACOTAN[5]	inverse trigonometric tangent function
11·10	ASEC[5]	inverse trigonometric secant function
11·11	ACOSEC[5]	inverse trigonometric secant function

12. Hyperbolic Functions

12·1	SINH[5]	hyperbolic sine function
12·2	COSH[5]	hyperbolic cosine function
12·3	TANH[5]	hyperbolic tangent function
12·4	COSECH[5]	hyperbolic cosecant function
12·5	SECH[5]	hyperbolic secant function
12·6	COTANH[5]	hyperbolic cotangent function

13. Inverse Hyperbolic Functions

13·1	ASINH[5]	inverse hyperbolic sine function
13·2	ACOSH[5]	inverse hyperbolic cosine function
13·3	ATANH[5]	inverse hyperbolic tangent function
13·4	ACOTANH[5]	inverse hyperbolic cotangent function
13·5	ASECH[5]	inverse hyperbolic secant function
13·6	ACOSECH[5]	inverse hyperbolic cosecant function

14. Bessel Functions

14·1	BESSJ[5]	Bessel function of the first kind
14·2	BESSY[5]	Bessel function of the second kind
14·3	BESSI[5]	modified Bessel function of the first kind
14·4	BESSK[5]	modified Bessel function of the second kind
14·5	AIRY[5]	Airy function Ai
14·6	BAIRY[5]	Airy Function Bi
14·7	AIRYD[5]	Airy Function Ai derivative

14·8	BAIRYD[5]	Airy Function Bi Derivative

15. Legendre Functions

15·1	ALEG[5]	associated Legendre function
15·2	LEG[5]	Legendre function

16. Elliptic Functions, Elliptic Integrals, and Theta Functions

16·1	EL2[5]	general elliptic integral of the second kind
16·2	CEL[5]	general complete elliptic integral
16·3	LE1[5]	Legendre elliptic integral of the first kind
16·4	LE2[5]	Legendre elliptic integral of the second kind
16·5	CE1[5]	complete elliptic integral of the first kind
16·6	CE2[5]	complete elliptic integral of the second kind
16·7	CE3[5]	complete elliptic integral of the third kind
16·8	JACOBIAM[5]	Jacobian am elliptic function
16·9	JACOBISN[5]	Jacobian sn elliptic function
16·10	JACOBICN[5]	Jacobian cn elliptic function
16·11	JACOBIDN[5]	Jacobian dn elliptic function
16·12	JACOBICD[5]	Jacobian cd elliptic function
16·13	JACOBIDC[5]	Jacobian dc elliptic function
16·14	JACOBINS[5]	Jacobian ns elliptic function
16·15	JACOBISD[5]	Jacobian sd elliptic function
16·16	JACOBINC[5]	Jacobian nc elliptic function
16·17	JACOBIDS[5]	Jacobian ds elliptic function
16·18	JACOBIND[5]	Jacobian nd elliptic function
16·19	JACOBISC[5]	Jacobian sc elliptic function
16·20	JACOBICS[5]	Jacobian cs elliptic function
16·21	THETA1[5]	Jacobi's first theta function
16·22	THETA2[5]	Jacobi's second theta function
16·23	THETA3[5]	Jacobi's third theta function
16·24	THETA4[5]	Jacobi's fourth theta function
16·25	THETAS[5]	Neville's s theta function
16·26	THETAC[5]	Neville's c theta function
16·27	THETAD[5]	Neville's d theta function
16·28	THETAN[5]	Neville's n theta function
16·29	ELLRC[5]	degenerate elliptic integral
16·30	ELLRD[5]	elliptic integral of the second kind
16·31	ELLRF[5]	elliptic integral of the first kind
16·32	ELLRJ[5]	elliptic integral of the third kind

17. Statistical Distribution Functions

17·1	GAUSSF[8]	normal distribution function
17·2	CHISQF[8]	non-central chi-squared distribution function
17·3	STUTF[8]	non-central Student's t distribution function
17·4	ASTUTF[8]	absolute non-central Student's t distribution function
17·5	QFF[8]	singly non-central F-distribution function
17·6	KSMF[8]	Kolmogorov-Smirnov distribution function
17·7	BINOMF[8]	binomial distribution function
17·8	POISSF[8]	Poisson distribution function
17·9	SPEARF[8]	Spearman rank-correlation distribution function
17·10	WILCF[8]	Wilcoxon signed-rank distribution function

17·11	MWF[8]	Mann-Whitney-Wilcoxon rank-sum distribution function
17·12	GAMMAF[8]	gamma distribution function
17·13	BETAF[8]	non-central beta distribution function
17·14	WEIBF[8]	Weibull distribution function
17·15	NBINF[8]	negative binomial distribution function
17·16	EXPF[8]	exponential distribution function
17·17	GEOMF[8]	geometric distribution function
17·18	HGEOMF[8]	hypergeometric distribution function
17·19	LOGISF[8]	logistic distribution function
17·20	CAUCHYF[8]	Cauchy distribution function
17·21	LAPLACEF[8]	Laplace distribution function
17·22	PARETOF[8]	Pareto distribution function
17·23	UNIF[8]	uniform distribution function
17·24	TRIF[8]	triangular distribution function
17·25	LNORMF[8]	lognormal distribution function
17·26	GUMBELF[8]	Gumbel extreme value distribution function
17·27	HNORMF[8]	halfnormal distribution function

18. Inverse Statistical Distribution Functions

18·1	GAUSSI[8]	inverse gaussian distribution function
18·2	CHISQI[8]	inverse non-central chi-squared distribution function
18·3	STUTI[8]	inverse non-central Student's t distribution function
18·4	ASTUTI[8]	inverse non-central absolute Student's t dist. function
18·5	QFI[8]	inverse singly non-central F-distribution function
18·6	BINOMI[8]	inverse binomial distribution function
18·7	POISSI[8]	inverse Poisson distribution function
18·8	GAMMAI[8]	inverse Gamma distribution function
18·9	BETAI[8]	inverse non-central Beta distribution function
18·10	WEIBI[8]	inverse Weibull distribution function
18·11	NBINI[8]	inverse negative binomial distribution function
18·12	EXPI[8]	inverse exponential distribution function
18·13	GEOMI[8]	inverse geometric distribution function
18·14	HGEOMI[8]	inverse hypergeometric distribution function
18·15	LOGISI[8]	inverse logistic distribution function
18·16	CAUCHYI[8]	inverse Cauchy distribution function
18·17	LAPLACEI[8]	inverse Laplace distribution function
18·18	PARETOI[8]	inverse Pareto distribution function
18·19	UNII[8]	inverse uniform distribution function
18·20	TRII[8]	inverse triangular distribution function
18·21	LNORMI[8]	inverse lognormal distribution function
18·22	GUMBELI[8]	inverse Gumbel distribution function
18·23	HNORMI[8]	inverse halfnormal distribution function

19. Probability Density Functions

19·1	GAUSSD[8]	normal density function
19·2	CHISQD[8]	non-central chi-squared density function
19·3	STUTD[8]	non-central Student's t density function
19·4	ASTUTD[8]	absolute non-central Student's t density
19·5	QFD[8]	singly non-central F density function
19·6	BINOMD[8]	binomial density function
19·7	POISSD[8]	Poisson density function
19·8	SPEARD[8]	Spearman rank-correlation test density function

19·9	WILCD[8]	Wilcoxon signed-rank test density function
19·10	MWD[8]	Mann-Whitney-Wilcoxon rank-sum test density function
19·11	BETAD[8]	non-central beta density function
19·12	GAMMAD[8]	gamma density function
19·13	WEIBD[8]	Weibull density function
19·14	NBIND[8]	negative binomial distribution density function
19·15	EXPD[8]	exponential density function
19·16	GEOMD[8]	geometric density function
19·17	HGEOMD[8]	hypergeometric distribution density function
19·18	LOGISD[8]	logistic density function
19·19	CAUCHYD[8]	Cauchy density function
19·20	LAPLACED[8]	Laplace density function
19·21	PARETOD[8]	Pareto density function
19·22	UNID[8]	uniform density function
19·23	TRID[8]	triangular density function
19·24	LNORMD[8]	lognormal distribution density function
19·25	GUMBELD[8]	Gumbel (extreme value) density function
19·26	LOGSERD[8]	logarithmic series distribution density function
19·27	HNORMD[8]	halfnormal density function

20. Random Number Generators

20·1	RAN[11]	uniform random number
20·2	NORMRAN[11]	normal random number
20·3	POISRAN[11]	Poisson-distributed random number
20·4	BINRAN[11]	binomial-distributed random number
20·5	CAUCHYRAN[11]	Cauchy-distributed random number
20·6	GEOMRAN[11]	geometric-distributed random number
20·7	GUMBELRAN[11]	Gumbel (extreme value) random number
20·8	LOGISRAN[11]	logistic-distributed random number
20·9	LNORMRAN[11]	lognormal-distributed random number
20·10	NBINRAN[11]	negative binomial-distributed random number
20·11	PARETORAN[11]	Pareto-distributed random number
20·12	TRIRAN[11]	triangular-distributed random number
20·13	WEIBRAN[11]	Weibull-distributed random number
20·14	HNORMRAN[11]	half normal-distributed random number
20·15	EXPRAN[11]	exponential-distributed random number
20·16	GAMRAN[11]	gamma-distributed random number
20·17	BETARAN[11]	non-central beta-distributed random number
20·18	CHISQRAN[11]	non-central chi squared-distributed random number
20·19	FRAN[11]	singly non-central F-distributed random number
20·20	STUTRAN[11]	Student't t-distributed random number
20·21	IRAN[11]	integer-valued random number
20·22	RANPERM[11]	random permutation of random combination

21. Statistical Computations

21·1	MEAN[0]	trimmed arithmetic mean
21·2	MEDIAN[9]	median
21·3	MODE[9]	mode
21·4	STDDEV[0]	standard deviation
21·5	VAR[0]	variance
21·6	AVDEV[9]	average absolute deviation
21·7	SKEW[9]	skew

21·8	KURT[9]	kurtosis
21·9	COV[9]	covariance matrix
21·10	CORR[9]	correlation matrix
21·11	CDF[9]	empirical cumulative distribution function
21·12	HISTO[9]	compute a histogram matrix
21·13	RANKORDER[9]	rank ordering
21·14	EWT[2]	estimated weights for curve-fitting
21·15	RMEAN*	rank-based robust mean estimators
21·16	RDEV*	rank-based robust standard deviation estimators
21·17	GMEAN*	geometric mean
21·18	HMEAN*	harmonic mean
21·19	DMEAN*	discrete functional mean
21·20	FMEAN*	functional mean
21·21	MOMENT*	sample moment
21·22	ENTROPY[9]	entropy of a probability density function
21·23	JENTROPY[9]	entropy of a contingency table
21·24	CENTROPY[9]	conditional entropy of a contingency table
21·25	UNCERT[9]	uncertainty of a contingency table
21·26	JUNCERT[9]	joint uncertainty of a contingency table

22. Statistical Tests

22·1	TMT[9]	Student's t test against a postulated constant mean
22·2	TST[9]	Student's t test for equal means (equal variances)
22·3	TDT[9]	Student's t test for equal means (unequal variances)
22·4	TPT[9]	Student's t test for equal means (paired samples)
22·5	QFT[9]	F-test for equal variances
22·6	CHISQ1T[9]	chi-square test on data versus expected values
22·7	CHISQFT[9]	chi-square test on histogram vs. cdf
22·8	CHISQ2T[9]	chi-square test on two data sets
22·9	CTABT[9]	chi-square test on contingency table
22·10	KSF1T[9]	Kolmogorov-Smirnov 1-sample test on a cdf
22·11	KSF2T[9]	Kolmogorov-Smirnov 2-sample test on 2 cdf's
22·12	KS1T[9]	Kolmogorov-Smirnov 1-sample test
22·13	KS2T[9]	Kolmogorov-Smirnov 2-sample test
22·14	PEART[9]	Pearson correlation test on bivariate normal data
22·15	CORR2T[9]	correlation-coefficient test
22·16	SPEART[9]	Spearman rank-correlation test on two data sets
22·17	WIL1T[9]	Wilcoxon 1-sample signed-rank test
22·18	WIL2T[9]	Wilcoxon 2-sample paired data signed-rank test
22·19	MWT[9]	Mann-Whitney-Wilcoxon rank-sum test
22·20	KEN1T[9]	Kendall's paired-sample tau correlation coefficient test
22·21	KEN2T[9]	Kendall's tau test on a contingency table
22·22	SKEWT[9]	D'Agostino's skewness test for normality
22·23	KURTT[9]	D'Agostino's kurtosis test for normality
22·24	NORMT[9]	D'Agostino's omnibus test for normality
22·25	KWT[9]	Kruskal-Wallis multi-sample rank-sum test
22·26	LEVENET[9]	Levene's test for equality of variances

23. Signal Processing Functions

23·1	REALDFT[6]	amplitude/phase real discrete Fourier transform
23·2	DFT[6]	discrete Fourier transform
23·3	IDFT[6]	discrete inverse Fourier transform

23·4	INTERPOLATE[10]	2D and 3D function interpolation
23·4	INTERP[10]	2D and 3D function interpolation
23·5	DINTERP[10]	2D function derivative interpolation
23·6	LOOKUP[10]	interpolate linearly into a tabular function of 1 variable
23·7	SMOOTH[10]	smooth a tabular function of 1 or 2 variables
23·8	SPLINE[10]	spline interpolation of plane or space curve
23·9	SPLINEP[10]	spline interpolate parametric plane or space curve
23·10	CROSSCORR[6]	cross-correlation of sets of vectors
23·10	CROSSCOV[6]	covariance of sets of vectors
23·11	DECONV[6]	inverse convolution (deconvolution)
23·12	CONVOLVE[6]	convolution
23·13	LMIN[6]	local minima of 2D curve
23·14	LMAX[6]	local maxima of 2D curve
23·15	MMEAN[9]	moving mean
23·16	MMEDIAN[9]	moving median
23·17	MVAR[9]	moving variance
23·18	MSTDDEV[9]	moving standard deviation
23·19	MAVDEV[9]	moving absolute average deviation
23·20	MQUANTILE[9]	moving quantile
23·21	MONOT[9]	Monotonic smoothing filter
23·22	SMOOTHSPLINE[9]	optimal smoothing spline

24. Functions of Complex Arguments

24·1	CPROD[0]	complex multiplication
24·2	CDIV[0]	complex division
24·3	CPR[0]	polar-to-rectangular conversion
24·4	CRP[0]	rectangular-to-polar conversion
24·5	CPOW[0]	complex exponentiation

25. Linear Algebra Computations

25·1	DOT[4]	scalar product of vectors
25·2	DET[4]	matrix determinant
25·3	TRACE[4]	matrix trace
25·4	LINEQ[4]	solve linear equations
25·5	INVERSE[4]	matrix inverse
25·6	EIGEN[4]	matrix eigenvalues and eigenvectors
25·7	SVD[4]	singular value decomposition
25·8	RANK[4]	matrix rank
25·9	LENGTH[4]	vector length
25·10	MNORM[4]	matrix norms
25·11	MRHO[4]	spectral radius of a matrix
25·12	COND[4]	condition number of a matrix
25·13	QRFAC[4]	QR-factorization of a matrix

26. Matrix Data Processing Functions

26·1	NROWS[0]	number of rows of a matrix
26·2	NCOLS[0]	number of columns of a matrix
26·3	MSIZE[0]	size of a matrix
26·4	MAXV[0]	maximum element of a matrix
26·5	MINV[0]	minimum element of a matrix
26·6	MAXROW[0]	row index of maximum element in matrix

26·7	MAXCOL[0]	column index of maximum element in matrix
26·8	MINROW[0]	row index of minimum element in matrix
26·9	MINCOL[0]	column index of minimum element in matrix
26·10	COMPRESS[0]	censor a matrix
26·11	EXTRACT[0]	extract rows from a matrix
26·12	SORT[0]	sort a matrix on a column
26·13	ROWSUM[0]	sum of the rows of a matrix
26·14	COLSUM[0]	sum of the columns of a matrix
26·15	PSUM[0]	partial row sums of a matrix
26·16	RDUP[6]	thresholded duplicate removal
26·17	TIES[9]	multiplicity of a matrix element
26·18	ELEMENT[9]	is a value in a matrix

27. Matrix Construction Functions

27·1	LIST[0]	construct a matrix from a list
27·2	CROSS[0]	tensor product of matrices
27·3	SHAPE[4]	restructure a matrix
27·4	DIAG[4]	build a matrix by diagonals
27·5	GETDIAG[4]	extract diagonals of a matrix
27·6	BAND[4]	build a diagonally-banded matrix
27·7	ROTATE[0]	rotate a matrix by rows or columns
27·8	MESH[0]	interleave two matrices by rows or columns

28. Graphics Computations

28·1	CONTOUR[12]	level lines of a surface
28·2	FILL[zz]	fill lines of a polygon
28·3	CHULL[7]	convex hull
28·4	CURVEM[0]	extract data of a curve or axis
28·5	LABELM[0]	extract label numbers of a curve or axis
28·6	PTSIZEM[0]	extract pointsize matrix of a curve or axis
28·7	WINDOWM[0]	extract limit numbers of a window
28·8	FRAMEM[0]	extract limit numbers of the frame of a window
28·9	IMAGEM[0]	extract limit numbers of the image of a window
28·10	STEPGRAPH[0]	construct step function graph matrix
28·11	BARGRAPH[0]	construct bar graph matrix
28·12	COLORN[0]	assign a color number each number
28·13	COLORX[0]	color number for an (R,G,B) triple

29. Functional Computations

29·1	INTEGRATE[1]	solve ordinary differential equations
29·2	QROMB[0]	definite integral
29·3	INTEGRAL[0]	definite integral
29·4	MINIMIZE[3]	minimize a function
29·4	MAXIMIZE[3]	maximize a function
29·5	LINPROG[3]	linear programming by the simplex method
29·6	SIMPLEX[3]	linear programming by the simplex method
29·7	EVAL[0]	evaluate by substitution
29·8	SUM[0]	sum of a sequence
29·9	PRODUCT[0]	product of a sequence
29·10	ROOT[0]	zero of a function of one argument
29·11	PROOT[0]	zeros of a polynomial (real or complex)

29·12	ITERATE[1]	solve first order difference equations
29·13	POINTS[0]	evaluate function(s) on a list of argument vectors
29·14	MAPPLY[0]	apply a function to matrix indices
29·15	LINLOG[0]	evaluate a function or data for linlog drawing
29·16	LOGLIN[0]	evaluate a function or data for loglin drawing
29·17	LOGLOG[0]	evaluate a function or data for log-log drawing
29·18	RPOLAR[0]	evaluate radial polar function for cartesian drawing
29·19	TPOLAR[0]	evaluate azimuthal polar function for cartesian drawing
29·20	PPOLAR[0]	parameterized polar functions for cartesian drawing

30. Input/Output Functions

30·1	READ[0]	read a matrix from a file
30·2	KREAD[0]	read from the keyboard one or more numbers
30·3	WREAD[0]	enter pairs of coordinates via the mouse
30·4	TYPEOUT[0]	type-out a value from within a function
30·5	ETIME[0]	elapsed time
30·6	JDATE[0]	Julian date
30·7	DTYPE[0]	data type of an object
30·8	CORE[0]	available workspace
30·9	KSREAD[0]	read a string from the keyboard
30·10	READON[0]	read a matrix from a file without closing
30·11	GETSTRINGS[0]	displays a form and returns user entries
30·12	MENUCHOICE[0]	display a menu and return user choice
30·13	RUN[0]	Run an independent program returning a matrix
30·14	SRUN[0]	Run an independent program returning a string array

31. String-valued Functions

31·1	STRVAL[0]	build a string from a defined object
31·2	STRTIME[0]	build a string with the current time
31·3	STRDATE[0]	build a string with the current date
31·4	TITLESTR[0]	get the string of a title
31·5	SUBSTR[0]	get a substring of a string
31·6	STRLEN[0]	length of a string
31·7	STRREV[0]	reverse a string
31·8	STRDBLANK[0]	deblank a string

32. Cluster Analysis Functions

32·1	ASCALE[7]	translation to zero mean and scaling to unit variance
32·2	RSCALE[7]	minimum/maximum scaling
32·3	DISTS[7]	Minkowsky p-metric, Euclidean distance
32·4	FISHERRK[7]	Fisher's rank-correlation
32·5	PRCOMP[7]	principal components of a covariance matrix
32·6	NLM[7]	dimensionality reduction using nonlinear mapping
32·7	KMEANS[7]	K-means clustering
32·8	CLUSTINFO[7]	cluster summary table
32·9	CLUSTERR[7]	clustering error
32·10	DELAUN[7]	Delaunay triangulation
32·11	DELCURVE[7]	construct matrix for drawing Delaunay triangulation
32·12	DCIRCLES[7]	matrix for circumcircles of Delaunay triangulation
32·13	VORREGIONS[7]	compute Voronoi diagram
32·14	VORCURVE[7]	construct matrix for drawing Voronoi diagram

32·15	VORSTAT[7]	edge-count, area and perimeter of Voronoi regions

33. Tree Cluster Analysis Functions

33·1	MST[7]	minimal spanning tree
33·2	TREECURVE[7]	tree/graph matrix for drawing
33·3	SLINK[7]	single linkage dendrogram
33·4	ALINK[7]	average linkage dendrogram
33·5	CLINK[7]	complete linkage dendrogram
33·6	CENTROID[7]	centroid linkage dendrogram
33·7	WARD[7]	Ward's linkage dendrogram
33·8	DENCURVE[7]	dendrogram tree matrix for drawing
33·9	COPHEN[7]	cophenetic correlation of a dendrogram
33·10	INCON[7]	inconsistency coefficients of a dendrogram
33·11	TREEGROUP[7]	regroups clusters from a dendrogram tree

34. Survival Analysis Functions

34·1	KMSURV[9]	Kaplan-Meier survival curve estimation
34·2	ACTSURV[9]	actuarial survival curve estimate
34·3	MHT[9]	Mantel-Haenszel-Armitage-Cochran test
34·4	THOMAST[9]	Thomas' test
34·5	SURV2T[9]	Mantel-Haenszel test on 2 group survival data
34·6	SURVT[9]	Mantel-Haenszel test on multiple group survival data
34·7	HBPOWER[9]	Mantel-Haenszel test power simulation

35. Statistical Power and Sample Size Calculations

35·1	STUTFA[9]	rejection error probability of the t-test for equal means
35·2	STUTFB[9]	acceptance error probability of t-test for equal means
35·3	STUTFN[9]	sample size of the t-test for equal means
35·4	STUTFT[9]	sample size ratio of the t-test for equal means
35·5	QFA[9]	rejection error probability of the variance-ratio F-test
35·6	QFB[9]	variance-ratio F-test acceptance error probability
35·7	QFN[9]	variance-ratio F-test sample size
35·8	QFE[9]	variance-ratio F-test alternate hypothesis variance ratio
35·9	CHISQFA[9]	chi square test rejection error probability
35·10	CHISQFB[9]	chi square test acceptance error probability
35·11	CHISQFN[9]	chi square test degrees of freedom
35·12	CHISQFD[9]	chi square test rms average bucket deviation

1. Basic Scalar and Matrix Arithmetic Operators

1·1. +,– unary and binary addition and subtraction

A + B	addition
A - B	subtraction
+ A	unary addition (equal to 0 + A)
- A	unary subtraction (equal to 0 - A)

If A and B are scalars, the usual scalar addition or subtraction operation is performed. If one of A and B is a matrix and the other is a scalar, the scalar is added or subtracted, element-by-element, to the matrix. If A and B are both matrices, element-by-element addition or subtraction is performed. If A and B are not the same shape, implicit cyclic extension of each array to the largest row and column size of both is done.

1·2. * multiplication

A ∗ B	multiplication

If both A and B are scalars, the usual scalar multiplication operation is performed. If one of A or B is a scalar and the other is a matrix, the element-by-element product of the matrix with the scalar is obtained. If both A and B are matrices, the matrix product of linear algebra is produced. In particular, if A is an $m \times n$ matrix and B is an $n \times p$ matrix, then A ∗ B is an $m \times p$ matrix, the $[i, j]$ element of which is

$$\sum_{k=1}^{n} A_{i,k} B_{k,j}.$$

If both A and B are matrices, they *must* be conformable, i.e. NCOLS(A) = NROWS(B). To obtain the element-by-element product of matrices, the star-prime ($*'$) operator may be used.

1·3. / division

A / B	division

If both A and B are scalars, the usual scalar division operation is performed. If one of A and B is a scalar and the other is a matrix, division is performed using the scalar and each element of the matrix, and the result has the same shape as the matrix. If both A and B are matrices, the operation is performed element-by-element using both matrices, which are implicitly cyclicly extended if necessary. Division by zero produces a warning message and the result MAXPOS∗SIGN(A).

Note that, for matrices, A/B does *not* mean A∗B^-1. To obtain the Moore-Penrose generalized matrix inverse of B, use the expression B^-1. To invert matrices by LU decomposition, the function INVERSE (or INV) may be used. To solve linear equations, the LINEQ function may be used.

1·4. ^ exponentiation

A^B	exponentiation (power operator)

If A and B are scalars, the usual scalar exponentiation operation is performed. If A is negative, then B must be an integer. 0^0 produces a warning message and the result 0.

If A is a scalar and B is a matrix, the exponentiation operation is applied using B element-by-element, producing a matrix with the same shape as B. If a is negative, then every element of B must be an integer.

If A is a matrix and B is a scalar, then B must be an integer. If $B < 0$ then the appropriate power of the Moore-Penrose generalized inverse of A is returned. When $B \neq -1$, A will be implicitly zero-expanded to be a square matrix, if necessary, before A^B is computed. For any matrix A, A^0 produces the identity matrix with the size $\max(\text{NROWS}(A), \text{NCOLS}(A))$. The inverse of a square non-singular matrix is identical to its Moore-Penrose generalized inverse, however the inverse may be obtained more efficiently by LU decomposition, using the INVERSE operator.

When $B = -1$, the Moore-Penrose inverse, A^-1, has NCOLS(A) rows and NROWS(A) columns.

If both A and B are matrices, the exponentiation operation is applied element-by-element, and both A and B will be cyclicly extended as needed to make them the same size. The size of the resulting matrix is $\max(\text{NROWS}(A), \text{NROWS}(B)) \times \max(HBOXNCOLS(A), HBOXNCOLS(B))$.

Examples:

Let $A = \begin{bmatrix} 2 & 0 \\ 0 & 2 \\ 0 & 1 \end{bmatrix}$. Then A^(-1)= $\begin{bmatrix} .5 & 4 & 0 \\ 0 & .4 & .2 \end{bmatrix}$. Also (A^0)^(-1)= $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

Let A = $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$. Then A^A = $\begin{bmatrix} 1 & 4 \\ 27 & 256 \end{bmatrix}$.

---

2. Relational Operators

2·1. <, <=, =, >=, >, NOT= relational (comparison) operators

A < B	less than test	1 if $A < B$ else 0
A > B	greater than test	1 if $A > B$ else 0
A = B	equality test	1 if $A = B$ else 0
A <= B	less than or equal test	1 if $A \leq B$ else 0
A >= B	greater than or equal test	1 if $A \geq B$ else 0
A NOT= B	inequality test	1 if $A \neq B$ else 0

These are the relational operators (`<, <=, =, >=, >, and NOT=`) These operators produce the scalar values 1 or 0, depending on whether the comparison being carried out is true or false. These operators are useful in the conditional (IF-THEN-ELSE) expressions and statement.

If both A and B are scalars, the described computation is performed.

If one of A and B is a matrix and the other is a scalar, the operation is applied using the scalar and the matrix element-by- element. The result is a matrix with the same shape as the matrix argument.

If both A and B are matrices, the result is a matrix M with $M[i,j] = A[i,j]\ r\ B[i,j]$, where r is the specified relational operator. A and B will be implicitly cyclicly extended if necessary. The shape of the result M is $\max(\text{NROWS}(A), \text{NROWS}(B)) \times \max(\text{NCOLS}(A), \text{NCOLS}(B))$.

The pre-defined MLAB scalars TRUE (=1) and FALSE (=0) are available for use in relational expressions.

---

3. Boolean Operators

3·1. NOT, AND, OR logical negation, and, or

NOT A	logical negation	1 if $A = 0$ else 0.
A AND B	logical and	1 if $A \neq 0$ and $B \neq 0$ else 0.
A OR B	logical or	1 if $A \neq 0$ or $B \neq 0$ else 0.

Expressions involving the Boolean operators AND, OR, and NOT produce the value 1 (if the expression is true) or 0 (if the expression is false). The predefined MLAB scalars TRUE (equal to 1), and FALSE (equal to 0) are available for convenience. These operators are useful in conditional (IF-THEN-ELSE) expressions and statements.

If both A and B are scalars, the described calculation is performed.

If one of A and B is a scalar and the other is a matrix, the operation is applied using the scalar and the matrix element-by element. The result is a matrix with the same shape as the matrix argument.

If both A and B are matrices, the result is a matrix M with $M[i,j] = A[i,j]$ rop $B[i,j]$, where rop is the specified Boolean operator. A and B will be cyclicly extended if necessary. The shape of M is $\max(\text{NROWS}(A), \text{NROWS}(B)) \times \max(\text{NCOLS}(A), \text{NCOLS}(B))$.

4. Conditional Computation Operator

4·1. IF-THEN-ELSE conditional operator

IF A THEN B ELSE C if $A \neq 0$ then B else C.

This is the conditional ternary operator. When an IF-THEN-ELSE expression is evaluated, only the appropriate expression (either the then-expression or the else-expression, depending upon whether A is true ($\neq 0$) or false ($= 0$)) will be evaluated.

If A is a scalar, the result will be either B or C. Ordinarily, but not necessarily, both B and C will be of the same type (either scalar or matrix).

If A is a matrix, the test is applied element-by-element to A. The result is a matrix M with the same shape as A. If B and C are both scalars, $M[i,j]$ will be B or C, depending upon $A[i,j]$. If one of B or C is a matrix, $M[i,j]$ will be set to be either the scalar or the $[i,j]$ element of the matrix, depending on the outcome of the test. If both B and C are matrices, $M[i,j]$ will be set to be the $[i,j]$ element of either B or C, depending on the outcome of the test. If B or C are not the same shape as A, they will be implicitly cyclicly extended as necessary.

Example: Let A = 1; B = 2; C = 3. Then, [IF A THEN B ELSE C] is equal to 2.

5. Matrix Manipulation Operators

5·1. : sequence generator

X : Y (colon) unit step sequence generator

The colon (:) operator is the unit step sequence generator "from X to Y in unit steps". X and Y must be scalars. If $Y \geq X$, the result is a 1-column matrix with the first element equal to X. The second element is X+1, the 3rd element is X+2, etc., and the last element is equal to $X + \lfloor Y - X \rfloor$, where $\lfloor Z \rfloor$ is the greatest integer less than or equal to Z. If $Y < X$, the result is a 1-column matrix with the first element equal to X. The second element is equal to X-1, the 3rd element is equal to X-2, etc., and the last element is equal to $X - \lfloor X - Y \rfloor$.

Example: 1:5 = [1,2,3,4,5]′ .

5·2. :: fixed step sequence generator

X : Y : Z (colon-colon operator) fixed step sequence generator

The colon-colon ( ) ternary operator is the fixed-step sequence generator. It produces the 1-column matrix of values: "from X to Y in steps of Z". X, Y, and Z must be scalars. Z must be non-zero. If Y = X, a 1×1 matrix whose [1,1] element is equal to X is constructed.

If $Y > X$, the result is a 1-column matrix with the first element equal to X. The second element is equal to X+ABS(Z), the 3rd element is equal to $X + 2\text{abs}(Z)$, etc. The final element is the last term in this increasing sequence which is less than or equal to Y.

If $Y < X$, The result is a 1-column matrix with the first element equal to X. The second element is equal to X-abs(Z), the 3rd element is equal to $X - 2\text{abs}(Z)$, etc. The final element is the last term in this decreasing sequence which is greater than or equal to Y.

Examples: 1:10:3 = [1 4 7 10]′ . -1:6:2 = [-1 1 3 5]′ .

5·3. :! fixed-size sequence generator

X : Y ! Z (colon-exclamation operator) fixed-size sequence generator

The colon-exclamation (: !) ternary operator is the fixed-number- of-members sequence generator. X, Y, and Z must be scalars. Z must be an integer. If Z is not an integer, a warning message is given and $\lfloor Z \rfloor$ is used. Now suppose Z is an integer.

If $Z > 0$, X : Y ! Z produces the 1-column matrix of values formed from z equally-spaced numbers running from X to Y.

If $Z \leq 0$, it means use 1-Z in place of Z and the 1-column matrix X:Y!(1-Z) is produced. There will be abs(Z)+1 values generated, corresponding to abs(Z) gaps.

If $Z = 1$ or $Y = X$, a 1×1 matrix whose [1,1] element is equal to X is constructed.

Now suppose $X \neq Y$, and $Z > 1$, and let s = (Y-X)/(Z-1). Note, if $Y > X$, then s is positive; if $Y < X$, then s is negative. The result is a Z-row vector. The first element is equal to X, the second element is equal to X+s, the third element is equal to X+2s, etc. The last, i.e. the Zth, element is equal to Y.

Examples: 10:15!3 = [10,13,15]′ . 10:16!-3 = [10,12,14,16]′ . 5:1!3 = [5,3,1]′ .

5·4. [ ] matrix subscripting

M[i, j] and M[i] (brackets operator) matrix subscripting

The brackets operator ([ ]) computes a scalar which is the value of an element in a 2-dimensional matrix. M must be a matrix unless it appears on the left-hand-side of an assignment statement (in which case it may be an unknown object which will be a matrix upon completion of the assignment). The paired brackets [ ] are the suscripting operator. i and j are scalars. $M[i, j]$ is the scalar value in the $\lfloor i \rfloor$-th row and $\lfloor j \rfloor$-th column of M. If $[i, j]$ indexes a non-existent value, an error message results. M[i] is an abbreviation for M[i, 1], and is often used to refer to matrices with a single column. While brackets are used in this manual for subscripting, they are not distinguished in MLAB from either parentheses or braces, all three of which may be used interchangeably.

5·5. ROW row submatrix construction

M ROW B row submatrix construction

The row operator constructs a matrix which is a row sub-matrix of a given matrix. M must be a matrix unless it appears on the left- hand-side of an assignment statement (in which case it may be an unknown object which will be a matrix upon completion of the assignment).

If B is a scalar, M ROW B is the ($\lfloor B \rfloor$-th row of M. If B is a matrix, M ROW B is the matrix formed by taking from M the rows specified by the elements of B. B may be specified as a parenthesized sequence of scalar and/or matrix values which is taken as an implicit argument to the built-in operator LIST described below. A value x in B specifies the row index $\lfloor x \rfloor$. Negative indices and indices beyond the last row of M result in an error message.

The ROW and COL operators may be used together to obtain a row-column submatrix; then the syntax M ROW A COL B or M COL B ROW A must be used.

The ROW operator may be iterated only if parentheses are used. Thus (M ROW B) ROW C means construct the B row sub-matrix of M, and then construct the C row submatrix of the previous result.

Note: M ROW i COL j is a 1×1 matrix, whereas $M[i, j]$ is a scalar.

Example:

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$. Then M ROW(3,1) = $\begin{bmatrix} 7 & 8 & 9 \\ 1 & 2 & 3 \end{bmatrix}$.

5·6. COL column submatrix construction

M COL B column submatrix construction

The col operator constructs a matrix which is a column sub-matrix of a given matrix. M must be a matrix unless it appears on the left-hand-side of an assignment statement (in which case it may be an unknown object which will be a matrix on completion of the assignment).

If B is a scalar, M COL B is the $\lfloor B \rfloor$-th column of M. If B is a matrix, M COL B is the matrix formed by taking from M the columns specified by the elements of B. B may be specified as a parenthesized sequence of scalar and/or matrix values which is taken as an implicit argument to the built-in operator LIST described below. A value x in B specifies the column index $\lfloor x \rfloor$. Negative indices and indices beyond the last row of M result in an error message.

The COL and ROW operators may be used together to obtain a row-column submatrix; then the syntax M COL A ROW B or M ROW B COL A must be used.

The COL operator may be iterated only if parentheses are used. Thus (M COL B) COL C means construct the B column-submatrix of M, and then construct the C column-submatrix of the previous result.

Note: M ROW i COL j is a 1×1 matrix, whereas M$[i, j]$ is a scalar.

Example:

Let M = $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$. Then M COL(3,1)= $\begin{bmatrix} 3 & 1 \\ 6 & 4 \end{bmatrix}$.

5·7. ' transpose of a matrix

A $'$ (prime operator) transpose of a matrix

The prime ($'$) operator constructs a matrix which is the transpose of a matrix. For any legal subscript pair $[i, j]$, $A'[i, j] = A[j, i]$. If A is a scalar, the result is a 1×1 matrix with its single element equal to A. If A is an $m \times n$ matrix, then A$'$ is a $n \times m$ matrix.

Example:

A = $\begin{bmatrix} 1 & 2 \end{bmatrix}$, then A′ = $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$, and vice versa.

5·8. & row-wise concatenation

A & B (ampersand operator) row-wise concatenation

The ampersand (&) operator constructs a matrix which is the row-wise concatenation of its arguments.

If A and B are both scalars, A & B is the 2-row matrix [A,B]′ .

If A is a scalar and B is a matrix, M = A & B has one more row than B and the same number of columns as B. In this case, each element of M ROW 1 = A, and M ROW 2: NROWS(M) = B.

If B is a scalar and A is a matrix, A & B has one more row than A, and the same number of columns as A. In this case, M row 1:NROWS(A) = A, and each element of M ROW NROWS(M) = B.

If A and B are both matrices, A & B has NROWS(A) + NROWS(B) rows, and it has max(NCOLS(A), NCOLS(B)) columns. In this case, the first NROWS(A) rows of the result are a copy of A, and the last NROWS(B) rows of the result are a copy of B. A and B are implicitly cyclicly extended if necessary.

Examples:

Let A = 4 and B = $\begin{bmatrix} 1 & 2 \\ 3 & 5 \end{bmatrix}$. Then A & B = $\begin{bmatrix} 4 & 4 \\ 1 & 2 \\ 3 & 5 \end{bmatrix}$.

Let A = $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and B= $\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$, then A & B = $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix}$.

5·9. &’ column-wise concatenation

A &′ B (ampersand-prime operator) column-wise concatenation

The ampersand-prime (&′) operator constructs a matrix which is the column-wise concatenation of its arguments.

If A and B are both scalars, A &′ B is the two-column matrix [A,B].

If A is a scalar and B is a matrix, M = A &′ B has one more column than B and the same number of rows as B. In this case, all the elements of M COL 1 are equal to A, and M COL (2:NCOLS(M)) = B.

If B is a scalar and A is a matrix, M = A &′ B has one more column than A and the same number of rows as A. In this case, M COL 1:NROWS(A) = A, and all the elements of the last column of M are equal to B.

If A and B are both matrices, A &′ B has NCOLS(A) + NCOLS(B) columns, and max(NROWS(A), NROWS(B)) rows. In this case, the first NCOLS(A) columns of the result are a copy of A, and the last NCOLS(B) columns of the result are a copy of B. A and B are implicitly cyclicly extended, if necessary, in order to compute A &′ B.

Examples:

Let A = 4 and B = $\begin{bmatrix} 1 & 2 \\ 3 & 5 \end{bmatrix}$. Then A & B = $\begin{bmatrix} 4 & 1 & 2 \\ 4 & 3 & 5 \end{bmatrix}$.

Let A = $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and B= $\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$, then A & B = $\begin{bmatrix} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \end{bmatrix}$.

5·10. ^^ row-wise replication

A ^^X (caret-caret operator) row-wise replication

The caret-caret (^^) operator constructs a matrix which is the X-fold row-wise replication of A.

X must be a scalar. A may be either a scalar or a matrix. X must be ≥ 0. $\lfloor X \rfloor$ is used in place of X if X is not an integer. A ^^X is equivalent to A & A & ... & A, where A occurs $\lfloor X \rfloor$ times in the expression.

Examples:

$1\text{^^}5 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$, $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}\text{^^}2 = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$, .5^^0 creates a 0×1 null matrix.

5·11. ^^ ' column-wise replication

A ^^′ X (caret-caret-prime operator) column-wise replication

The caret-caret-prime (^^′) operator constructs a matrix which is the X-fold column-wise replication of A.

X must be a scalar. A may be either a scalar or a matrix. X must be ≥ 0. $\lfloor X \rfloor$ is used in place of X if X is not an integer. A ^^′ X is equivalent to A &′ A &′ ... &′ A, where A occurs $\lfloor X \rfloor$ times in the expression.

Examples: $1\text{^^}'\ 5 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}'$, $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}\text{^^}'2 = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}$, .5^^′ 0 creates a 1×0 null matrix.

5·12. # vector cross product

M # N (sharp operator) vector cross product

The sharp (#) operator constructs a matrix which is the 3-space vector cross product of two 3-space vectors.

M and N must both be matrices representing vectors in 3-space.

If both M and N are 1×3 or 3×1 matrices, then the result is a 1×3 matrix, which is the vector cross product of M and N.

If both M and N are 3-column matrices, the rows of each of which are vectors in 3-space, then the result is a 3-column matrix R with MAX(NROWS(M),NROWS(N)) rows where R ROW i is the vector cross product of M ROW i and N ROW i, where M or N is implicitly cyclicly extended, if necessary, in order to compute M # N.

Recall that P = M#N is a vector from to origin perpendicular to the plane formed by the line from the origin to M and the line from the origin to N, extending in the direction which a right-handed screw would advance if the screw-driver blade angle were rotated clockwise from being parallel to M to being parallel to N. The length of P is the area of a parallelogram with adjacent sides formed by vectors parallel to M and N, respectively.

Analytically, in a Cartesian coordinate system,

M # N = (M[2]N[3]-M[3]N[2], M[3]N[1]-M[1]N[3], M[1]N[2]-M[2]N[1]).

Example: Let M = [1 0 0], and N = [0 1 0]. Then P = [0 0 1].

5·13. ON function application operator

F ON M function application operator

The on operator constructs a matrix which is the result of applying a function to a sequence of argument lists.

F must be a function of one or more scalar arguments. M must be an $r \times c$ matrix with at least as many columns as F has required arguments. The number of columns of M must be no greater than the total number of required and optional arguments of F. The result of F ON M is the $r \times 1$ matrix, the rows of which are the values of the function F applied successively to each row of the matrix M, taken as the first c arguments of F.

If F is a function which depends upon the solution to a system of ordinary differential equations, that system will be identified automatically and solved numerically. In this case, m must be a 1-column matrix of independent variable values on which the solution of the ordinary differential equation system will be tabulated. The column of values corresponding to F will be extracted and returned. The current values of the differential equation solver control variables (METHOD, ERRFAC, ODERPT, JACSW, DISASTERSW) apply when differential equations are solved by use of the ON operator; however MANDSW is taken as true, although its value is not changed.

5·14. *' element-by-element multiplication

A $*'$ B (star-prime operator)) element-by-element multiplication

The star-prime ($*'$) operator constructs a matrix which is the element-by-element product of its inputs.

If A and B are scalars, this operator produces a 1×1 matrix whose [1,1] element is the result of scalar multiplication of the scalars A and B.

If one of A and B is a scalar and the other is a matrix, the result is a matrix with the same shape as the matrix operand, where the $[i, j]$ element of the result is the product of the scalar and the $[i, j]$ element of the matrix operand.

If A and B are both matrices, the result is a matrix with MAX(NROWS(A), NROWS(B)) rows and MAX(NCOLS(A), NCOLS(B)) columns. The $[i, j]$ element of the result is the product of A$[i, j]$ and B$[i, j]$. If there are no elements in one matrix which correspond to elements the other matrix, the matrices A and B will be implicitly cyclicly extended in order to compute the result A $*'$ B.

Example: [1 2] $*'$ [3 4] = [3 8].

5·15. /' element-by-element division

A $/'$ B (slash-prime operator) element-by-element division

The slash-prime ($/'$) operator constructs a matrix which is the element-by-element quotient of its inputs.

If A and B are scalars, this operator produces a 1×1 matrix whose [1,1] element is the result of scalar division of the scalars A and B.

If one of A and B is a scalar and the other is a matrix, the result is a matrix with the same shape as the matrix operand, where the $[i, j]$ element of the result is the quotient formed in the specified order of the scalar and the $[i, j]$ element of the matrix operand.

If A and B are both matrices, the result is a matrix with MAX(NROWS(A), NROWS(B)) rows and MAX(NCOLS(A), NCOLS(B)) columns. The $[i, j]$ element of the result is the quotient of A$[i, j]$ and B$[i, j]$. If there are no elements in one matrix which correspond to elements the other matrix, the matrices A and B will be implicitly cyclicly extended in order to compute the result A $/'$ B.

Example: [3 6] $/'$ [3 2] = [1 3].

---

6. String Operators

6·1. + string concatenation

A + B	(plus operator) string concatenation

The plus operator "+" is the string concatenation operator to construct a string.

If A and B are both strings, the result string is the concatenation of A and B.

If either A or B (but not both) are scalars, the value of the scalar is converted to a string before concatenation.

Example: Let A = 5 and B = "2+3 = ". Then A+" cm" = "5 cm" and B+A = "2+3 = 5"

6·2. ^^ string replication

A ^^B	(caret -caret operator) string replication

The caret -caret operator "^^" is overloaded to be the string replication operator for creating a string.

A must be a string, and B must be a non-negative integer scalar; the result string is the B-fold replication of A. If B is zero, a null string is created.

Examples:

Let A = " " (one space), and B = 5. Then, A^^B = " " (five spaces). Let A = "abc", and B = 3. Then A^^B = "abcabcabc".

Let A = "2.45437". Then A+" "^^(10-STRLEN(A)) is a 10-character string with the value 2.45437 right-adjusted in the field. This facility is convenient for constructing precise strings in graphs.

6·3. <, <=, =, >=, > relational (comparison) string operators

A = B	equality test
A NOT= B	inequality test
A < B	less than test
A > B	greater than test
A >= B	greater than or equal test
A <= B	less than or equal test

The string relational operators produce the value 1 or 0, depending on whether the comparison being carried out is true or false.

The string equality test operator "=" is used to compare two strings for equality. The result is 1 if A and B are identical strings and 0 if the strings are not identical.

The string inequality test operator "NOT=" is used to compare two strings for inequality. The result is 1 if A and B are identical strings and 0 if the strings are not identical.

The string order relation operators <, <=, >=, and > are used to compare two strings lexicographically.

$A < B$ is true if the first $k-1$ characters of A are identical to the first $k-1$ characters of B, and the kth character of A is less than the kth character of B, where each character is considered as an ASCII number. Also, $A < B$ if A is a proper prefix of B.

$A > B$ is true if the first $k-1$ characters of A are identical to the first $k-1$ characters of B, and the kth character of A is greater than the kth character of B, where each character is considered as an ASCII number. Also, $A > B$ if A is a proper prefix of B.

$A >= B$ is true if $A = B$ is true or if $A > B$ is true.
$A <= B$ is true if $A = B$ is true or if $A < B$ is true.

If either A or B (but not both) are scalars, the value of the scalar is converted to a string before comparison.

---

7. Integer Arithmetic Functions and Combinatorial Functions

7·1. SIGN(X)^0 sign function

SIGN(X) computes a scalar which is the value of the sign function, SIGN(X) = if $X > 0$ then 1 else if $X < 0$ then -1 else 0. X must be a scalar.

7·2. INT(X)^0 integer part function

INT(X) computes a scalar which is the value of the entier function, the integer part of X. X must be a scalar. In effect INT(X) rounds X towards zero.

Examples: int(5.25) = 5; int(-5.25) = -5.

7·3. CEILING(X)^0 ceiling function

CEILING(X) computes a scalar which is the value of the ceiling function, $\lceil x \rceil$ = the least integer n such that $n \geq x$. X must be a scalar.

Examples: CEILING(5.25) = 6; CEILING(-5.25) = -5.

7·4. FLOOR(X)^0 floor function

FLOOR(X) computes a scalar which is the value of the floor function, $\lfloor x \rfloor$ = the greatest integer n such that $n \leq x$. X must be a scalar.

Examples: FLOOR(5.25) = 5; FLOOR(-5.25) = -6.

7·5. ROUND(X)^0 round to nearest integer

ROUND takes a scalar as an argument and returns a scalar that is the integer closest to the floating point value input.

Example: If Z = 3.2, then ROUND(Z) = 3.

7·6. MOD(X,Y)^0 modulus (integer division remainder) function

MOD(X,Y) computes a scalar which is the value of the modulus function,
$\mathrm{mod}(x, y) = x - y\lfloor x/y \rfloor$. X and Y must be scalars. Recall that when X and Y are positive integers, MOD(X,Y) is the integer division remainder of X/Y. For any $X \geq 0$, MOD(X,1) is the fractional part of X.

Example: For X an integer, MOD(X,2) is 1 if X is odd and 0 if X is even.

7·7. BINCOEF(X,Y)[5] binomial coefficient

BINCOEF(X,Y) computes a scalar which is the value of the binomial coefficient $\binom{X}{Y}$. X and Y must be scalars. Y should be an integer. If Y is not an integer, $\lfloor Y \rfloor$ is used. BINCOEF(X,Y) = 0 for $Y < 0$ or $Y > X$. Recall that

$$\binom{x}{y} = \frac{x(x-1)\ldots(x-y+1)}{y(y-1)\ldots 1}.$$

Recall that $\binom{x}{y}$ is the number of ways that y distinct things may be chosen from among x distinct things, without regard to order.

7·8. BERNUM(X)[5] Bernoulli number

BERNUM(X) computes a scalar which is the value of the $\lfloor X \rfloor$-th (signed) Bernoulli number $B_{\lfloor X \rfloor}$. X must be a scalar. If $X < 0$, a warning message will be given and zero will be returned. The definition of the Bernoulli numbers is $B_0 = 1$ and

$$B_n = -1^{\lfloor n/2 \rfloor + 1} \frac{2(n!)}{(2^n - 1)\pi^n} \left[1 + \frac{1}{3^n} + \frac{1}{5^n} + \frac{1}{7^n} + \ldots\right]$$

for $n > 0$.

Note: $B_0 = 1$, $B_1 = -1/2$, $B_2 = 1/6$, $B_4 = -1/30$, $B_6 = 1/42$, $B_8 = -1/30$, etc. Also, $B_{2k+1} = 0$ for all $k > 0$.

7·9. EULNUM(X)[5] Euler number

EULNUM(X) computes a scalar which is the value of the $\lfloor X \rfloor$-th (signed) Euler number: $E_{\lfloor X \rfloor}$. X must be a scalar. If $x < 0$, a warning message will be given and zero will be returned. Recall the definition:

$$E_n = \frac{2^{n+2} n!}{\pi^{n+1}} \left[1 - \frac{1}{3^{n+1}} + \frac{1}{5^{n+1}} - \frac{1}{7^{n+1}} + \ldots\right]$$

Note: $E_0 = 1$, $E_2 = -1$, $E_4 = 5$, $E_6 = -61$, $E_8 = 1385$, $E_{10} = -50521$, $E_{12} = 2702765$, etc. Also $E_{2k+1} = 0$ for all $k \geq 0$.

7·10. STIRL1(X,Y)[5] Stirling number of the first kind

STIRL1(X,Y) computes a scalar which is the value of the (unsigned) Stirling number of the first kind: $\left[\begin{array}{c} x \\ y \end{array} \right]$. X and Y must be scalars. If X or Y are not integers, $\lfloor X \rfloor$ is used for X and $\lfloor Y \rfloor$ is used for Y.

The Stirling number of the first kind, usually denoted $\left[\begin{array}{c} x \\ y \end{array} \right]$, is the number of permutations of $[1, 2, \ldots, x]$ which have exactly y cycles.

Recall the definition of the Stirling number of the first kind:

$\left[\begin{array}{c} x \\ y \end{array} \right] = 0$ for $x < 0$ or $y \leq 0$ or $y \geq x$,

$\left[\begin{array}{c} 0 \\ y \end{array} \right] =$ if $y = 0$ then 1 else 0, and,

$\left[\begin{array}{c} x \\ y \end{array} \right] = (x-1) \left[\begin{array}{c} x-1 \\ y \end{array} \right] + \left[\begin{array}{c} x-1 \\ y-1 \end{array} \right]$ for $x \geq 0$.

7·11. STIRL2(X,Y)[5] Stirling number of the second kind

STIRL2(X,Y) computes a scalar which is the value of the Stirling number of the second kind: $\left\{ \begin{array}{c} x \\ y \end{array} \right\}$. X and Y must be scalars. If X or Y are not integers, $\lfloor X \rfloor$ is used for X and $\lfloor Y \rfloor$ is used for Y. The Stirling number of the second kind, usually denoted $\left\{ \begin{array}{c} x \\ y \end{array} \right\}$, is the number of ways of partitioning the set of x elements $\{1, 2, \ldots, x\}$ into y non-empty subsets.

Recall the definition of the Stirling number of the second kind:

$\left\{ \begin{array}{c} x \\ y \end{array} \right\} = 0$ for $x < 0$, or $y \leq 0$, or $y > x$,

$\left\{ \begin{array}{c} 0 \\ y \end{array} \right\} =$ if $y = 0$ then 1 else 0, and, for $x \geq 0$,

$\left\{ \begin{array}{c} x \\ y \end{array} \right\} =$ (x-1)*stirl1(x-1,y-1)+y*stirl2(x-1,y)

7·12. PARTU(X)[5] unrestricted partitions of an integer

PARTU(X) computes a scalar which is the number of unrestricted integer partitions (without regard to order) of the integer $\lfloor X \rfloor$ into non-zero parts. X must be a scalar. If $X < 0$, a warning message will be given and zero will be returned.

Example: The unrestricted partitions of 5 are $\{5, 1+4, 2+3, 1+1+3, 1+2+2, 1+1+1+2,$ and $1+1+1+1+1\}$, so PARTU(5) = 7.

7·13. PARTD(X)[5] distinct partitions of an integer

PARTD(X) computes a scalar which is the number of partitions (without regard to order) of the integer $\lfloor X \rfloor$ into distinct non-zero parts. X must be a scalar. If $X < 0$, a warning message will be given and zero will be returned.

Example: The distinct partitions of 5 are $\{5, 1+4,$ and $2+3\}$, so PARTD(5) = 3.

7·14. GCD(X,Y)[5] greatest common divisor

GCD(X,Y) computes a scalar which represents the greatest common divisor of the scalars X and Y. If $X \leq 0$ or $Y \leq 0$, a warning message will be given and zero is returned.

Example:

GCD(1769,551) = 29

7·15. EGCD(X,Y)[0] extended greatest common divisor

EGCD(X,Y) computes a 3×1 matrix which contains the three scalars arising in the extended greatest common divisor computation . X and Y must be scalars. If $X \leq 0$ or $Y \leq 0$, a warning message will be given and a 3×1 matrix of zeros is returned.

The extended greatest common divisor computation on two positive integers m and n is defined as the determination of the three scalars a, b, and d such that $am + bn = d$, where $d = \gcd(m, n)$. EGCD(X,Y) returns a result matrix R, such that $R[1]X + R[2]Y = R[3]$

Example:

EGCD(1769,551) produces the matrix [5,-16,29]$'$, since 5∗1769- 16∗551 = 29.

7·16. FACTOR(X)^0 prime factors of an integer

FACTOR(X) computes a 1-column matrix containing the prime factors of the positive integer X. X must be a scalar. If $X \leq 0$ or is not an integer, an error message will be given. Since X must be an integer, it must be no bigger than the largest integer exactly expressible as a floating point number, i.e. $2^{53} - 1 \approx 9 \times 10^{15}$.

FACTOR returns a 1-column matrix containing the prime factors of its input in decreasing order.

Example:

FACTOR(30031) produces the matrix [509, 59]$'$, since 31001 = 509∗59.

7·17. PRIMES(X)^0 list of primes up to argument

PRIMES(X) computes a matrix containing all the prime numbers up to the positive integer X. X must be a scalar. If $X \leq 0$ or is not an integer, an error message will be given. If x is too large, there may not be enough memory to hold the answer.

PRIMES returns a 1-column matrix containing the prime numbers up to X in increasing order.

Example:

PRIMES(100) produces the matrix :

[2 3 5 7 11 13 17 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97]$'$.

7·18. NPRIME(X)^5 the X^{th} prime number

NPRIME(X) computes a scalar which is the prime number at position X in the sequence of prime numbers (starting with 2) . X must be a scalar. If $X \leq 0$ or is not an integer, an error message will be given. A sieve method is used, so space is required to store all the integers up to the X^{th} prime, which may exhaust available memory if X is too large.

Example: NPRIME(25)= 97.

7·19. DIVISORS(X)^0 divisors of an integer

DIVISORS(X) computes a 1-column matrix containing the divisors of the integer X . X must be a scalar. If $X \leq 0$ or is not an integer, an error message will be given. The divisors of X are sorted into increasing order.

Example: DIVISORS(144)= [2 3 4 6 8 9 12 16 24 36 48 72]$'$

7·20. CATNUM(X) Catalan number

CATNUM(X) computes an integer-valued scalar which is the X-th Catalan number C_X. The argument X must be a non-negative integer. If X is not an integer, $\lfloor X \rfloor$ will be used. If $X < 0$, zero will be returned and a warning message will be produced. Catalan numbers may be computed using binomial coefficients using the relation

$$C_n = \binom{2n}{n}/(n+1)$$

The Catalan numbers arise in various combinatorial problems. For example, C_n is the number of binary trees with n nodes. Also, C_{n-2} is the number of ways to divide a polygon with n vertices into triangles using only the vertex points.

7·21. PARITY(M) parity of a permutation

PARITY(M) computes a scalar which is the parity of the permutation contained in the argument M. M must be a 1-column matrix containing a permutation of the integers 1:NROWS(M). If M is not a permutation, zero is returned and a warning message will be produced.

PARITY(M) returns the value 1 if the parity of the permutation M is even, or -1 if the parity of the permutation M is odd. In physics, the parity of a permutation is called the Levi-Civita (or epsilon) symbol, and denoted ϵ_{ijk}, for example, in three dimensions.

Example: PARITY(LIST(1,3,2,4,6,5)) = 1

7·22. BERPOL(X,Y)[5] Bernoulli polynomial

BERPOL(X,Y) computes a scalar which is the value of the $\lfloor Y \rfloor$-th Bernoulli polynomial with argument X. X and Y must be scalars, and Y must be non-negative. If $Y < 0$, a warning message will be produced and zero will be returned.

The definition of the Bernoulli polynomials is given below, where B_k is the k^{th} Bernoulli number

$$\text{BERPOL}(X,Y) = \sum_{k=0}^{X} \binom{Y}{k} B_k X^{Y-k}$$

7·23. EULPOL(X,Y)[5] Euler polynomial

EULPOL(X,Y) computes a scalar which is the value of the $\lfloor Y \rfloor$-th Euler polynomial with argument X. X and Y must be scalars, and Y must be non-negative. If $Y < 0$, a warning message will be produced and zero will be returned.

The definition of the Euler polynomials is given below, where E_k is the kth Euler number.

$$\text{EULPOL}(X,Y) = \sum_{k=0}^{X} \binom{Y}{k} E_k X^{Y-k}$$

7·24. FACT(X)[5] factorial function

FACT(X) computes the factorial of X, i.e. $X!$. X must be a positive integer. Recall that $x! = 1{\cdot}2{\cdot}\ldots{\cdot}x$

Example: FACT(5) = 120

7·25. RFACT(X,Y)[5] rising factorial function

RFACT(X,Y) computes a scalar which is the value of the rising factorial function, also called Pochhammer's symbol. X and Y must be scalars. $\lfloor Y \rfloor$ must be non-negative.

The rising factorial function is defined as $\mathrm{RFACT}(X,Y) = X^{\overline{\lfloor Y \rfloor}}$ where $a^{\overline{b}} = a(a+1)\ldots(a+b-1)$.

For X not a negative integer, the rising factorial function is equal to $\Gamma(X+Y)/\Gamma(X)$.

Example: RFACT(5,3) = 210

---

8. Elementary Functions

Arguments to many of these functions, as indicated individually in their documentation, may be scalars or 2-element (2-row or 2- column) matrices representing complex numbers, with the real part of the complex number in the first element, and the imaginary part of the complex number in the second element.

8·1. ABS(X)[0] absolute value function

ABS(X) computes a scalar which is the value of the absolute value function. There are two cases:

- If X is a scalar, the absolute value function is defined as: abs(x) = if $x > 0$ then x else $-x$.
- If X is a 2-element vector, the first element $X[1]$ is taken as the real part x and the second element $X[2]$ is taken as the imaginary part y of the complex number $z = x + iy$. The absolute value of the complex number z is defined as: abs(z) = $(x^2 + y^2)^{1/2}$.

Examples: Let X = -5. Then ABS(X) = 5. Let Z = [4,3]. Then ABS(Z) = 5.

8·2. SQRT(X)[0] square root function

SQRT(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the square root function. There are two cases.

- If X is a scalar, the square root function is defined as $X^{1/2}$ when $X \geq 0$. If $X < 0$, SQRT(X) produces a warning message and returns zero.
- If X is a 2-element vector, the first element $X[1]$ is taken as the real part p and the second element $X[2]$ is taken as the imaginary part q of a complex number $z = p + iq$. z may also be written in the form $re^{i\theta} = r(\cos\theta + i\sin\theta)$, where r = abs(z), the modulus of z, and θ = arg(z), the phase angle of z. Then, the complex result is $\sqrt{z} = \sqrt{r}e^{i\theta/2} = \sqrt{r}\cos\frac{\theta}{2} + i\sqrt{r}\sin\frac{\theta}{2}$ represented as a 2 element matrix.

Examples: Let X = 25. Then SQRT(X) = 5. Let Z = [-1,0], i.e. the real number -1. Then SQRT(Z) = [0,1], i.e. the complex number i.

8·3. EXP(X)[0] exponential function

EXP(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the exponential function. There are two cases.

- If X is a scalar. The result is the scalar value e^X. X must be less than about 709 to avoid overflow. If EXP(X) overflows, MAXPOS is returned and a warning message is produced. If X is less than about -709, EXP(X) underflows. In this case EXP(X) produces 0. Recall that e, the base of natural logarithms, is the constant: 2.718281828459045...
- If X is a 2-element vector, the first element is taken as the real part p and the second element is taken as the imaginary part q of a complex number $z = p + iq$. Recall the definition: $e^z = e^p(\cos(q) + i\sin(q))$, and the result is a complex number $w = u + iv$, where $u = e^p\cos(q)$, and $v = e^p\sin(q)$, returned in the form of a 2-column vector of the form $[u, v]$. The range restrictions noted above for real arguments apply equally to the real part of complex arguments.

Examples: Let X = 1. Then EXP(X) = 2.718281828... . Let Z = 0&′π, i.e. the imaginary number $i\pi$. Then EXP(Z) = [1,0], i.e. the real number 1.

8·4. LOG(X[,Y])^0 logarithm function

LOG(X[,Y]) computes a scalar or a 2-column matrix representing a complex number, which is the value of the logarithm function. There are two cases.

- If X is a scalar, then a scalar is computed which is the base-y logarithm of X. Recall $\log_y(x) = \log(x)/\log(y)$. X and Y must be positive scalars. If the base Y is omitted, it defaults to $e = 2.718281828459\ldots$, i.e. the base of natural logarithms. LOG(X) is undefined for $X \leq 0$. If $X < 0$, LOG(X) produces a warning message and returns LOG(ABS(X)). If X = 0, LOG(X) produces a warning message and returns MAXNEG. Recall the relations: $\log_y(y^x) = x$ and $y^{\log_y(x)} = x$, for x and $y > 0$.
- If X is a 2-element vector, the first element is taken as the real part p and the second element is taken as the imaginary part q of a complex number $z = p + iq$. The second argument representing the base is not used. z may also be written in the form $re^{i\theta}$, where $r = \text{abs}(z)$, the modulus of z, and $q = \arg(z)$, the phase angle of z. Then $\log(z) = \log(r) + i\theta$, where $-\pi < \theta \leq \pi$. The result is a 2 column matrix representing the complex number $\log(r) + i\theta$. Note that $\log(z)$ is singular when $z = 0$. For any value y, if $X = [0, y]$, LOG(X) returns MAXNEG&′0, and a warning message is produced.

Examples: Let X = 100, and Y = 10. Then LOG(X,Y) = 2. Let Z = [1,1]. Then LOG(Z) = [.034657359, .785398163].

8·5. LN(X)^0 natural (base e) logarithm function

LN(X) computes a scalar which is the value of the natural (base e) logarithm function, ln(v), for the argument X. X must be a scalar. The function LN(X) may be used interchangeably with LOG(X). LN(X) is undefined for $X \leq 0$. If $X < 0$, LN(X) produces a warning message and returns LN(ABS(X)). If X = 0, LN(X) produces a warning message and returns MAXNEG. Recall the relations: $\ln(e^x) = x$ and $e^{\ln(x)} = x$, for $x > 0$.

8·6. LOG10(X)^5 base-10 logarithm function

LOG10(X) computes a scalar which is the value of the common (base 10) logarithm function, $\log_{10}(v)$ for the argument X. X must be a scalar. $\log_{10}(v)$ is undefined for $v < 0$. If $X < 0$, LOG10(X) produces a warning message and returns LOG10(ABS(X)). If X = 0, LOG10(X) produces a warning message and returns MAXNEG.

8·7. LOG2(X)[5] base-2 logarithm function

LOG2(X) computes a scalar which is the value of the base-2 logarithm function, $\log_2(v)$ for the argument X. X must be a scalar. $\log_2(v)$ is undefined for $v \leq 0$. If $X < 0$, LOG2(X) produces a warning message and returns LOG2(ABS(X)). If X = 0, LOG2(X) produces a warning message and returns MAXNEG.

8·8. MIN(X,Y,...)[0] minimum of list of scalars

MIN(X,Y,...) computes a scalar which is the value of the minimum of the one or more scalar arguments X, Y, ... X, Y, ... must be scalars.

8·9. MAX(X,Y,...)[0] maximum of list of scalars

MAX(X,Y,...) computes a scalar which is the value of the maximum of the one or more scalar arguments X, Y, ... X, Y, ... must be scalars.

---

9. Non-Elementary Functions

9·1. GAMMA(X)[0] Gamma function

GAMMA(X) computes a scalar which is the value of the gamma function, $\Gamma(v)$ for the argument X. X must be a scalar. If X is 0 or a negative integer, zero is returned and a warning message is given. Recall the relation to the factorial function: $\Gamma(v+1) = v!$, for $v = 0, 1, \ldots$.

9·2. LOGGAMMA(X)[0] logarithmic gamma function

LOGGAMMA(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the natural logarithm of the gamma function for the argument X.

If X is a scalar value, then LOGGAMMA(X) computes the real value which is $\log(\Gamma(x))$. The logarithm is useful because $\Gamma(x)$ grows so rapidly with x. Recall the relation to the factorial function: $\Gamma(x+1) = x!$, for $x = 0, 1, \ldots$.

If $X = 0$, MAXPOS is returned. If $X < 0$, zero is returned and a warning message is given. For $x < 0$, $\Gamma(x)$ changes sign in each unit interval so that $\log(\Gamma(x))$ is ill-behaved, and is singular for integer values of x. To compute $\Gamma(x)$ for $x < 0$, one may use the MLAB function GAMMA or the recursion relation:

$$\Gamma(1-x) = \frac{\pi}{\Gamma(x)\sin(\pi x)}$$

so, for $x < 0$,

$$\Gamma(x) = \frac{\pi}{e^{\log(\Gamma(1-x))}\sin(\pi(1-x))}.$$

If X is a 2-element vector, the first element is taken as the real part and the second element is taken as the imaginary part of a complex number z. A 2-column matrix $[u, v]$ is returned, representing the complex number $\log\Gamma(z) = u + iv$.

9·3. DIGAMMA(X)[0] derivative of the logarithmic Gamma function

DIGAMMA(X) computes the value of the derivative of the natural logarithm of the gamma function.

If X is a scalar value, then DIGAMMA(X) computes $d(\log(\Gamma(x))/dx)$. DIGAMMA(X) is singular for $X = 0, -1, -2, \ldots$. If X is equal to 0, a warning message is given and MAXPOS is returned. If X is a negative integer, a warning message is given and 0 is returned. If X is so close to zero or a negative integer that an inaccurate answer would result, a warning message is given.

If X is a 2-element vector, the first element is taken as the real part and the second element is taken as the imaginary part of a complex number z. A 2-column matrix $[u, v]$ is returned, representing the complex number $d(\log\Gamma(z)/dz) = u + iv$.

Examples: Let X = 1. Then DIGAMMA(X) = -.5772156649... (the Euler-Mascheroni constant). Let Z = [1,2]. Then DIGAMMA(Z) = [1.876078786431, .129646316310].

9·4. IGAMMA(X,Y)[5] incomplete Gamma function

IGAMMA(X,Y) computes a scalar which is the value of the incomplete gamma function, $\Gamma_p(x, y)$, with argument x and parameter y for the input arguments X and Y. X and Y must be scalars, with $X \geq 0$ and $Y > 0$. If $X < 0$ or $Y \leq 0$, zero is returned and a warning message is produced. Recall the definition:

$$\Gamma_p(x, y) = \frac{1}{\Gamma(x)} \int_0^y t^{x-1} e^{-t} dt.$$

Note: $\Gamma_p(x, \infty) = 1$. The un-normalized integral $\gamma(x, y) = \Gamma(x)\Gamma_p(x, y)$ is also sometimes called the incomplete gamma function, but not in MLAB. The relation between the chi-squared distribution function and the incomplete gamma function is CHISQF(X,Y) = IGAMMA(Y/2,X/2).

9·5. BETA(X,Y)[5] Beta function

BETA(X,Y) computes a scalar which is the value of the beta function, $B(x, y)$ for the input arguments X, Y. X and Y must be positive scalars. Otherwise, MAXPOS is returned and a warning message is produced. Recall the definition:

$$B(x, y) = \int_0^1 t^{x-1}(1 - t)^{y-1} dt$$

and the relations $B(x, y) = B(y, x)$ and $B(x, y) = \Gamma(x)\Gamma(y)/\Gamma(x + y)$.

9·6. IBETA(X,Y,Z)[5] incomplete beta function

IBETA(X,Y,Z) computes a scalar which is the value of the incomplete beta function, $I_x(y, z)$ for the input arguments X, Y, Z. X, Y and Z must be scalars, with $0 \leq X \leq 1$, $y > 0$, and $z > 0$. Otherwise, zero is returned and a warning message is produced. Recall the definition:

$$I_x(y, z) = \frac{1}{B(y, z)} \int_0^x t^{y-1}(1 - t)^{z-1}$$

and the relations $I_0(y, z) = 0$, $I_1(y, z) = 1$, and $I_x(y, z) = 1 - I(1 - x)(y, z)$. The unnormalized incomplete beta function $B_x(y, z) = B(y, z)I_x(y, z)$ is sometimes called the incomplete beta function, but not in MLAB. The MLAB function IBETA is equivalent to another MLAB function, BETAF, which is the beta statistical distribution function.

9·7. ERF(X)[5] error function

ERF(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the error function erf(x) for the input argument X.

If X is a scalar value, then the scalar value of the real function, erf is computed for the input argument X. Recall the definition:

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-u^2} du$$

and the relations: $\mathrm{erf}(\infty) = 1$, $\mathrm{erf}(x) = -\mathrm{erf}(-x)$, and $\mathrm{erf}(x) = \Gamma_p(0.5, x^2)$.

If X is a 2-element vector, the first element is taken as the real part and the second element is taken as the imaginary part of a complex number z. A 2-column matrix $[u, v]$ is returned, representing the complex number $\mathrm{ERF}(X) = u + iv$.

9·8. ERFC(X)[5] complementary error function

ERFC(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the complementary error function erf(x) for the input argument X.

If X is a scalar value, then the scalar value of the real function erfc is computed for the input argument X. Recall the definition:

$$\mathrm{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^0 e^{-u^2} du$$

and the relations: $\mathrm{erfc}(x) + erf(x) = 1$, and $\mathrm{erfc}(-x) = 2 - \mathrm{erf}(x)$, where $\mathrm{erf}(x)$ is the usual error function, and the special cases: $\mathrm{erfc}(0) = 1$, $\mathrm{erfc}(\infty) = 0$.

If X is a 2-element vector, the first element is taken as the real part and the second element is taken as the imaginary part of a complex number z . A 2-column matrix $[u, v]$ is returned, representing the complex number $\mathrm{ERFC}(X) = u + iv$.

9·9. EXPINTN(X,Y)[5] Yth order exponential integral function

EXPINTN(X,Y) computes a scalar which is the value of the Y-th order exponential integral function $E_y(x)$ with argument x and parameter y for the input X and Y. X and Y must be scalars. Y must be a non-negative integer and X must be non-zero. Otherwise, EXPINTN(X,Y) returns zero and produces a warning message. Recall the definitions:

$$E_0(x) = e^{-x}/x$$

$$E_1(x) = -sign(x) \int_x^\infty \frac{e^{-t}}{t} dt$$

and

$$E_y(x) = \frac{e^{-x} - xE_{y-1}(x)}{y-1}$$

for $y > 1$.

For negative values of X, the Cauchy principal value of the integral $E_1(X)$ is used.

The related MLAB function EXPINT(X), which is often called the exponential integral function and denoted $Ei(x)$, is related to $E_y(x)$ by $Ei(x) = -E_1(-x)$.

The related function, the logarithmic integral function, usually called $Li(x)$, is available in MLAB as LOGINT(X). Recall the relation: $Li(x) = Ei(\log(x))$.

9·10. SININT(X)[5] sine integral function

SININT(X) computes a scalar which is the value of the sine integral function, $Si(x)$ for the input X. X must be a scalar. Recall the definition:

$$Si(x) = \int_0^x \frac{\sin(t)}{t} dt.$$

9·11. COSINT(X)[5] cosine integral function

COSINT(X) computes a scalar which evaluates the cosine integral function, $Ci(x)$ for the input X. X must be a scalar. If $X < 0$, COSINT(X) returns COSINT(ABS(X)), and a warning message is produced. Recall the definition:

$$Ci(x) = \gamma - \log(x) + \int_0^x \frac{\cos(t) - 1}{t} dt$$

where γ is the Euler-Mascheroni constant = 0.5772156649.... .

9·12. EXPINT(X)[5] exponential integral function

EXPINT(X) computes a scalar which is the Cauchy principal value of the exponential integral function, usually called $Ei(x)$, for the input X. X must be a non-zero scalar. If X = 0, a warning message is produced and zero is returned.

Recall the definition:

$$Ei(x) = sign(x) \int_{-x}^{\infty} \frac{e^{-t}}{t} dt$$

for $x \neq 0$.

Recall the relation to the first order exponential integral function value $E_1(x)$, which is available in MLAB as EXPINTN(X,1): $Ei(x) = -E_1(-x)$.

9·13. LOGINT(X)[5] logarithmic integral function

LOGINT(X) computes a scalar which is the Cauchy principle value of the logarithmic integral function, usually called, $Li(x)$ for the input X. X must be a scalar greater than or equal to 1. If X is less than 1, LOGINT(X) returns zero and produces a warning message. Recall the definition:

$$Li(x) = \int_0^x \frac{dt}{\log(t)}.$$

9·14. SINHINT(X)[5] hyperbolic sine integral function

SINHINT(X) computes a scalar which is the value of the hyperbolic sine integral function, $Shi(x)$ for the input X. X must be a non-negative scalar. If X = 0, zero is returned. If $X < 0$, 0 is returned and a warning message is produced. The definition of $Shi(x)$ is:

$$Shi(x) = \int_0^x \frac{\sinh(t)}{t} dt.$$

9·15. COSHINT(X)[5] hyperbolic cosine integral function

COSHINT(X) computes a scalar which is the value of the hyperbolic cosine integral function, $Chi(x)$ for the input X. X must be a positive scalar. If $X \leq 0$, zero is returned and a warning message is produced. The definition of Chi(x) is:

$$Chi(x) = \gamma + \log(x) + \int_0^x \frac{\cosh(t) - 1}{t} dt$$

where γ is the Euler-Mascheroni constant, equal to 0.577215664...

9·16. JBETA(X,Y,Z)[5] function used in derivative of beta distribution function

JBETA(X,Y,Z) computes a scalar which is the value of an integral occurring in the derivative of the beta distribution function for the input X, Y and Z. X, Y, and Z must be scalars. If $X < 0$ or $X > 1$, or $Y \leq 0$, JBETA(X,Y,Z) returns MAXNEG and produces a warning message. The definition of JBETA(X,Y,Z) is:

$$\text{JBETA}(X, Y, Z) = \int_0^X \log(t) t^{Y-1} (1-t)^{Z-1} dt.$$

9·17. JGAMMA(X,Y)[5] function used in derivative of gamma distribution function

JGAMMA(X,Y) computes a scalar which is the value of an integral occurring in the derivative of the gamma distribution function for the inputs X and Y. X and Y must be scalars. If $X < 0$, or $Y \leq 0$, JGAMMA(X,Y) returns MAXPOS and produces a warning message. The definition of JGAMMA(X,Y) is:

$$\text{JGAMMA}(X, Y) = \int_0^X \log(t) t^{Y-1} e^{-t} dt.$$

9·18. SPENCE(X)[5] Spence's dilogarithm

SPENCE(X) computes a scalar which is the value of a form of Spence's dilogarithm function for the argument value X. The scalar X must be non-negative.

Recall that Spence's dilogarithm function is defined by the integral

$$\text{SPENCE}(X) = \int_0^X \frac{\log(1-y)}{y} dy.$$

For ABS(X) ≤ 1 , the following uniformly convergent series is valid.

$$\text{SPENCE}(X) = \sum_1^\infty \frac{X^k}{k^2}$$

Example: SPENCE(.5) = 0.582240526

9·19. DAWSON(X)[5] Dawson's integral

DAWSON(X) computes a scalar which is the value of a form of Dawson's integral for the argument value X.

Dawson's integral is defined by

$$\text{DAWSON}(X) = e^{-X^2} \int_0^X e^{y^2} dy.$$

Example: DAWSON(1) = 0.5380795069

---

10. Trigonometric Functions

Arguments to these functions may be either scalars or 2-element matrices representing complex numbers, with the real part of the complex number in the first element, and the imaginary part of the complex number in the second element. The corresponding output is a scalar value or a 2-element matrix representing the complex result.

10·1. SIN(X)[0] trigonometric sine function

SIN(X) computes a scalar, or a 2-column matrix representing a complex number, which is the value of the trigonometric sine function.

If X is a scalar value (in radians) then the scalar value of the real function sin(X) is computed. If X is so large that reduction to the interval $[0, 2\pi)$ would result in loss of precision, SIN(X) returns zero and a warning message will be given. Recall that $\sin(x)^2 + \cos(x)^2 = 1$, and the relation to the trigonometric cosecant function: csec(x) = 1/sin(x).

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$ represented by X. The definition $\sin(z) = \sin(a)\cosh(b) + i\cos(a)\sinh(b)$ is used to compute the complex result. A 2-column matrix $[u, v]$ is returned, representing the complex number SIN(X) $= u + iv$.

Examples: Let X = $\pi/4$. Then SIN(X) = .0707106781. Let Z = [1,2]. Then SIN(Z) = [3.15677851, -1.5960104]

10·2. SIND(X)[5] trigonometric sine function (argument in degrees)

SIND(X) computes a scalar which is the value of the trigonometric sine function with its argument in degrees. $\text{sind}(x) = \sin((\pi/180)x)$. X must be a scalar. If X is so large that reduction to the interval [0,360) would result in loss of precision, SIND(X) returns zero, and a warning message will be given.

10·3. COS(X)[0] trigonometric cosine function

COS(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the trigonometric cosine function.

If X is a scalar(in radians), then the scalar value of the real function cos(X) is computed. If X is so large that reduction to the interval $[0, 2\pi)$ would result in loss of precision, COS(X) returns

zero and a warning message will be given. Recall the definition of the trigonometric secant function $\sec(x) = 1/\cos(x)$.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. The definition $\cos(z) = \cos(x) \cdot \cosh(y) - i\sin(x) \cdot \sinh(y)$. A 2-column matrix $[u, v]$ is returned, representing the complex number $\cos(X) = u + iv$.

Examples: Let X = $\pi/4$. Then COS(X) = .0707106781. Let Z = [1,2]. Then COS(Z) = [2.032073,-3.0591779].

10·4. COSD(X)^5 trigonometric cosine function (argument in degrees)

COSD(X) computes a scalar which is the value of the trigonometric cosine function with its argument in degrees. $\mathrm{cosd}(x) = \cos((\pi/180)x)$. X must be a scalar. If X is so large that reduction to the interval [0,360) would result in loss of precision, COSD(X) returns zero and a warning message will be given.

10·5. TAN(X)^0 trigonometric tangent function

TAN(X) computes a scalar or a 2-column vector representing a complex number, which is the value of the trigonometric tangent function.

If X is a scalar value (in radians). Then the scalar value of the real function $\tan(x)$ is computed for the input X. If X is so large that reduction to the interval $[0, 2\pi)$ would result in loss of precision, TAN(X) returns zero, and a warning message will be given. Recall that $\tan(x)$ is singular for $x = (2n-1)\pi/2$, for $n = \ldots, -2, -1, 0, 1, 2, \ldots$. For these values of X, TAN(X) returns values near MAXPOS (for n odd) or MAXNEG (for n even).

Recall the relation: $\mathrm{cotan}(x) = 1/\tan(x)$.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. The definition $\tan(z) = [\sin(2x) + i\sinh(2y)]/[\cos(2x) + \cosh(2y)]$ is used to compute the result. $\tan(z)$ is singular at the points $z = \pm(2n-1)\pi/2$, for $n = \ldots, -1, 0, 1, \ldots$. When X takes on any of these values, TAN(X) returns 0 and produces a warning message. A 2-column matrix $[u, v]$ is returned, representing the complex number $\tan(z) = u + iv$.

Examples: Let X = $\pi/4$. Then TAN(X) = 1. Let Z = [1,2]. Then TAN(Z) = [.033812826, 1.01479361]

10·6. TAND(X)^5 trigonometric tangent function (argument in degrees)

TAND(X) computes a scalar which is the value of the trigonometric tangent function with its argument in degrees. $\mathrm{tand}(x) = \tan((\pi/180)x)$. X must be a scalar. If X is so large that reduction to the interval [0,360) would result in loss of precision, zero is returned and a warning message will be given. For $x = (2n-1)90$ with $n = \ldots, -2, --1, 0, 1, 2, \ldots$, tand(x) is singular. For these values of X, TAND(X) returns values near MAXPOS (for n odd) or MAXNEG (for n even).

10·7. SEC(X)^0 trigonometric secant function

SEC(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the trigonometric secant function.

If X is a scalar value (in radians), the scalar value of the real function $\sec(x)$ is computed for the input X. If X is so large that reduction to the interval $[0, 2\pi)$ would result in loss of precision, SEC(X)

returns MAXPOS, and a warning message will be given. sec(x) is singular for $x = (2n-1)\pi/2$, for $n = \ldots, -2, -1, 0, 1, 2, \ldots$. For these values, a value near MAXPOS is returned.

Recall the relation: sec(x) = 1/cos(x).

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. sec(z) is singular at the real values $z = \pm(2n-1)\pi/2$, for n an integer. When z takes on any of these values, SEC(Z) returns 0 and produces a warning message. A 2-column matrix $[u, v]$ is returned, representing the complex number sec(z) = $u + iv$.

Examples: Let X = $\pi/4$. Then SEC(X) = 1.414213563. Let Z = [1,2]. Then SEC(Z) = [.151176298, .226973675]

10·8. SECD(X)* trigonometric secant function (argument in degrees)

SECD(X) computes a scalar which is the value of the trigonometric secant function with its argument in degrees. secd(x) = sec($(\pi/180)x$). X must be a scalar. If X is so large that reduction to the interval [0,360) would result in loss of precision, SECD(X) returns MAXPOS, and a warning message will be given. secd(x) is singular for $x = (2n-1)90$, for $n = \ldots, -2, -1, 0, 1, 2, \ldots$. For these values of X, a value near MAXPOS is returned.

10·9. COSEC(X)[5] trigonometric cosecant function

COSEC(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the trigonometric cosecant function.

If X is a scalar value (in radians) the scalar value of the real function, cosec(x) is computed for the input X. If X is so large that reduction to the interval $[0, 2\pi)$ would result in loss of precision, COSEC(X) returns zero, and a warning message will be given. cosec(x) is singular for $x = n\pi$, with $n = \ldots, -2, -1, 0, 1, 2, \ldots$. For these values of X, a value near MAXPOS is returned.

Recall the relation: cosec(x) = 1/sin(x).

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. cosec(z) is singular at the real values $z = \pm n\pi$. When z takes on any of these values, zero is returned and a warning message is produced. A 2-column matrix $[u, v]$ is returned, representing the complex number cosec(z) = $u + iv$.

Examples: Let X = $\pi/2$. Then COSEC(X) = .707186781. Let Z = [1,2]. Then COSEC(Z) = [.228375065, 141363021].

10·10. COSECD(X)* trigonometric cosecant function (argument in degrees)

COSECD(X) computes a scalar which is the value of the trigonometric cosecant function with its argument in degrees. cosecd(x) = cosec($(\pi/180)x$). X must be a scalar. If X is so large that reduction to the interval [0,360) would result in loss of precision, COSECD(X) returns zero, and a warning message will be given. cosecd(x) is singular for $x = 180n$, for $n = \ldots, -2, -1, 0, 1, 2, \ldots$. For these values of x, a value near MAXPOS is returned.

10·11. COTAN(X)[5] trigonometric cotangent function

COTAN(X) computes a scalar or a 2-column matrix representing a complex number, which is which is the value of the trigonometric cotangent function.

If X is a scalar value (in radians), the scalar is returned that is the value of the real function, cotan(x). If x is so large that reduction to the interval $[0, 2\pi)$ would result in loss of precision, zero is returned

and a warning message given. cotan(x) is singular for $x = n\pi$, for $n = \ldots, -2, -1, 0, 1, 2, \ldots$. For these values of x, a value near MAXPOS (for x even) or near MAXNEG (for x odd) is returned.

Recall the relation cotan(x) = 1/tan(x).

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. cotan(z) is singular at the real values $z = \pm n\pi$. When z takes on any of these values, an error message is produced. A 2-column matrix $[u, v]$ is returned, representing the complex number cotan(z) $= u + iv$.

Examples: Let X = $\pi/4$. Then COTAN(X) = 1. Let Z = [1,2]. Then COTAN(Z) = [3.27977555e-2, -.984329226].

10·12. COTAND(X)* trigonometric cotangent function

COTAND(X) computes a scalar which is the value of the trigonometric cotangent function with its argument in degrees. cotand(x) = cotan($(\pi/180)x$). X must be a scalar (in radians). If X is so large that reduction to the interval $[0, 2\pi)$ would result in loss of precision, zero is returned, and a warning message given. cotand(x) is singular for $x = 180n$, with $n = \ldots, -2, -1, 0, 1, 2, \ldots$. For these values of x, a value near MAXPOS (for x even) or near MAXNEG (for x odd) is returned.

---

11. Inverse Trigonometric Functions

Arguments to most of these functions, as indicated individually in the documentation, may be scalars or 2-element (2-row or 2-column) matrices representing complex numbers, with the real part of the complex number in the first element, and the imaginary part of the complex number in the second number.

11·1. ASIN(X)^0 inverse trigonometric sine function

ASIN(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the inverse trigonometric sine function.

If X is a scalar value, then a scalar is returned which is the value (in radians) of the real function, arcsin(x) also denoted $\sin^{-1}(x)$. ASIN produces a result in radians which lies in the range $[-\pi/2, \pi/2]$. $\sin^{-1}(x)$ is undefined for abs(x) > 1. For abs(x) > 1, zero is returned and a warning message produced. Recall the relations:

$\text{cosec}^{-1}(x) = \sin^{-1}(1/x)$, $\sin^{-1}(\sin(x)) = x$, and $\sin(\sin^{-1}(x)) = x$, for abs(x) ≤ 1.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. The definition: $\sin^{-1}(z) = -i\log(iz + \sqrt{1 - z^2})$ is used to compute the complex result. A 2-column matrix $[u, v]$ is returned, representing the complex number $\sin^{-1}(z) = u + iv$.

Examples: Let X = .5. Then ASIN(X) = .523598775. Let Z = [1,2]. Then ASIN(Z) = [.427078586, 1.52857091]

11·2. ASIND(X)^0 inverse trigonometric sine function (result in degrees)

ASIND(X) computes a scalar which is the value of the inverse trigonometric sine function.

If X is a scalar value, then a scalar is returned which is the value (in degrees) of the real function, arcsin(x) also denoted $\sin^{-1}(x)$. ASIND produces a result in degrees which lies in the range $[-90, 90]$. $\sin^{-1}(x)$ is undefined for abs(x) > 1. For abs(x) > 1, zero is returned and a warning message produced. Recall the relations:

$\text{cosec}^{-1}(x) = \sin^{-1}(1/x)$, $\sin^{-1}(\sin(x)) = x$, and $\sin(\sin^{-1}(x)) = x$, for $\text{abs}(x) \leq 1$.

Examples: Let X = .5. Then ASIND(X) = 30.

11·3. ACOS(X)^0 inverse trigonometric cosine function

ACOS(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the inverse trigonometric cosine function.

If X is a scalar value, then a scalar is computed which is the value (in radians) of the real function $\arccos(x)$, also denoted $\cos^{-1}(x)$. ACOS produces a result in radians which lies in the range $[0, \pi]$. $\cos^{-1}(x)$ is undefined for $\text{abs}(x) > 1$. For $\text{abs}(x) > 1$, zero is returned and a warning message produced. Recall the relations:

$\cos(\cos^{-1}(x)) = x$, $\cos^{-1}(\cos(x)) = x$, for $\text{abs}(x) \leq 1$, $\sec^{-1}(x) = \arccos^{-1}(1/x)$.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. The definition: $\cos^{-1}(z) = -i\log(z + \sqrt{z^2 - 1})$ is used to compute the complex result. A 2-column matrix $[u, v]$ is returned, representing the complex number $\cos^{-1}(z) = u + iv$.

Examples: Let X = 0.5. Then ACOS(X) = 1.047187551. Let Z = [1,2]. Then ACOS(Z) = [.328499479, -1.21728328]

11·4. ACOSD(X)^0 inverse trigonometric cosine function (result in degrees)

ACOSD(X) computes a scalar which is the value of the inverse trigonometric cosine function.

If X is a scalar value, then a scalar is computed which is the value (in degrees) of the real function $\arccos(x)$, also denoted $\cos^{-1}(x)$. ACOSD produces a result in degrees which lies in the range $[0, 180]$. $\cos^{-1}(x)$ is undefined for $\text{abs}(x) > 1$. For $\text{abs}(x) > 1$, zero is returned and a warning message produced. Recall the relations:

$\cos(\cos^{-1}(x)) = x$, $\cos^{-1}(\cos(x)) = x$, for $\text{abs}(x) \leq 1$, $\sec^{-1}(x) = \arccos^{-1}(1/x)$.

Examples: Let X = 0.5. Then ACOSD(X) = 1.047187551.

11·5. ATAN(X)^0 inverse trigonometric tangent function

ATAN(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the inverse trigonometric tangent function.

If X is a scalar value x, then a scalar is computed which is the value (in radians) of the real function $\arctan(x)$ also denoted $\tan^{-1}(x)$. ATAN produces a result in radians which lies in $[-\pi/2, \pi/2]$. Recall the relations: $\tan^{-1}(\tan(x)) = x$, $\tan(\tan^{-1}(x)) = x$, and $\text{cotan}^{-1}(x) = \tan^{-1}(1/x)$.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. The definition: $\tan^{-1}(z) = i\log[(i + z)/(i - z)]/2$ is used to compute the complex result. A 2-column matrix $[u, v]$ is returned, representing the complex number $\tan^{-1}(z) = u + iv$.

Examples: Let X = .5. Then ATAN(X) = .463647609. Let Z = [1,2]. Then ATAN(Z) = [-.231823804, .402359478].

11·6. ATAND(X)^0 inverse trigonometric tangent function (result in degrees)

ATAND(X) computes a scalar which is the value of the inverse trigonometric tangent function.

If X is a scalar value x, then a scalar is computed which is the value (in degrees) of the real function arctan(x) also denoted $\tan^{-1}(x)$. ATAND produces a result in degrees which lies in $[-90, 90]$. Recall the relations: $\tan^{-1}(\tan(x)) = x$, $\tan(\tan^{-1}(x)) = x$, and $\text{cotan}^{-1}(x) = \tan^{-1}(1/x)$.

Examples: Let X = .5. Then ATAND(X) = .463647609.

11·7. ATAN2(Y,X)[0] 2-argument inverse trigonometric tangent function

ATAN2(Y,X) computes a scalar which is the value of the 2-argument inverse trigonometric tangent function arctan2(y, x) for the inputs X and Y. It is similar to the ordinary inverse tangent function $\tan^{-1}(y/x)$, with special attention taken to avoid the branch cut at x = 0. X and Y must be scalars. ATAN2 produces a result in radians which lies in the range $[-\pi, \pi]$. arctan2$(0, 0)$ is undefined. ATAN2(0,0) returns the value zero and a warning message is produced. In general, atan2(y, x) is the angle with respect to the positive x-axis of the vector from the origin to the point (x, y) in the cartesian plane.

11·8. ATAN2D(Y,X)[0] 2-argument inverse trigonometric tangent function (result in degrees)

ATAN2D(Y,X) computes a scalar which is the value of the 2-argument inverse trigonometric tangent function arctan2(y, x) for the inputs X and Y. It is similar to the ordinary inverse tangent function $\tan^{-1}(y/x)$, with special attention taken to avoid the branch cut at x = 0. X and Y must be scalars. ATAN2 produces a result in degrees which lies in the range $[-180, 180]$. arctan2$(0, 0)$ is undefined. ATAN2D(0,0) returns the value zero and a warning message is produced. In general, atan2(y, x) is the angle with respect to the positive x-axis of the vector from the origin to the point (x, y) in the cartesian plane.

11·9. ACOTAN(X)[5] inverse trigonometric tangent function

ACOTAN(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the inverse trigonometric tangent function.

If X is a scalar value, then a scalar is returned which is the value (in radians) of the real function arccotan(x), also denoted $\text{cotan}^{-1}(x)$. ACOTAN produces a result in radians which lies in $[0, \pi]$.

Recall the definition: arccotan(x) = arctan($1/x$)

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. A 2-column matrix $[u, v]$ is returned, representing the complex number $\text{cotan}^{-1}(z) = u + iv$.

Examples: Let X = .5. Then ACOTAN(X) = 1.107148718. Let Z = [1,2]. Then ACOTAN(Z) = [.231823804, -.402359478]

11·10. ASEC(X)[5] inverse trigonometric secant function

ASEC(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the inverse trigonometric secant function.

If X is a scalar value, then a scalar is computed which is the value (in radians) of the real function arcsec(x), also denoted $\sec^{-1}(x)$. $\sec^{-1}(x)$ is undefined for abs(x) < 1. For abs(x) < 1, ASEC returns the value zero and a warning message is produced. ASEC produces a result in radians which lies in $[0, \pi/2]$.

Recall the definition: $\sec^{-1}(x) = \cos^{-1}(1/x)$.

If X is a 2-element vector, the first element is taken as the real part x and the second element is taken as the imaginary part y of the complex number $z = x + iy$. A 2-column matrix $[u, v]$ is returned, representing the complex number $\text{sec}^{-1}(z) = u + iv$.

Examples: Let X = 2. Then ASEC(X) = 1.047197551. Let Z = [1,2]. Then ASEC(Z) = [-.251946887,-.263971954].

11·11. ACOSEC(X)[5] inverse trigonometric secant function

ACOSEC(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the inverse trigonometric secant function.

If X is a scalar value, then a scalar is computed which is the value (in radians) of the real function arccosec(x), also denoted $\text{cosec}^{-1}(x)$. arccosec(x) is undefined for abs(x) < 1. For abs(x) < 1, the value zero is returned and a warning message is produced. ACOSEC produces a result in radians which lies in $[0, \pi/2]$.

Recall the definition: $\text{cosec}^{-1}(x) = \sin^{-1}(1/x)$.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a+ib$. The definition: $\text{cosec}^{-1}(z) = \sin^{-1}(1/z)$ is used to compute the complex result. A 2-column matrix $[u, v]$ is returned, representing the complex number $\text{cosec}^{-1}(z) = u + iv$.

Examples: Let X = 2. Then ACOSEC(X) = .523598775. Let Z = [1,2]. Then ACOSEC(Z) = [-.186318054,-.39656823].

---

12. Hyperbolic Functions

Arguments to of these functions may be scalars or 2-element (2-row or 2-column) matrices representing complex numbers, with the real part of the complex number in the first element, and the imaginary part of the complex number in the second number.

12·1. SINH(X)[5] hyperbolic sine function

SINH(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the hyperbolic sine function.

If X is a scalar value, then a scalar is computed which is the value of the real function $\sinh(x) = (e^x - e^{-x})/2$. If abs($x$) is too large (greater than about 709 for the PC), SINH returns MAXPOS∗SIGN(X) and produces a warning message. Recall the relations: $\cosh^2(x) - \sinh^2(x) = 1$ and $\text{cosech}(x) = 1/\sinh(x)$, where cosech(x) is the hyperbolic cosecant function.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. The definition: $\sinh(z) = \sinh(x)\cos(y) - i\cosh(x)\sin(y)$ is used to compute the complex result. A 2-column matrix $[u, v]$ is returned, representing the complex number $\sinh(z) = u + iv$.

Examples:

Let X = 1. Then SINH(X) = 1.175201194.
Let Z = [1,2]. Then SINH(Z) = [-.489056250,1.48311925].

12·2. COSH(X)[5] hyperbolic cosine function

COSH(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the hyperbolic cosine function.

If X is a scalar value, then a scalar is computed which is the value of the real function $\cosh(x) = (e^x + e^{-x})/2$. If abs($x$) is too large (greater than about 709 for the PC), COSH returns MAXPOS and produces a warning message. cosh produces a result which lies in the range $[1, \infty]$.

Recall the relation: $\text{sech}(x) = 1/\cosh(x)$, where sech($x$) is the hyperbolic secant function.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. The definition: $\cosh(z) = \cosh(x)\cos(y) + i\sinh(x)\sin(y)$ is used to compute the complex result. A 2-column matrix $[u, v]$ is returned, representing the complex number $\sinh(z) = w = u + iv$.

Examples:

Let X = 1. Then COSH(X) = 1.543080635.
Let Z = [1,2]. Then COSH(Z) = [-.489056250,1.48311925].

12·3. TANH(X)[5] hyperbolic tangent function

TANH(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the hyperbolic tangent function.

If X is a scalar, then a scalar is computed which is the value of the real function $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$. tanh($x$) is defined for all x. However, if abs(x) is too large (greater than about 709 for the PC), TANH returns zero; and produces a warning message. TANH produces a result which lies in the range [-1,1]. Recall the relation: $\text{cotanh}(x) = 1/\tanh(x)$, where cotanh($x$) is the hyperbolic cotangent function.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a+ib$. The definition: $\tanh(z) = \sinh(z)/\cosh(z)$ is used to compute the complex result. A 2-column matrix $w = [u, v]$ is returned, representing the complex number $\tanh(z) = w = u + iv$.

Examples:

Let X = 1. Then TANH(X) = .761594156.
Let Z = [1,2]. Then TANH(Z) = [-.489056250,1.48311925].

12·4. COSECH(X)[5] hyperbolic cosecant function

COSECH(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the hyperbolic cosecant function.

If X is a scalar value, then a scalar is computed which is the value of the real function $\text{cosech}(x) = 2/(e^x - e^{-x})$. cosech($x$) is singular for $x = 0$. If $x = 0$, COSECH returns MAXPOS. If abs(x) is too large (greater than about 709 for the PC), COSECH returns zero and produces a warning message. COSECH produces a result which lies in the range $[-\infty, \infty]$.

Recall the relation: $\text{cosech}(x) = 1/\sinh(x)$.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. A 2-column matrix $[u, v]$ is returned, representing the complex number $\text{cosech}(z) = u + iv$.

Examples:

Let X = 1. Then COSECH(X) = .850918128.
Let Z = [1,2]. Then COSECH(Z) = [-.22150093,.635493799].

12·5. SECH(X)[5] hyperbolic secant function

SECH(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the hyperbolic secant function.

If X is a scalar value, then a scalar is computed which is the value of the real function $\text{sech}(x) = 2/(e^x + e^{-x})$. sech(x) is defined for all x. However, if abs(x) is too large (greater than about 709 on the PC), SECH returns zero and produces a warning message. SECH produces a result which lies in the range [0,1].

Recall the relation: $\text{sech}(x) = 1/\cosh(x)$.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. A 2-column matrix $w = [u, v]$ is returned, representing the complex number $\text{sech}(z) = u + iv$.

Examples:

Let X = 1. Then SECH(X) = .648054273.
Let Z = [1,2]. Then SECH(Z) = [-.22150093,.635493799].

12·6. COTANH(X)[5] hyperbolic cotangent function

COTANH(X) computes a scalar or a 2-column matrix representing a complex number, which is the value of the hyperbolic cotangent function.

If X is a scalar value, then a scalar is computed which is the value of the real function $\text{cotanh}(x) = (e^x + e^{-x})/(e^x - e^{-x})$. cotanh(x) is singular for $x = 0$. If $x = 0$, COTANH returns MAXPOS. If abs(x) is to large (greater than about 709 for the PC), COTANH returns zero and produces a warning message. COTANH produces a result which lies in the range $[-\infty, \infty]$.

Recall the relation: $\text{cotanh}(x) = 1/\tanh(x)$.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. The definition: $\text{cotanh}(z) = \sinh(z)/\cosh(z)$ is used to compute the complex result. A 2-column matrix $[u, v]$ is returned, representing the complex number $\text{cotanh}(z) = u + iv$.

Examples:

Let X = 1. Then COTANH(X) = 1.31303529.
Let Z = [1,2]. Then COTANH(Z) = [.821329797,.171383613].

---

13. Inverse Hyperbolic Functions

Arguments of these functions may be scalars or 2-element (2-row or 2-column) matrices representing complex numbers, with the real part of the complex number in the first element, and the imaginary part of the complex number in the second number.

13·1. ASINH(X)[5] inverse hyperbolic sine function

ASINH(X) computes a scalar, or a 2-column matrix representing a complex number, which is the value of the inverse hyperbolic sine function.

If X is a scalar value, then a scalar result is computed which is the value of the real function arcsinh, where $\text{arcsinh}(x) = \log(x + \sqrt{x^2 + 1})$. It is also denoted $\sinh^{-1}(x)$.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. The definition: $\sinh^{-1}(z) = \log(z + \sqrt{z^2 + 1})$ is used to compute the complex result. A 2-column matrix $[u, v]$ is returned, representing the complex number $\sinh^{-1}(z) = u + iv$.

Examples:

Let X = 1. Then ASINH(X) = .648054273.
Let Z = [1,2]. Then ASINH(Z) = [-.22150093,.635493799].

13·2. ACOSH(X)[5] inverse hyperbolic cosine function

ACOSH(X) computes a scalar, or a 2-column matrix representing a complex number, which is the value of the inverse hyperbolic cosine function.

If X is a scalar value, then a scalar result is computed which is the value of the real function arccosh, where $\text{arccosh}(x) = \log(x + \sqrt{x^2 - 1})$. It is also denoted $\cosh^{-1}(x)$. ACOSH is undefined for $X < 1$. If $X < 1$, then the value zero is returned and a warning message is produced.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. The definition: $\cosh^{-1}(z) = \log(z + \sqrt{z^2 - 1})$ is used to compute the complex result. A 2-column matrix $[u, v]$ is returned, representing the complex number $\cosh^{-1}(z) = u + iv$.

Examples:

Let X = 2. Then ACOSH(X) = 1.316957897.
Let Z = [1,2]. Then ACOSH(Z) = [-9.41132032e-2, 1.57079632].

13·3. ATANH(X)[5] inverse hyperbolic tangent function

ATANH(X) computes a scalar, or a 2-column matrix representing a complex number which is the value of the inverse hyperbolic tangent function.

If X is a scalar value, then a scalar result is computed which is the value of the real function arctanh, where $\text{arctanh}(x) = 0.5\log((1 + x)/(1 - x))$. It is also denoted $\tanh^{-1}(x)$. arctanh(x) is undefined for abs(x) ≥ 1. If abs(x) ≥ 1, the value zero is returned and a warning message is produced.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a + ib$. Recall the definition: $\cosh^{-1}(z) = \log(z + \sqrt{z^2 - 1})$ is used to compute the complex result. A 2-column matrix $[u, v]$ is returned, representing the complex number $\cosh^{-1}(z) = u + iv$.

Examples:

Let X = .5. Then ATANH(X) = .549306144.
Let Z = [1,2]. Then ATANH(Z) = [.173286795,1.17809725].

13·4. ACOTANH(X)[5] inverse hyperbolic cotangent function

ACOTANH(X) computes a scalar, or a 2-column matrix representing a complex number which is the value of the inverse hyperbolic cotangent function.

If X is a scalar value, then a scalar result is computed which is the value of the real function arccotanh. arccotanh(x) is also denoted $\text{cotanh}^{-1}(x)$. $\text{cotanh}^{-1}(x)$ is undefined for abs(x) ≤ 1. If abs(x) ≤ 1, zero is returned and a warning message is produced. Recall the definition:

$\text{cotanh}^{-1}(x) = 0.5\log((x + 1)/(x - 1))$.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a+ib$. The definition: $\text{cotanh}^{-1}(z) = \tanh^{-1}(1/z)$ is used to compute the complex result. A 2-column matrix [u,v] is returned, representing the complex number $\cosh^{-1}(z) = u + iv$.

Examples:

Let X = 2. Then ACOTANH(X) = 1.316957897.
Let Z = [1,2]. Then ACOTANH(Z) = [-9.41132032e-2, 1.57079632].

13·5. ASECH(X)[5] inverse hyperbolic secant function

ASECH(X) computes a scalar, or a 2-column matrix representing a complex number, which is is the value of the inverse hyperbolic secant function.

If X is a scalar value, then a scalar result is computed which is the value of the real function arcsech. arcsech(x) is also denoted $\text{sech}^{-1}(x)$. arcsech(x) is singular ($+\infty$) for $x = 0$. arcsech(x) is undefined if abs(x) ≥ 1. If $x = 0$ or abs(x) ≥ 1, then the value zero returned and a warning message is produced. Recall the definition:

$\text{sech}^{-1}(x) = \log(1/x + \sqrt{1/x^2 - 1})$.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a+ib$. The definition: $\text{sech}^{-1}(z) = \cosh^{-1}(1/z)$ is used to compute the complex result. A 2-column matrix $[u, v]$ is returned, representing the complex number $\text{sech}^{-1}(z) = u + iv$.

Examples:

Let X = 2. Then ASECH(X) = 1.316957897.
Let Z = [1,2]. Then ASECH(Z) = [-9.41132032e-2, 1.57079632].

13·6. ACOSECH(X)[5] inverse hyperbolic cosecant function

ACOSECH(X) computes a scalar or a 2-column matrix representing a complex number, which is is the value of the inverse hyperbolic cosecant function.

If X is a scalar value, then a scalar is computed which is the value of the real function arccosech(x). It is also denoted $\text{cosech}^{-1}(x)$. $\text{cosech}^{-1}(x)$ is singular for $x = 0$. ACOSECH(0) produces a warning message and returns zero. Recall the definition:

$\text{cosech}^{-1}(x) = \log(1/x + \sqrt{1/x^2 + 1})$.

If X is a 2-element vector, the first element is taken as the real part a and the second element is taken as the imaginary part b of the complex number $z = a+ib$. The definition: $\text{cosech}^{-1}(z) = \sinh^{-1}(1/z)$ is used to compute the complex result. A 2-column matrix $[u, v]$ is returned, representing the complex number $\text{cosech}^{-1}(z) = w = u + iv$.

Examples:

Let X = 2. Then ACOSECH(X) = 1.316957897.
Let Z = [1,2]. Then ACOSECH(Z) = [-9.41132032e-2, 1.57079632].

---

14. Bessel Functions

The Bessel Functions of the first and second kinds are the linearly independent solutions to Bessel's differential equation, which arises in solutions of Laplace's equation in cylindrical coordinates. The J and Y Bessel functions correspond to the real components of the solution, while the K and I bessel functions correspond to the imaginary components.

14·1. BESSJ(A,B)[5] Bessel function of the first kind

BESSJ(A,B) computes a scalar which is the value of the Bessel function of the first kind $J_b(a)$ for the inputs A and B. A and B must be scalars or 2-element matrices representing the real and imaginary part of a complex number. A is the argument. B is the order.

There are two cases. If both A and B are scalars, the result is a scalar. Bessel functions are undefined for negative real argument. If $A < 0$, zero is returned and a warning message is produced.

If either A or B are 2-element matrices representing complex numbers, the result is a 2-element matrix representing the complex result.

Recall that, for $x \geq 0$, $J_z(x)$, is a solution to Bessel's equation: $x^2y'' + xy' + (x^2 - z^2)y = 0$.

14·2. BESSY(A,B)[5] Bessel function of the second kind

BESSY(A,B) computes a scalar which is the value of the modified Bessel function of the second kind $Y_b(a)$, for the inputs A and B. $Y_b(a)$ is also called the Weber function and the Neumann function. A and B must be scalars or 2-element matrices representing complex numbers. A is the argument. B is the order.

There are two cases. If both A and B are scalars, the result is a scalar. Bessel functions are undefined for negative real argument. If $A < 0$, zero is returned and a warning message is produced. The order Z may take any real value. BESSY(0,Z) returns MAXNEG.

If either A or B are 2-element matrices representing complex numbers, the result is a 2-element matrix representing the complex result.

Recall that $Y_z(x)$, for $x \geq 0$, is a solution to Bessel's equation: $x^2y'' + xy' + (x^2 - z^2)y = 0$.

14·3. BESSI(A,B)[5] modified Bessel function of the first kind

BESSI(A,B) computes a scalar which is the value of the modified Bessel function of the first kind, $I_b(a)$, for the inputs A and B. $I_b(a)$ is also called hyperbolic Bessel functions. A and B must be scalars or 2-element matrices representing complex numbers. A is the argument. b is the order.

There are two cases. If both A and B are scalars, the result is a scalar. Bessel functions are undefined for negative real argument. If $A < 0$, zero is returned and a warning message is produced. The order B may take any real value.

If either A or B are 2-element matrices representing complex numbers, the result is a 2-element matrix representing a complex number.

Recall that for $x \geq 0$, $I_z(x)$ is a solution to the modified Bessel's equation: $x^2y''+xy'-(x^2-z^2)y = 0$.

14·4. BESSK(A,B)[5] modified Bessel function of the second kind

BESSK(A,B) computes a scalar which is is the value of the Bessel function of the second kind $K_b(a)$ for the inputs A and B. A and B must be scalars or 2-element matrices representing complex numbers. A is the argument and B is the order.

There are two cases. If both A and B are real scalars, the result is a real scalar. Bessel functions are undefined for negative real argument values. If $A < 0$, zero is returned and a warning message is produced. BESSK(0,B) produces MAXPOS. The order B may take any real value.

If either A or B are 2-element matrices representing complex numbers, the result is a 2-element matrix representing a complex number.

Recall that $K_b(a)$, for $a > 0$, is a solution $y(a)$ to the modified Bessel's equation: $a^2y''(a) + ay'(a) - (a^2 + b^2)y(a) = 0$.

14·5. AIRY(X)[5] Airy function Ai

AIRY(X) computes a scalar which is the value of the Airy function $Ai(x)$ for the argument value X. X must be a scalar.

Recall that the Airy functions $Ai(x)$ and $Bi(x)$ are linearly independent solutions to the differential equation $y'' = xy$.

The Airy functions are representable as linear combinations of Bessel functions of order $\pm 1/3$.

Example: AIRY(-1) = 0.53556088

14·6. BAIRY(X)[5] Airy Function Bi

BAIRY(X) computes a scalar which is the value of the Airy function Bi for the argument value X. X must be a scalar.

Recall that the Airy functions $Ai(x)$ and $Bi(x)$ are linearly independent solutions to the differential equation $y'' = xy$.

The Airy functions are representable as linear combinations of Bessel functions of order $\pm 1/3$.

Example: BAIRY(-1) = 0.10399739

14·7. AIRYD(X)[5] Airy Function Ai derivative

AIRYD(X) computes a scalar which is the value of the derivative of the Airy function $\frac{dAi(x)}{dx}$ for the input argument value X. X must be a scalar.

Recall that the Airy functions $Ai(x)$ and $Bi(x)$ are linearly independent solutions to the differential equation $y'' = xy$.

The derivatives of the Airy functions are representable as linear combinations of Bessel functions of order $\pm 2/3$.

Example: AIRYD(-1) = 0.01016057

14·8. BAIRYD(X)[5] Airy Function Bi Derivative

BAIRYD(X) Computes a scalar which is the value of the derivative of the Airy function $\frac{dBi(x)}{dx}$ for the input argument value X. X must be a scalar.

Recall that the airy functions Ai(X) and Bi(X) are linearly independent solutions to the differential equation $y'' = xy$.

The derivatives of the Airy functions are representable as linear combinations of Bessel functions of order $\pm 2/3$.

Example: BAIRYD(-1) = 0.59237563

---

15. Legendre Functions

15·1. ALEG(X,Y,Z)[5] associated Legendre function

ALEG(X,Y,Z) computes a scalar which is the value of the associated Legendre function, $P_z^y(x)$ for the inputs X, Y, and Z. X, Y, and Z must be scalars. Z, the degree, and Y, the order, will be truncated to integers if necessary. The argument X must be in the interval [0,1]; otherwise, a warning message is produced and zero is returned. The associated Legendre functions satisfy Legendre's equation:

$(1 - x^2)w'' - 2xw' + [z(z + 1) - y^2/(1 - x^2)]w = 0.$

15·2. LEG(X,Y)[5] Legendre function

LEG(X,Y) computes a scalar which is the value of the degree-Y Legendre function $P_y(x)$ with the inputs X and Y. X and Y must be scalars. The argument X must be in the interval [-1,1]; otherwise, a warning message is produced and zero is returned. Y is the degree Currently, Y must be an integer; otherwise, an error message is produced. Recall the relation to the associated Legendre function: LEG(X,Y) = ALEG(X,0,Y).

Legendre functions satisfy Legendre's equation: $(1-x^2)w'' - 2xw' + y(y+1)w = 0$.

---

16. Elliptic Functions, Elliptic Integrals, and Theta Functions

The definition of the Jacobian elliptic functions is made in terms of the function $\theta(x,y)$, where $\theta(x,y)$ is implicitly defined by the following fundamental equation as the value z such that:

$$x = \int_0^z \frac{d\theta}{\sqrt{1-y(\sin(\theta))^2}}.$$

z depends upon the parameters x and y. $(0 \le y \le 1)$. $\sqrt{y}$ is sometimes called the modulus value.

The Jacobi elliptic functions are inverses of elliptic integrals. For example:

$$sn^{-1}(x,y) = \int_0^x \frac{dt}{\sqrt{(1-t^2)(1-yt^2)}} \qquad cn^{-1}(x,y) = \int_0^x \frac{dt}{\sqrt{(1-t^2)(1-y+yt^2)}}$$

Reference: Methods of Theoretical Physics, Morse & Feshbach, Vol 1, p. 486.

16·1. EL2(X,V,Y,Z)[5] general elliptic integral of the second kind

EL2(X,V,Y,Z) computes a scalar which is the value of the general elliptic integral of the second kind. V, X, Y, and Z must be scalars. X must be non-negative or zero is returned. V is the parameter; it must lie in [0,1]; otherwise, zero is returned, and a warning message is produced. Recall the definition of EL2(X,V,Y,Z):

$$\mathrm{EL2}(X,V,Y,Z) = \int_0^X \frac{(Y+Zt)dt}{(1+t^2)\sqrt{(1+t^2)(1+Vt^2)}}$$

16·2. CEL(V,X,Y,Z)[5] general complete elliptic integral

CEL(V,X,Y,Z) computes a scalar which is the value of the general complete elliptic integral. V, X, Y, and z must be scalars. V is the parameter; it must lie in [0,1]; otherwise, zero is returned and a warning message is produced. Recall the definition:

$$\mathrm{CEL}(V,X,Y,Z) = \int_0^\infty \frac{(Y+Zt^2)dt}{(1+Xt^2)\sqrt{(1+t^2)(1+Vt^2)}}$$

16·3. LE1(X,Y)[5] Legendre elliptic integral of the first kind

LE1(X,Y) computes a scalar which is the value of the Legendre elliptic integral of the first kind, usually denoted $F(x,y)$. X and Y must be scalars. X is the angle parameter; it must lie in $[0,\pi/2]$. Y

is the parameter; it must lie in [0,1]. Otherwise zero is returned and a warning message is produced. Recall the definition of LE1(x,y):

$$\mathrm{LE1}(X,Y) = \int_0^X \frac{dt}{\sqrt{1 - Y(\sin(t))^2}}$$

16·4. LE2(X,Y)[5] Legendre elliptic integral of the second kind

LE2(X,Y) computes a scalar which is the value of the Legendre elliptic integral of the second kind, usually denoted $E(x, y)$. X and Y must be scalars. X is the angle; it must lie in $[0, \pi/2]$. Y is the parameter; it must lie in [-1,1]. Otherwise, zero is returned and a warning message is produced. Recall the definition of LE2(X,Y):

$$\mathrm{LE2}(X,Y) = \int_0^{\pi/2} \sqrt{1 - Y(\sin(t))^2} dt$$

16·5. CE1(X)[5] complete elliptic integral of the first kind

CE1(X) computes a scalar which the value of the complete elliptic integral of the first kind, usually denoted $K(x)$. X must be a scalar. X is the parameter; it must lie in [0,1]. Otherwise, zero is returned and a warning message is produced. Recall the relation: CE1(x) = LE1(π/2,x), where LE1 is the Legendre elliptic integral of the first kind.

16·6. CE2(X)[5] complete elliptic integral of the second kind

CE2(X) computes a scalar which is the value of the complete elliptic integral of the second kind, usually denoted $E(x)$. X must be a scalar. X is the parameter; it must lie in [0,1]. Otherwise, zero is returned, and a warning message is produced. Recall the relation: CE2(x) = LE2(π/2,x), where LE2 is the Legendre elliptic integral of the second kind.

16·7. CE3(X,Y)[5] complete elliptic integral of the third kind

CE3(X,Y) computes a scalar which is the value of the complete elliptic integral of the third kind, usually denoted $\Pi(x, y)$. X and Y must be scalars. X is the parameter; it must lie in [0,1]. Otherwise, zero is returned and a warning message is produced. Recall the relation: CE3(x,y) = CEL(x,y+1,1,1), where CEL is the general complete elliptic integral.

16·8. JACOBIAM(X,Y)[5] Jacobian am elliptic function

JACOBIAM(X,Y) computes a scalar which is the value of the Jacobian elliptic function, denoted $am(x, y)$. X and Y must be scalars. Y must lie in the interval [0,1]. Otherwise, zero is returned and a warning message is produced.

The function $am(x, y)$ is just the Jacobi function $\phi(x, y)$ described at the beginning of this section.

$$\mathrm{JACOBIAM}(X,Y) = \phi(x, y).$$

16·9. JACOBISN(X,Y)[5] Jacobian sn elliptic function

JACOBISN(X,Y) computes a scalar which is the value of the Jacobian elliptic function, denoted $sn(x, y)$. X and Y must be scalars. Y must lie in the interval [0,1]. Otherwise, zero is returned and a warning message is produced.

The definition of $sn(x, y)$ is made in terms of the Jacobi θ function, so that

$$\text{JACOBISN}(X, Y) = \sin(\phi(x, y)).$$

16·10. JACOBICN(X,Y)[5] Jacobian cn elliptic function

JACOBICN(X,Y) computes a scalar which is the value of the Jacobian elliptic function, denoted $cn(x, y)$. X and Y must be scalars. Y must lie in the interval [0,1]. Otherwise, zero is returned and a warning message is produced.

The definition of $cn(x, y)$ is made in terms of the Jacobi ϕ function, so that

$$\text{JACOBICN}(X, Y) = \cos(\phi(X, Y)).$$

16·11. JACOBIDN(X,Y)[5] Jacobian dn elliptic function

JACOBIDN(X,Y) computes a scalar which is the value of the Jacobian elliptic function, denoted $dn(x, y)$. X and Y must be scalars. Y must lie in the interval [0,1]. Otherwise, zero is returned and a warning message is produced.

The definition of $dn(X, Y)$ is made in terms of the Jacobi ϕ function, so that

$$\text{JACOBIDN}(X, Y) = \sqrt{1 - y(\sin\phi(x, y))^2}.$$

16·12. JACOBICD(X,Y)[5] Jacobian cd elliptic function

JACOBICD(X,Y) computes a scalar which is the value of the Jacobian elliptic function, denoted $cd(x, y)$. X and Y must be scalars. Y must lie in the interval [0,1]. Otherwise, zero is returned and a warning message is produced.

The definition of $cd(x, y)$ is made in terms of the principal Jacobian elliptic functions:

$\text{cd}(x, y) = \text{cn}(x, y)/\text{dn}(x, y)$.

16·13. JACOBIDC(X,Y)[5] Jacobian dc elliptic function

JACOBIDC(X,Y) computes a scalar which is the value of the Jacobian elliptic function, denoted $dc(x, y)$. X and Y must be scalars. Y must lie in the interval [0,1]. Otherwise, zero is returned and a warning message is produced.

The definition of $dc(x, y)$ is made in terms of the principal Jacobian elliptic functions dn and cn:

$\text{dc}(x, y) = \text{dn}(x, y)/\text{cn}(x, y)$.

16·14. JACOBINS(X,Y)[5] Jacobian ns elliptic function

JACOBINS(X,Y) computes a scalar which is the value of the Jacobian elliptic function, denoted $ns(x, y)$. X and Y must be scalars. Y must lie in the interval [0,1]. Otherwise, zero is returned and a warning message is produced.

The definition of $ns(x, y)$ is made in terms of the principal Jacobian elliptic functions:

$\mathrm{ns}(x, y) = 1/\mathrm{sn}(x, y)$.

16·15. JACOBISD(X,Y)[5] Jacobian sd elliptic function

JACOBISD(X,Y) computes a scalar which is the value of the Jacobian elliptic function, denoted $sd(x, y)$. X and Y must be scalars. Y must lie in the interval [0,1]. Otherwise, zero is returned and a warning message is produced.

The definition of $sd(x, y)$ is made in terms of the principal Jacobian elliptic functions sn and dn:

$\mathrm{sd}(x, y) = sn(x, y)/dn(x, y)$.

16·16. JACOBINC(X,Y)[5] Jacobian nc elliptic function

JACOBINC(X,Y) computes a scalar which is the value of the Jacobian elliptic function, denoted $nc(x, y)$. X and Y must be scalars. Y must lie in the interval [0,1]. Otherwise, zero is returned and a warning message is produced.

The definition of $nc(x, y)$ is made in terms of the principal Jacobian elliptic function cn:

$\mathrm{nc}(x, y) = 1/\mathrm{cn}(x, y)$.

16·17. JACOBIDS(X,Y)[5] Jacobian ds elliptic function

JACOBIDS(X,Y) computes a scalar which is the value of the Jacobian elliptic function, denoted $ds(x, y)$. X and Y must be scalars. Y must lie in the interval [0,1]. Otherwise, zero is returned and a warning message is produced.

The definition of $ds(x, y)$ is made in terms of the principal Jacobian elliptic functions dn and sn:

$\mathrm{ds}(x, y) = \mathrm{dn}(x, y)/\mathrm{sn}(x, y)$.

16·18. JACOBIND(X,Y)[5] Jacobian nd elliptic function

JACOBIDND(X,Y) computes a scalar which is the value of the Jacobian elliptic function, denoted $nd(x, y)$. X and Y must be scalars. Y must lie in the interval [0,1]. Otherwise, zero is returned and a warning message is produced.

The definition of $nd(x, y)$ is made in terms of the principal Jacobian elliptic function dn:

$\mathrm{nd}(x, y) = 1/\mathrm{dn}(x, y)$.

16·19. JACOBISC(X,Y)[5] Jacobian sc elliptic function

JACOBIDSC(X,Y) computes a scalar which is the value of the Jacobian elliptic function, denoted $sc(x, y)$. X and Y must be scalars. Y must lie in the interval [0,1]. Otherwise, zero is returned and a warning message is produced.

The definition of $sc(x, y)$ is made in terms of the principal Jacobian elliptic functions sn and cn:

$\mathrm{sc}(x) = \mathrm{sn}(x, y)/\mathrm{cn}(x, y)$.

16·20. JACOBICS(X,Y)[5] Jacobian cs elliptic function

JACOBIDCS(X,Y) computes a scalar which is the value of the Jacobian elliptic function, denoted $cs(x, y)$. X and Y must be scalars. Y must lie in the interval [0,1]. Otherwise, zero is returned and a warning message is produced.

The definition of $cs(x, y)$ is made in terms of the principal Jacobian elliptic functions cn and sn:

$\mathrm{cs}(x, y) = \mathrm{cn}(x, y)/\mathrm{sn}(x, y)$.

16·21. THETA1(X,Y)[5] Jacobi's first theta function

THETA1(X,Y) computes a scalar which is the value of Jacobi's first theta function with argument x and parameter y. The value of y must lie in [0,1]. Otherwise, a warning message will be given and zero will be returned.

Let $K(y)$ denote the complete elliptic integral of the first kind implemented by CE1, and define $q = \exp(-\pi K(1 - y)/K(y))$, and let $z = \pi x/(2K(y))$.

THETA1(X,Y) is defined by the following infinite series which converges with extreme rapidity because q is never larger than 0.04321... .

$$\mathrm{THETA1}(X, Y) = 2q^{1/4} \sum_{n=0}^{\infty} (-1)^n q^{n(n+1)} \sin[2(n + 1)z]$$

16·22. THETA2(X,Y)[5] Jacobi's second theta function

THETA2(X,Y) computes a scalar which is the value of Jacobi's second theta function with argument X and parameter Y. The value of Y must lie in [0,1]. Otherwise, a warning message will be given and zero will be returned.

Let K(Y) denote the complete elliptic integral of the first kind implemented by CE1, and define $q = \exp(-\pi K(1 - Y)/K(Y))$, and let $z = \pi x/(2K(Y))$.

THETA2(X,Y) is defined by the following infinite series which converges with extreme rapidity because q is never larger than 0.04321... .

$$\mathrm{THETA2}(X, Y) = 2q^{1/4} \sum_{n=0}^{\infty} (-1)^n q^{n(n+1)} \cos[2(n + 1)z]$$

16·23. THETA3(X,Y)[5] Jacobi's third theta function

THETA3(X,Y) computes a scalar which is the value of Jacobi's third theta function with argument X and parameter Y. The value of Y must lie in [0,1]. If it does not, a warning message will be given and zero will be returned.

Let K(y) denote the complete elliptic integral of the first kind implemented by CE1, and define $q = \exp(-\pi K(1 - y)/K(y))$, and let $z = \pi x/(2K(y))$.

THETA3(X,Y) is defined by the following infinite series which converges with extreme rapidity because q is never larger than 0.04321... .

$$\mathrm{THETA3}(X, Y) = 1 + 2 \sum_{0}^{\infty} q^{n^2} \cos(2nz)$$

16·24. THETA4(X,Y)[5] Jacobi's fourth theta function

THETA4(X,Y) computes a scalar which is the value of Jacobi's fourth theta function with argument X and parameter Y. The value of Y must lie in [0,1]. Otherwise, a warning message will be given and zero will be returned.

Let K(y) denote the complete elliptic integral of the first kind implemented by CE1, and define $q = \exp(-\pi K(1-y)/K(y))$, and let $z = \pi x/(2K(y))$.

THETA4(X,Y) is defined by the following infinite series which converges with extreme rapidity because q is never larger than 0.04321... .

$$\text{THETA4}(X,Y) = 1 + 2\sum_{0}^{\infty} -1^n q^{n^2} \cos(2nz)$$

16·25. THETAS(X,Y)[5] Neville's s theta function

THETAS(X,Y) computes a scalar which is the value of Neville's s theta function with argument X and parameter Y. The value of Y must lie in [0,1]. Otherwise, a warning message will be given and zero will be returned.

Let K(y) denote the complete elliptic integral of the first kind implemented by CE1, and define $q = \exp(-\pi K(1-y)/K(y))$, and let $z = \pi x/(2K(y))$.

THETAS(X,Y) is defined by the following infinite series which converges with extreme rapidity because q is never larger than 0.04321... .

$$\text{THETAS}(X,Y) = (2\pi/K(Y))^{1/2}(q/(Y(1-Y)))^{1/4}\sum_{0}^{\infty} -1^n q^{n(n+1)} \sin[2(n+1)z]$$

16·26. THETAC(X,Y)[5] Neville's c theta function

THETAC(X,Y) computes a scalar which is the value of Neville's c theta function with argument X and parameter Y. The value of Y must lie in [0,1]. Otherwise, a warning message will be given and zero will be returned.

Let K(y) denote the complete elliptic integral of the first kind implemented by CE1, and define $q = \exp(-\pi K(1-y)/K(y))$, and let $z = \pi x/(2K(y))$.

THETAC(X,Y) is defined by the following infinite series which converges with extreme rapidity because q is never larger than 0.04321... .

$$\text{THETAC}(X,Y) = (2\pi/K(Y))^{1/2}(q/Y)^{1/4}\sum_{0}^{\infty} -1^n q^{n(n+1)} \cos[2(n+1)z]$$

16·27. THETAD(X,Y)[5] Neville's d theta function

THETAD(X,Y) computes a scalar which is the value of Neville's d theta function with argument X and parameter Y. The value of Y must lie in [0,1]. Otherwise, a warning message will be given and zero will be returned.

Let K(y) denote the complete elliptic integral of the first kind implemented by CE1, and define $q = \exp(-\pi K(1-y)/K(y))$, and let $z = \pi x/(2K(y))$.

THETAD(X,Y) is defined by the following infinite series which converges with extreme rapidity because q is never larger than 0.04321... .

$$\text{THETAD}(X,Y) = (\pi/2K(Y))^{1/2} \sum_{0}^{\infty} q^{n(n+1)} \cos(2nz)$$

16·28. THETAN(X,Y)[5] Neville's n theta function

THETAN(X,Y) computes a scalar which is the value of Neville's n theta function with argument X and parameter Y. The value of Y must lie in [0,1]. Otherwise, a warning message will be given and zero will be returned.

Let K(Y) denote the complete elliptic integral of the first kind implemented by CE1, and define $q = \exp(-\pi K(1-y)/K(y))$, and let $z = \pi x/(2K(y))$.

THETAN(X,Y) is defined by the following infinite series which converges with extreme rapidity because q is never larger than 0.04321... .

$$\text{THETAN}(X,Y) = (1/Y)^{1/4}(\pi/2K(Y))^{1/2} \sum_{0}^{\infty} -1^{n} q^{n(n+1)} \cos(2nz)$$

16·29. ELLRC(X,Y)[5] degenerate elliptic integral

ELLRC(X,Y) computes a scalar which is the value of the degenerate elliptic integral for the inputs X and Y. X and Y must be scalars. X must be non-negative. Y must be positive. Otherwise, a warning message is produced and zero is returned.

The degenerate elliptic integral is defined to be

$$\text{ELLRC}(X,Y) = \int_0^{\infty} \frac{dt}{2(t+X)^{1/2}(t+Y)}$$

Let x, y, and z be positive numbers, where $z = (x \cdot y)^{1/2}$. Then

$\text{ELLRC}(x, x+z) + \text{ELLRC}(y, y+z) = \text{ELLRC}(0, z)$

Special choices of arguments permit the calculation of many special functions, as listed below.

$\log(x) = (x-1)\text{ELLRC}(((1+x)/2)^2, x)$	$x > 0$
$\text{asin}(x) = x\text{ELLRC}(1-x^2, 1)$	$-1 \le x \le 1$
$\text{acos}(x) = (1-x^2)1/2\text{ELLRC}(x^2, 1)$	$0 \le x \le 1$
$\text{atan}(x) = x\text{ELLRC}(1, 1+x^2)$	$-\infty < x < \infty$
$\text{acotan}(x) = \text{ELLRC}(x^2, x^2+1)$	$0 \le x < \infty$
$\text{asinh}(x) = x\text{ELLRC}(1+x^2, 1)$	$-\infty < x < \infty$
$\text{acosh}(x) = (x^2-1)^{1/2}\text{ELLRC}(x^2, 1)$	$x \ge 1$
$\text{atanh}(x) = x\text{ELLRC}(1, 1-x^2)$	$-1 \le x \le 1$
$\text{acoth}(x) = \text{ELLRC}(x^2, x^2-1)$	$x > 1.$

Examples: ELLRC(0,1/4) = π; ELLRC(9/4,2) = log(2)

16·30. ELLRD(X,Y,Z)[5] elliptic integral of the second kind

ELLRD(X,Y,Z) computes a scalar which is the value of the elliptic integral of the second kind for the inputs X, Y, and Z. X, Y, and Z must be scalars. X and Y must be non-negative; X+Y and Z

must be positive. Otherwise, a warning message is produced and zero is returned. If X or Y is zero, the integral is complete; otherwise it is incomplete.

The elliptic integral of the second kind is defined to be

$$\mathrm{ELLRD}(X,Y,Z) = \int_0^\infty \frac{(3/2)dt}{(t+X)^{1/2}(t+Y)^{1/2}(t+Z)^{3/2}}$$

Let x, y, and z be positive numbers. Then

$\mathrm{ELLRD}(x,y,z) + \mathrm{ELLRD}(y,z,x) + \mathrm{ELLRD}(z,x,y) = 3/(x+y+z)^{1/2}$

Special choices of arguments permit the calculation of several standard elliptic integrals using ELLRD and ELLRF, as given below.

The Legendre form of the incomplete elliptic integral of the second kind, where x is the limit of integration, and y is the square of the modulus, is defined to be

$E(x,y) = \sin(x)\mathrm{ELLRF}(\cos(x)^2, 1-y\sin(x)^2, 1) - (y/3)\sin(x)^3\mathrm{ELLRD}(\cos(x)^2, 1-y\sin(x)^2, 1).$

The Legendre form of the complete elliptic integral of the second kind, where x is the square of the modulus, is defined to be

$E(x) = \mathrm{ELLRF}(0, 1-x, 1) - (x/3)\mathrm{ELLRD}(0, 1-x, 1).$

The Bullirsch form of the incomplete elliptic integral of the second kind, where x is the limit of integration, y is the square of the complementary modulus, and a and b are the parameters, is defined to be

$\mathrm{EL2}(x,y,a,b) = ax\mathrm{ELLRF}(1, 1+yx^2, 1+x^2) + (1/3)(b-a)x^3\mathrm{ELLRD}(1, 1+y.x^2, 1+x^2).$

The Legendre form of the composite elliptic integral of the second kind, where x is the limit of integration, and y is the square of the modulus, is defined by

$$\mathrm{LED}(x,y) = \int_0^x \frac{\sin(t)^2 dt}{(1-y\sin(t)^2)^{1/2}}$$

Note that $\mathrm{LED}(x,y) = (1/3)\sin(x)^3\mathrm{ELLRD}(\cos(x)^2, 1-y\sin(x)^2, 1)$

The lemniscate constant, is defined as

$$(1/3)\mathrm{ELLRD}(0,2,1) = \int_0^1 \frac{t^2 dt}{(1-t^4)^{1/2}}$$

Heuman's lambda function, $\Lambda_0(a,b)$, where a and b are the parameters, is defined by

$(\pi/2)\Lambda_0(a,b) = \sin(b)(\mathrm{ELLRF}(0, \cos(a)^2, 1) -$
$(1/3)\sin(a)^2\mathrm{ELLRD}(0, \cos(a)^2, 1)\mathrm{ELLRF}(\cos(b)^2, 1-\cos(a)^2, \sin(b)^2) -$
$(1/3)\cos(a)^2\sin(b)^3\mathrm{ELLRF}(0, \cos(a)^2, 1)\mathrm{ELLRD}(\cos(b)^2, 1-\cos(a)^2\sin(b)^2, 1).$

The Jacobi zeta function, $\zeta(x,y)$, where x is the angle and y is the square of the modulus, is defined by

$\zeta(x,y) = (y/3)\sin(x)\mathrm{ELLRF}(\cos(x)^2, 1-y\sin(x)^2, 1)$
$\mathrm{ELLRD}(0, 1-y, 1)/\mathrm{ELLRF}(0, 1-y, 1) - (y/3)\sin(x)^3\mathrm{ELLRD}(\cos(x)^2, 1-y\sin(x)^2, 1)$

16·31. ELLRF(X,Y,Z)[5] elliptic integral of the first kind

ELLRF(X,Y,Z) computes a scalar which is the value of the elliptic integral of the first kind for the parameter values X, Y, and Z. X, Y, and Z must be scalars. X, Y, and Z must be non-negative and

at most one of them may be zero. Otherwise, a warning message is produced and zero is returned. If one of them is zero, the integral is complete; otherwise it is incomplete.

The elliptic integral of the first kind is defined to be

$$\mathrm{ELLRF}(X,Y,Z) = \int_0^\infty \frac{(1/2)dt}{(t+X)^{1/2}(t+Y)^{1/2}(t+Z)^{1/2}}.$$

Let x, y, z, and w be positive numbers, such that $xy = zw$.

Then $\mathrm{ELLRF}(x, x+z, x+w) + \mathrm{ELLRF}(y, y+z, y+w) = \mathrm{ELLRF}(0, z, w)$.

Special choices of arguments permit the calculation of standard elliptic integrals of the first kind using ELLRF, as given below.

The Legendre form of the incomplete elliptic integral of the first kind, where x is the limit of integration, and y is the square of the modulus, is defined by

$F(x,y) = \sin(x)\mathrm{ELLRF}(\cos(x)^2, 1-y\sin(x)^2, 1)$.

The Legendre form of the complete elliptic integral of the first kind, where x is the square of the modulus, is defined by

$E(x) = \mathrm{ELLRF}(0, 1-x, 1)$.

The Bullirsch form of the incomplete elliptic integral of the first kind, where x is the limit of integration and y is the square of the complementary modulus, is defined to be

$\mathrm{el1}(x,y) = x\mathrm{ellrf}(1, 1+yx^2, 1+x^2)$.

The lemniscate constant, is defined as $a = \int_0^1 \frac{dt}{(1-t^4)^{1/2}}$.

Note: a = ELLRF(0,1,2) = ELLRF(0,2,1).

16·32. ELLRJ(X,Y,Z,P)[5] elliptic integral of the third kind

ELLRJ(X,Y,Z,P) computes a scalar which is the value of the elliptic integral of the third kind for the parameter values X, Y, Z, and P. X, Y, Z, and P must be scalars. X, Y, and Z must be non-negative and at most one of them may be zero. P must be positive. Otherwise, a warning message is produced and zero is returned. If one of X, Y, and Z is zero, the integral is complete; otherwise it is incomplete.

The elliptic integral of the first kind is defined to be

$$\mathrm{ELLRJ}(X,Y,Z,P) = \int_0^\infty \frac{(3/2)dt}{(t+X)^{1/2}(t+Y)^{1/2}(t+Z)^{1/2}(t+P)}$$

Let x, y, z, w, and p be positive numbers, such that $xy = zw$. Let $a = p^2(x+y+z+w)$, and $b = p(p+x)(p+y)$, so that $b - a = p(p-z)(p-w)$.

Then

$\mathrm{ELLRJ}(x, x+z, x+w, x+p) + \mathrm{ELLRJ}(y, y+z, y+w, y+p) + (a-b)\mathrm{ELLRJ}(a,b,b,a) + 3a^{-1/2} = \mathrm{ELLRJ}(0, z, w, p)$.

Also $(a-b)\mathrm{ELLRJ}(a,b,b,a) + 3a^{-1/2} = 3\mathrm{ELLRC}(a,b)$

Special choices of arguments permit the calculation of standard elliptic integrals of the third kind using ELLRF and ELLRJ, as given below.

The Legendre form of the incomplete elliptic integral of the 3rd kind, where x is the limit of integration, and y is the square of the modulus, is defined by

$$\Pi(X,Y,Z) = \int_0^X \frac{dt}{(1+Z\sin(t)^2)(1-Y\sin(t)^2)^{1/2}}$$

Note: $\Pi(x,y,z) = \sin(x)\text{ELLRF}(\cos(x)^2, 1-y\sin(x)^2, 1) -$
$(z/3)\sin(x)^3\text{ELLRJ}(\cos(x)^2, 1-y\sin(x)^2, 1, 1+z\sin(x)^2)$

The Bullirsch form of the incomplete elliptic integral of the third kind, where y is the square of the complementary modulus, is defined by

$\text{el3}(x,y,z) = x\text{ELLRF}(1, 1+yx^2, 1+x^2) + (1/3)(1-z)x^3\text{ellrj}(1, 1+yx^2, 1+zx^2)$.

The Bullirsch form of the complete elliptic integral of the third kind, where x is the square of the complementary modulus, is defined by

$\text{cel}(x,p,a,b) = a\text{ELLRF}(0,x,1) + (1/3)(b-pa)\text{ELLRJ}(0,x,1,p)$.

Heuman's Lambda function $\Lambda(a,b,p)$ is given by

$\Lambda(a,b,p) = (\cos(a)^2\sin(b)\cos(b)(1-\cos(a)^2\sin(b)^2)^{-1/2}(\sin(p)\text{ELLRF}(\cos(p)^2, 1-\sin(a)^2\sin(p)^2, 1)+$
$(\sin(a)^2\sin(p)^3/(3(1-\cos(a)^2\sin(b)^2)))\text{ELLRJ}(\cos(p)^2, 1-\sin(a)^2\sin(p)^2, 1, 1-\sin(a)^2\sin(p)^2/(1-$
$\cos(a)^2\sin(b)^2))))$.

Heuman's Lambda function $\Lambda_0(a,b)$ is given by

$(\pi/2)\Lambda_0(a,b) = \Lambda(a,b,\pi/2) = \cos(a)^2\sin(b)\cos(b)(1-\cos(a)^2\sin(b)^2) -$
$1/2\text{ELLRF}(0, \cos(a)^2, 1) + (1/3)\sin(a)^2\cos(a)^2\sin(b)\cos(b)(1-\cos(a)^2\sin(b)^2))^{-3/2}$
$\text{ELLRJ}(0, \cos(a)^2, 1, \cos(a)^2\cos(b)^2/(1-\cos(a)^2\sin(b)^2)))$.

The Jacobi zeta function, $\zeta(x,y)$, where x is the angle and y is the square of the modulus, is defined as

$\zeta(x,y) = (y/3)\sin(x)\cos(x)(1-y\sin(x)^2)^{1/2}\text{ELLRJ}(0, 1-y, 1, 1-y\sin(x)^2)/\text{ELLRF}(0, 1-y, 1)$.

17. Statistical Distribution Functions

17·1. GAUSSF(X[,Y[,Z]])[8] normal distribution function

GAUSSF(X[,Y[,Z]]) computes a scalar which is the value corresponding to X of the (mean Y, variance Z) normal distribution function. X, Y, and Z must be scalars. If Y is omitted, it defaults to zero. If Z is omitted, it defaults to one. If $Z \le 0$, zero is returned and a warning message is given. In terms of the standard normal distribution function ϕ, $\text{GAUSSF}(X,Y,Z) = \phi((X-Y)/\sqrt{Z})$.

Recall the definition: $\phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{X} e^{\frac{-t^2}{2}}dt = \Pr(N(0,1) \le x)$, where N(0,1) is a normal random variable with mean 0 and variance 1.

17·2. CHISQF(X,Y[,Z])[8] non-central chi-squared distribution function

CHISQF(X,Y[,Z]) computes a scalar which is the value corresponding to X of the non-central chi-squared (χ^2) distribution function with Y degrees of freedom and optional non-centrality parameter Z. X, Y, and Z must be scalars. Y must be positive. Z, if supplied, must be non-negative; if Z is omitted, it defaults to zero. Otherwise, zero is returned and a warning message is produced.

For the central chi-squared distribution, let $N_1, N_2, \ldots, N_Y$ be independent, $N(0,1)$ (zero mean, unit variance, normal) random variables. Then, $\chi^2_Y = N_1^2 + N_2^2 + \ldots + N_Y^2$ chi-squared random variable with Y degrees of freedom.

Also, if $X_1, X_2, \ldots, X_n$ are independent $N(\mu, \sigma^2)$ random variables, then the distribution of the sample variance S^2 is simply related to the chi-squared distribution, namely:

$$S^2 = \frac{1}{(n-1)} \sum_{j=1}^{n} (X_j - \overline{X})^2 \sim \sigma^2 \chi^2_{n-1}/(n-1).$$

Recall the relation to the incomplete Gamma function, also available in MLAB:

$$\text{CHISQF}(X, Y) = \text{IGAMMA}(Y/2, X/2).$$

For the non-central chi-squared distribution, let $U_1 \sim N(d_1, 1), U2 \sim N(d_2, 1), \ldots, U_y \sim N(d_y, 1)$ be independent normal random variables, with means $d_1, d_2, \ldots, d_Y$ and unit variance. Let $Z = d_1^2 + d_2^2 + \ldots + d_Y^2$. Then, $\chi^2_Y(Z) = U_1^2 + U_2^2 + \ldots + U_y^2$ is a non-central chi-squared random variable with Y degrees of freedom and non-centrality parameter Z. CHISQF(X,Y,Z) is defined as $\Pr[\chi^2_Y(Z) \leq X]$, the probability that a sample value of χ^2 with Y degrees of freedom and non-centrality parameter Z will be less than or equal to X. In statistical applications, Y is ordinarily an integer related to sample size.

17·3. STUTF(X,Y[,Z])[8] non-central Student's t distribution function

STUTF(X,Y[,Z]) computes a scalar which is the value corresponding to X of the distribution function for the random variable $t_Y(Z)$, which is Student's statistic t with Y degrees of freedom and non-centrality parameter Z. X, Y, and Z must be scalars. Y, which represents the degrees of freedom, must be a positive value. The optional parameter Z, which represents the noncentrality parameter, must be non-negative; if omitted, Z defaults to zero. Otherwise, zero is returned, and a warning message is produced. $\text{STUTF}(X, Y, Z) = \Pr[t_Y(Z) \leq X]$.

For the central Student's t distribution, where $Z = 0$, let $N_0, N_1, N_2, \ldots, N_y$ be independent, $N(0, 1)$ (zero mean, unit variance, normal) random variables. Then, a Student's t random variable with Y degrees of freedom may be constructed as

$$t_Y = N_0 / \sqrt{\frac{N_1^2 + N_2^2 + \ldots + N_Y^2}{Y}}.$$

Also, if $X_1, X_2, \ldots, X_n$ are independent $N(\mu, \sigma^2)$ random variables, and

$$S^2 = \frac{1}{(n-1)\sum_{j=1}^{n}(X_j - \overline{X})^2}, \text{ with } \overline{X} = (X_1 + X_2 + \ldots, +X_n)/n, \text{ then}$$

$\sqrt{n}(\overline{X} - \mu)/\sigma \sim N(0, 1)$, and $\sqrt{n}(\overline{X} - \mu)/S \sim t_{n-1}$.

In statistical applications, the value of the input argument X is often a sample of the "studentized" difference in the means of two data sets from normally-distributed populations, and the input argument Y is an integer related to sample size.

Recall the definition of the central Student's t distribution function in terms of the complete Beta function B:

$$\Pr[t_Y \leq X] = \frac{1}{\sqrt{Y} B(.5, Y/2)} \int_{-\infty}^{X} (1 + u^2/Y)^{-(Y+1)/2} du$$

The non-central Student's t distribution with Y degrees of freedom and non-centrality parameter Z is the distribution of the random variable $t_Y(Z) = (N(0, 1) + Z)/(\chi^2_Y/Y)^{1/2}$, where χ^2_Y is a central chi-squared random variable with Y degrees of freedom.

17·4. ASTUTF(X,Y[,Z])[8] absolute non-central Student's t distribution function

ASTUTF(X,Y[,Z]) computes a scalar which is the value corresponding to X of the distribution function for the random variable $|t_Y(Z)|$, which is the absolute value of a non-central Student's t random variable with Y degrees of freedom, and non-centrality parameter Z. X, Y, and Z must be scalars. Y, which represents the degrees of freedom, must be a positive value. The optional argument Z, if present, must be non-negative. If omitted, Z defaults to zero. If $Y \le 0$, or $Z < 0$, then zero is returned, and a warning message is produced. If $X \le 0$, the result is identically zero. $\text{ASTUTF}(X, Y, Z) = \Pr[|t_Y(Z)| \le X]$

Recall the definition for the central case, where $Z = 0$:

$$\Pr[|t_Y| \le s] = \Pr[-s \le t_Y \le s] = \frac{1}{\sqrt{Y}B(0.5, Y/2)} \int_{-s}^{s} (1 + X^2/Y)^{-(Y+1)/2} dx, \text{ for } s \ge 0.$$

where $B(x, y)$ is the complete beta function. Also, recall the relation:

$\Pr[|t_n| \le s] = 1 - I_{\nu/(\nu+s^2)}(\nu/2, 1/2)$, where I_x is the incomplete beta function.

17·5. QFF(X,Y,Z[,V])[8] singly non-central F-distribution function

QFF(X,Y,Z[,V]) computes a scalar which is the value corresponding to X of the singly non-central F distribution function with (Y,Z) degrees of freedom, and non-centrality parameter V. X, Y, Z and V must be scalars. $\text{QFF}(X, Y, Z, V) = \Pr[F_{YZV} \le X]$.

Y and Z must be positive and V must be non-negative. If omitted, V defaults to zero, so that QFF computes the central F-distribution. If $Y \le 0$, $Z \le 0$, or $V < 0$, then zero is returned, and a warning message is produced.

Let $F_{YZV} = (c_1/Y)/(c_2/Z)$, where c_1 and c_2 are independent random variables such that c_1 has a non-central chi-squared distribution with Y degrees of freedom and non-centrality parameter V, and c_2 has a non-central chi-squared distribution with Z degrees of freedom and non-centrality parameter V. Then, the random variable F_{YZV} has a singly non-central F-distribution with Y and Z degrees of freedom and non-centrality parameter V.

The definition of the central F distribution function with (Y,Z) degrees of freedom (and non-centrality parameter 0) is:

$$\Pr[F_{yz0} \le x] = I_t(Y/2, Z/2), \text{ where } t = YX/(Z + YX)$$

where $I_x(y, z)$ is the incomplete beta function, available in MLAB as IBETA(Y,Z,X).

The definition of the singly non-central F-distribution function with Y and Z degrees of freedom and non-centrality parameter V is:

$$\Pr[F_{YZV} \le X] = e^{-V/2} \sum_{j=0}^{\infty} [(V/2)^j/j!] I_g(Y/2 + j, Z), \text{ where } g = XY/(Z + XY).$$

In statistical applications, the random variable F_{YZ0} is the ratio of the sample variances of two sets of independent identically-distributed normal random variables, where Y and Z are integers related to the sizes of the two sets of samples.

17·6. KSMF(X)[8] Kolmogorov-Smirnov distribution function

KSMF(X) computes a scalar which is the value corresponding to X of the Kolmogorov-Smirnov maximum-deviation distribution function. KSF(X) = Pr[$sD \le X$], which lies in [0,1]. X must be a scalar.

The K-S distribution is the distribution of the statistic sD, where $D = \max(K^+, K^-)$, with $K^+ = \sup(c(t)-f(t))$ and $K^- = -\inf(c(t)-f(t))$. The functions $c(t)$ and $f(t)$ are two specified distribution functions, where c is an empirical distribution (i.e. a non-decreasing step-function based on n sample values) and f is either a distribution function which is not based on sample data, in which case $s = \sqrt{n}$, or f is also an empirical distribution function based on m sample values, in which case $s = \sqrt{nm/(n+m)}$.

The K-S distribution is used in the K-S one- and two-sample tests.

17·7. BINOMF(X[,Y[,Z]])[8] binomial distribution function

BINOMF(X[,Y[,Z]]) computes a scalar which is the value corresponding to X of the binomial distribution function, Pr[$S_{YZ} \le X$], which is the probability of an event with probability Z occuring $\lfloor X \rfloor$ or fewer times in Y independent trials. S_{YZ} is a binomially-distributed random variable with number-of-trials parameter Y and success probability parameter Z. X, Y, and Z must be scalars.

The optional parameter Y, if supplied, must be a positive integer. If omitted, it defaults to one. The optional parameter Z, if supplied, must be a number in the interval [0,1]. If omitted, it defaults to 0.5. The result is a scalar which is the probability of $\lfloor X \rfloor$ or fewer successes in Y Bernoulli trials with success probability Z. If $Y < 0$ or $Z \notin [0,1]$, zero is returned and a warning message is produced. Recall the relation of the binomial distribution function to the incomplete beta function, which is available in MLAB as IBETA:

$$\sum_{j=0}^{k} \binom{n}{j} p^j (1-p)^{n-j} = 1 - IBETA(p, k, n-k+1).$$

The following relationship with the F-distribution function is sometimes useful

$$\text{BINOMF}(S, N, P) = \text{QFF}((S+1)(1-P)/(P(N-S)), 2(N-S), 2(S+1)).$$

17·8. POISSF(X[,Y])[8] Poisson distribution function

POISSF(X[,Y]) computes a scalar which is the value corresponding to X of the Poisson distribution function. POISSF(X, Y) = Pr[$S_Y \le X$], the probability that a mean Y Poisson random variable S_Y is no greater than X. X and Y must be scalars.

The mean $E(S_Y) = Y$. Y must be non-negative. If Y is omitted, it defaults to one. If $Y < 0$, zero is returned, and a warning message is produced.

The Poisson density function is $\Pr[S_Y = k] = e^{-Y}Y^k/k!$, for $k = 0, 1, 2, \ldots$. The Poisson distribution function is: $\Pr[S_Y \le X] = \sum_{k=0}^{X} e^{-Y}Y^k/k!$.

The Poisson distribution is the limit as $N \to \infty$ and $p \to 0$ such that $Np = Y$, of the binomial distribution $\sum_{k=0}^{X} \binom{N}{k} p^k (1-p)^{N-k}$.

Recall the relation: POISSF(X, Y) = $1 - \Gamma_p(Y, X)$, where Γ_p is the incomplete gamma function, and the special cases: POISSF($0, Y$) = e^{-Y}, and POISSF(∞, Y) = 1.

17·9. SPEARF(X[,Y])[8] Spearman rank-correlation distribution function

SPEARF(X,Y) computes a scalar which is the value corresponding to the rank correlation value X of the Spearman rank-correlation distribution with sample size parameter Y. X and Y must be scalars. If Y is omitted, it defaults to 1. The result is a probability, which lies in [0,1].

Let $A_1, A_2, \ldots, A_n$ be identically-distributed random variables, and let $B_1, B_2, \ldots, B_n$ be identically distributed random variables paired with $A_1, A_2, \ldots, A_n$. Assign ranks between 1 and n to the values $\{A_i\}$ and similarly to the values $\{B_i\}$, with ties within each group assigned the mean of their integral ranks. Let d_i be the difference between the ranks of A_i and B_i. Then, the Spearman rank-correlation random variable with sample size parameter n is

$$r_n = 1 - 6 \sum_{i=1}^{n} \frac{d_i^2}{n(n^2-1)}.$$

SPEARF$(X, Y) = \Pr[r_Y \le X]$. Ordinarily X lies in [-1,1]. If $X \le -1$, zero is returned. If $X \ge 1$ then one is returned. Y must be a positive integer. If Y is not a positive integer, zero is returned, and a warning message is produced.

The Spearman rank-correlation distribution is used in the Spearman rank-correlation test. The discrete range of the Spearman rank-correlation random variable r_Y for Y samples is $\{-1 : 1 : 3/(Y(Y^2-1)+1)\}$, i.e. $1 + Y(Y^2-1)/6$ equally-spaced values in $[-1, 1]$.

For $Y \le 10$, SPEARF(X,Y) is computed exactly by direct enumeration. For $Y > 10$, direct enumeration is prohibitively lengthy, so an asymptotic approximation using the Student's t distribution is employed.

17·10. WILCF(X,Y)[8] Wilcoxon signed-rank distribution function

WILCF(X,Y) computes a scalar which is the value corresponding to the signed-rank value X of the Wilcoxon signed-rank distribution with sample size parameter Y. X and Y must be scalars. The result is a probability, which lies in [0,1].

Let $A_1, A_2, \ldots, A_n$ be identically-distributed random variables with the median value M, and let $B_1, B_2, \ldots, B_n$ be identically-distributed random variables, also with the median value M. A_i and B_i are assumed to be matched so that their respective correlations among their cohort members are identical, i.e. $\mathrm{corr}(A_i, A_j) = \mathrm{corr}(B_i, B_j)$, for $1 \le i, j \le n$.

The following procedure defines the Wilcoxon signed-rank random variables t^+ and t^-, with effective sample size Y.

1. Compute the absolute differences $|A_1 - B_1|, |A_2 - B_2|, ..., |A_n - B_n|$.

2. Count the number of absolute differences which are zero, and subtract that value from N to obtain the effective sample size Y. Also, discard the zero differences to obtain the list of non-zero absolute differences $D_1, D_2, \ldots, D_y$.

3. Compute the ranks $R_1, R_2, \ldots, R_y$ of the values $D_1, D_2, \ldots, D_y$, by assigning the rank values between 1 and Y to the sorted D_i values, with ties assigned the mean of their integral ranks.

4. Assign the algebraic sign of the original non-zero difference associated with each rank value R_i to that rank value, so that each R_i value is now either positive or negative.

5. Define t^+ to be the sum of the positive rank values, and define t^- to be the absolute value of the sum of the negative rank values. Both t^+ and t^- have the same distribution as the Wilcoxon signed-rank random variable W_Y with effective sample size parameter Y. t^+ and t^- are linearly related: $t^+ + t^- = Y(Y+1)/2$.

WILCF$(X, Y) = \Pr[W_Y \le X]$. The range of the Wilcoxon signed-rank random variable W_Y is an integer in $[0, Y(Y+1)/2]$; however, the distribution function is defined for all X. If $X < 0$, then zero

is returned. If $X > Y(Y+1)/2$, then one is returned. Y must be a positive integer. If Y is not a positive integer, then zero is returned and a warning message is produced.

The Wilcoxon signed-rank distribution function is used in the Wilcoxon signed-rank test.

For $Y \leq 15$, WILCF(X,Y) is computed exactly by direct enumeration. For $Y > 15$, direct enumeration is prohibitively lengthy, so a continuity-corrected asymptotic approximation using the normal distribution with mean $Y(Y+1)/4$ and variance $Y(Y+1)(2Y+1)/24$ is used.

17·11. MWF(X,Y,Z)[8] Mann-Whitney-Wilcoxon rank-sum distribution function

MWF(X,Y,Z) computes a scalar which is the value corresponding to the lesser group-size rank-sum value, $\lfloor X \rfloor$, of the Mann-Whitney-Wilcoxon rank-sum distribution with sample sizes Y and Z. X, Y and Z must be scalars. Y and Z must be positive integers. If Y and Z are not positive integers, then zero is returned and a warning message is produced. The result is a probability, which lies in [0,1].

Let $A_1, \ldots, A_y$ be independent, identically distributed random variables, and let $B_1, \ldots, B_z$ be independent, identically distributed random variables. Let $n = y + z$, and let $m = \min(y, z)$. Assign ranks to the n pooled values, with ties assigned the means of their integral ranks. Let w_{yz} be the minimum of the sum of the ranks assigned to $B_1, \ldots, B_z$ and the sum of the ranks assigned to $A_1, \ldots, A_y$. w_{yz} is the Mann-Whitney-Wilcoxon rank-sum random variable with parameters y and z. $\mathrm{MWF}(X, Y, Z) = \Pr[w_{YZ} \leq X]$.

A value of the Mann-Whitney-Wilcoxon rank-sum random variable w_{yz} can only be an integer in the range $[m(m+1)/2, m(2n-m+1)/2]$; however, the distribution function is defined for all X. If $X < m(m+1)/2$, then zero is returned. If $X > m(2n-m+1)/2$, then one is returned.

The Mann-Whitney-Wilcoxon rank-sum distribution function is used in the Mann-Whitney-Wilcoxon rank-sum test.

For $\min(Y, Z) \leq 15$, MWF(X,Y,Z) is computed exactly by direct enumeration. For $\min(Y, Z) > 15$, direct enumeration is prohibitively lengthy, so a continuity-corrected asymptotic approximation using the normal distribution with mean $m(n+1)/2$ and variance $(m(n+1)(n-m)/12$ is used.

17·12. GAMMAF(X[,Q[,R[,S]]])[8] gamma distribution function

GAMMAF(X,Q,R,S) computes a scalar which is the value of the gamma distribution function with argument X, shape parameter Q, scaling parameter R, and shift parameter S. X, Q, R, and S must be scalars. Q, R, and S are all optional. If Q is omitted, it defaults to 1. If R is omitted, it defaults to 1. If S is omitted, it defaults to 0. The constraints $Q > 0$ and $R > 0$ must hold. Note, if $X \leq S$, the result is zero. When $R = 1$ and $S = 0$, the expected value and variance of a gamma distributed random variable are both the value Q. In general the mean is $RQ + S$ and the variance is QR^2.

The definition of the gamma distribution function is:

$$\mathrm{GAMMAF}(X, Q, R, S) = \frac{1}{\Gamma(Q)} \int_0^{\frac{X-S}{R}} t^{Q-1} e^{-t} dt.$$

Recall the relation to the incomplete gamma function (denoted $\mathrm{IGAMMA}(X, Q)$ in MLAB): $\mathrm{GAMMAF}(X, Q, R, S) = \mathrm{IGAMMA}(Q, (X-S)/R)$.

17·13. BETAF(X[,A[,B[,C[,D[,E]]]]])[8] non-central beta distribution function

BETAF(X[,A[,B[,C[,D[,E]]]]]) computes a scalar which is the value of the non-central beta distribution function with argument X, shape parameters A and B, non-centrality parameter C, and range parameters D and E. X, A, B, C, D, and E must be scalars. The parameters A,B,C,D, and E are

optional. If A and/or B are omitted, they default to $A = 1$ and $B = 1$. If D and/or E are omitted, they default to D = 0 and E = 1. The constraints $A > 0$, $B > 0$, and $E > D$ must hold. If the optional argument C is omitted, it defaults to zero. In statistical applications, the non-centrality parameter C is non-negative.

If $X \leq D$, the result is zero. If $X \geq E$, the result is one.

The definition of the central beta distribution function in terms of the complete beta function β is:

$$\text{BETAF}(X, A, B, 0, D, E) = \frac{1}{(E-D)^{A+B+1}\beta(A,B)} \int_D^X (t-D)^{A-1}(E-t)^{B-1}dt.$$

The central beta distribution function is related to the incomplete beta function (denoted IBETA(X, A, B) in MLAB) such that: IBETA(X, A, B) = BETAF$(X, A, B, 0, 0, 1)$.

The function BETAF(X,.5,.5,0,0,1) is known as the arcsine distribution function. The function BETAF(X,A,1-A,0,0,1) is sometimes called the generalized arcsine distribution function.

The non-central beta distribution function is defined in terms of the central form as:

$$\text{BETAF}(X, A, B, C, D, E) = e^{-\frac{C}{2}} \sum_{k=0}^{\infty} \frac{(\frac{C}{2})^k}{k!} \text{BETAF}(X, A+k, B, 0, D, E).$$

17·14. WEIBF(X[,Y[,Z[,V]]])[8] Weibull distribution function

WEIBF(X[,Y[,Z[,V]]]) computes a scalar which is the value corresponding to the input X of the Weibull distribution function with width parameter value Y, slope parameter value Z, and displacement parameter value V. X, Y, Z, and V must be scalars. The result is a probability, which lies in [0,1].

The optional width parameter Y must be non-negative. If omitted, Y defaults to one. The optional slope parameter value Z must be positive. If omitted, Z defaults to one. The displacement parameter V is optional. If V is omitted, it defaults to zero.

Let W be a Weibull (Y,Z,V) random variable. The definition of the Weibull distribution function is:

$$\Pr[W \leq X] = 1 - \exp\{-[(X-V)/Y]Z\} \text{for } X \geq V, \text{ and 0 otherwise.}$$

17·15. NBINF(X[,Y[,Z]])[8] negative binomial distribution function

NBINF(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input X of the negative binomial distribution function with number-of-successes parameter value Y and probability-of-success parameter value Z. X, Y, and Z must be scalars. The result is a probability which lies in [0,1].

The optional number-of-successes parameter Y must be a non-negative integer. If Y is omitted, it defaults to 1. The optional probability-of-success parameter Z must lie in [0,1]. If omitted, it defaults to 0.5. If $Y < 0$ or $Z \notin [0, 1]$, 0 is returned and a warning message is produced.

The negative binomial distribution is the distribution of the random variable F = the number of trials before Y successes in an indefinitely-long sequence of Bernoulli trials with probability of success Z on each trial. The negative binomial distribution function is piecewise constant, and has jumps at the integer values 0,1,... .

Recall the definition of the negative binomial distribution function:

$$\Pr[F \le X] = \sum_{j=Y}^{X} \binom{j-1}{Y-1} Z^Y (1-Z)^{j-Y}.$$

The negative binomial distribution may be expressed in terms of the binomial distribution:
NBINF$(X, Y, Z) = 1 -$ BINOMF$(Y, X + Y, 1/(1 - Z))$.

For integer argument values, the negative binomial distribution may also be expressed in terms of the incomplete beta function:

NBINF$(X, Y, Z) = 1 -$ BETAI$(1/(1 - Z), Y, X + 1)$.

17·16. EXPF(X[,Y[,Z]])[8] exponential distribution function

EXPF(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input X of the exponential distribution function with shifted mean parameter Y and displacement parameter Z. X, Y, and Z must be scalars. The result is a probability, which lies in [0,1].

By convention the exponential distribution is zero for $X \le Z$. Y is the value of the optional shifted mean parameter. Y must be non-negative. If omitted, Y defaults to one. If $Y < 0$, then zero is returned and a warning message is produced. Z is the value of the optional displacement parameter. If omitted, Z defaults to zero.

Let V be a random variable with the exponential distribution function with shifted mean parameter Y and displacement parameter Z. Then

$$\Pr[V \le X] = \int_Z^X \frac{e^{-\frac{u-Z}{Y}}}{Y} du = 1 - e^{\frac{X-Z}{Y}}.$$

Note: $E(X) = Y - Z$.

17·17. GEOMF(X[,Y])[8] geometric distribution function

GEOMF(X[,Y]) computes a scalar which is the value corresponding to the input X of the geometric distribution function with probability of success parameter value Y. X and Y must be scalars. The optional probability of success parameter Y must lie in [0,1]. If Y is omitted, it defaults to 0.5. If $Y \notin [0, 1]$, then zero is returned and a warning message is produced.

The geometric distribution is the distribution of the random variable G = the number of failures occurring before the first success in an indefinitely-long sequence of Bernoulli trials with probability of success Y. The geometric distribution function is piecewise constant with jumps at the integer values 0, 1, ...

Recall the definition of the geometric distribution function:

$$\Pr[G \le X] = 1 - (1 - Y)^{X+1}, \text{ for } X \ge 0, \text{ and 0 otherwise.}$$

17·18. HGEOMF(X,Y,Z,V[,U])[8] hypergeometric distribution function

HGEOMF(X,Y,Z,V[,U]) computes a scalar which is the value corresponding to the input argument value X of the hypergeometric distribution function. X, Y, Z, and V must be non-negative integer-valued scalars. If the optional non-centrality parameter U is omitted, it defaults to 1, corresponding to the central hypergeometric distribution function. If supplied, U must be a non-negative scalar.

If the arguments are not non-negative integers, zero is returned and a warning message is produced. The result is a probability which lies in [0,1].

The central hypergeometric distribution is the distribution of the random variable N = number of defectives found when a sample of size Y is drawn without replacement from a lot of size Z actually containing V defectives.

The definition of the central hypergeometric distribution function is

$$\Pr[N \le X] = \sum_{j=0}^{X} \frac{\binom{Z}{j}\binom{V-Z}{Y-j}}{\binom{V}{Y}}.$$

The hypergeometric distribution is zero for $X < \max(0, Y - V + Z)$, and one for $X \ge \min(Y, V)$. Note that HGEOMF(X,Y,Z,V,0) = HGEOMF(V,Y,Z,X,0).

17·19. LOGISF(X[,Y[,Z]])[8] logistic distribution function

LOGISF(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input argument X of the logistic distribution function with mean Y and slope Z. X, Y, and Z must be scalars. The result is a probability, which lies in [0,1].

If Y is omitted, it defaults to zero. Z must be positive. If Z is omitted, it defaults to one. If $Z \le 0$, zero is returned, and a warning message is produced.

Let L be a random variable with the logistic distribution function with mean Y and slope Z. Then

$$\Pr[L \le X] = \frac{e^{(X-Y)Z}}{1 + e^{(X-Y)Z}}.$$

17·20. CAUCHYF(X[,Y[,Z]])[8] Cauchy distribution function

CAUCHYF(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input X of the Cauchy distribution function with displacement parameter Y and width parameter Z. X, Y, and Z must be scalars. The result is a probability, which lies in [0,1].

If Y is omitted, it defaults to zero. Z must be non-negative. If Z is omitted it defaults to one. If Z = 0, then, by convention, 0.5 is returned for any values of the other arguments. If $Z < 0$, then zero is returned and a warning message is produced.

Let C be a random variable with the Cauchy distribution function with displacement parameter Y and scale parameter Z. Then

$$\Pr[C \le X] = .5 + \operatorname{atan}((X - Y)/Z)/\pi.$$

17·21. LAPLACEF(X[,Y[,Z]])[8] Laplace distribution function

LAPLACEF(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input X of the Laplace distribution function with displacement parameter Y and width parameter Z. X, Y, and Z must be scalars. The result is a probability, which lies in [0,1].

If the optional parameter Y is omitted, it defaults to zero. The optional parameter Z, if supplied, must be positive. If Z is omitted, it defaults to one. If $Z \le 0$, then zero is returned and a warning message is produced.

Let P be a random variable with the Laplace distribution function with width Z and displacement Y. Then

$$\Pr[P \le X] = \begin{cases} .5 + e^{\frac{X-Y}{Z}} & \text{for } X \le Y \\ 1 - .5e^{\frac{X-Y}{Z}} & \text{for } X \ge Y \end{cases} .$$

17·22. PARETOF(X[,Y[,Z]])[8] Pareto distribution function

PARETOF(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input X of the Pareto distribution function with minimum parameter Y and slope Z. X, Y, and Z must be scalars. The result is a probability which lies in [0,1).

Y must be non-negative. If omitted, Y defaults to one. Z must be non-negative. If omitted, Z defaults to one. If $Y < 0$ or $Z < 0$, then zero is returned and a warning message is produced.

Let P be a random variable with the Pareto distribution function with minimum value Y and shape Z. Then

$$\Pr[P \le X] = 1 - (Y/X)^Z.$$

17·23. UNIF(X[,Y[,Z]])[8] uniform distribution function

UNIF(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input X of the uniform distribution function with lower limit Y and upper limit Z. X, Y, and Z must be scalars. The result is a probability, which lies in [0,1].

If Y is omitted, it defaults to zero. If Z is omitted, it defaults to one. If $Y > Z$, zero is returned and a warning message is produced.

The definition of the uniform distribution function is:

$$\text{UNIF}(X, Y, Z) = \begin{cases} 0 & \text{for } X \le Y \\ (X - Y)/(Z - Y) & \text{for} X \in [Y, Z] \\ 1 & \text{for} X \ge Z \end{cases} .$$

17·24. TRIF(X[,Y[,Z]])[8] triangular distribution function

TRIF(X[,Y],Z]) computes a scalar which is the value corresponding to the input X of the triangular distribution function with lower limit Y and upper limit Z. X, Y, and Z must be scalars. The result is a probability, which lies in [0,1].

If Y is omitted, it defaults to zero. If Z is omitted, it defaults to one. Y must be greater than or equal to Z; otherwise, zero is returned and a warning message is produced.

The triangular distribution is symmetrical about its mean $m = (Y + Z)/2$. The definition of the triangular distribution function is:

$$\text{TRIF}(X, Y, Z) = \begin{cases} 2(\frac{X-Y}{Z-Y})^2 & \text{for } X \in [Y, m] \\ 1 - 2(\frac{X-Z}{Y-Z})^2 & \text{for } X \in [m, Z] \end{cases} .$$

17·25. LNORMF(X[,Y[,Z[,V]]])[8] lognormal distribution function

LNORMF(X[,Y[,Z[,V]]]) computes a scalar which is the value corresponding to the input argument value X of the lognormal distribution function with log-related mean Y, log-related variance Z, and displacement V. X, Y, Z, and V must be scalars. The result is a probability which lies in [0,1)

If Y is omitted it defaults to zero. Z must be positive. If Z is omitted, it defaults to one. If omitted, V defaults to zero. If $Z \leq 0$, then zero is returned and a warning message is produced.

Let $\log(L - V)$ denote a normally distributed random variable with mean Y and variance Z. Then the random variable L has a lognormal distribution with log-related mean Y and log-related variance Z. The lognormal distribution takes on the value zero for $X < V$. The mean of L is $\exp(Y + .5V)$.

The definition of the lognormal distribution function is:

$$\Pr[L \leq X] = \frac{1}{\sqrt{2\pi Z}} \int_{-V}^{X-V} \frac{e^{-\frac{(\log(u)-Y)^2}{2Z}}}{u} du \text{ for } X \geq V, \text{ and 0 otherwise.}$$

17·26. GUMBELF(X[,Y[,Z]])[8] Gumbel extreme value distribution function

GUMBELF(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input argument X of the Gumbel (extreme value) distribution function with displacement Y and width Z. X, Y, and Z must be scalars. The result is a probability which lies in (0,1).

The optional width parameter Z must be positive; if omitted, Z defaults to one. The displacement parameter Y is optional. If omitted, Y defaults to zero. If $Z \leq 0$, then zero is returned and a warning message is produced.

Let G be a random variable with the Gumbel distribution function with scale Y and location Z. Then

$$\Pr[G \leq X] = e^{-e^{-\frac{X-Y}{Z}}}.$$

17·27. HNORMF(X[,Y[,Z]])[8] halfnormal distribution function

HNORMF(X[,Y[,Z]]) computes a scalar which is the value of the halfnormal distribution function with mean Y and variance Z for the input argument X. X, Y, and Z must be scalars. Y and Z are optional. If Y is omitted, it defaults to zero. If Z is omitted, it defaults to one. Z must be greater than zero. If $Z \leq 0$, a warning message is printed and zero is returned.

The halfnormal distribution is the rescaled restriction of the normal distribution to positive values, so that, for $X > 0$, $\text{HNORMF}(X, Y, Z) = 2\text{GAUSSF}(X, Y, Z) - 1$. For $X \leq 0$, $\text{HNORMF}(X, Y, Z) = 0$. Y and Z are the respective mean and variance of the underlying normal distribution.

---

18. Inverse Statistical Distribution Functions

18·1. GAUSSI(X[,Y[,Z]])[8] inverse gaussian distribution function

GAUSSI(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input X of the inverse of the gaussian distribution function. X, Y, and Z must be scalars. X corresponds to a probability value and must lie in [0,1]. The optional argument Y is the mean of the distribution; if omitted, it defaults to zero. The optional argument Z, which must be non-negative, is the variance of the distribution; if omitted it defaults to one. If $X \notin [0, 1]$ or $Z < 0$, zero is returned, and a warning message is produced.

18·2. CHISQI(X,Y,[Z])[8] inverse non-central chi-squared distribution function

CHISQI(X,Y[,Z]) computes a scalar which is the value corresponding to the input X of the inverse of the non-central chi-squared distribution function with Y degrees of freedom, and non-centrality parameter Z. X, Y, and Z must be scalars. X corresponds to a probability, and must lie in [0,1]. Y must be a positive value representing the degrees of freedom. In statistical applications, Y is an integer related to sample size. The optional non-centrality argument Z must be non-negative. If omitted, Z defaults to zero. If X lies outside [0,1], $Y \leq 0$, or $Z < 0$, then zero is returned, and a warning message is produced.

18·3. STUTI(X,Y[,Z])[8] inverse non-central Student's t distribution function

STUTI(X,Y[,Z]) computes a scalar which is the value corresponding to the input X of the inverse of the noncentral Student's t distribution function with Y degrees of freedom and non-centrality parameter Z. X, Y, and Z must be scalars. X corresponds to a probability, and must lie in [0,1]. Y must be a positive value representing the degrees of freedom. In statistical applications, Y is an integer related to sample size. The optional non-centrality argument Z, if omitted, defaults to zero. In statistical applications, Z is non-negative. If X lies outside [0,1], $Y \leq 0$, or $Z < 0$, then zero is returned, and a warning message is produced.

18·4. ASTUTI(X,Y[,Z])[8] inverse non-central absolute Student's t dist. function

ASTUTI(X,Y[,Z]) computes a scalar which is the value corresponding to the input X of the inverse of the absolute non-central Student's t distribution function with Y degrees of freedom and non-centrality parameter Z. X, Y and Z must be scalars. X corresponds to a probability, and must lie in [0,1]. Y must be a positive value representing the degrees of freedom. In statistical applications, Y is an integer related to sample size. The optional non-centrality argument, if omitted, defaults to zero. In statistical applications, Z is non-negative. If X lies outside [0,1], $Y \leq 0$, or $Z < 0$, then zero is returned, and a warning message is produced.

18·5. QFI(X,Y,Z[,V])[8] inverse singly non-central F-distribution function

QFI(X,Y,Z[,V]) computes a scalar which is the value corresponding to the input X of the inverse of the singly non-central F-distribution function, with (Y,Z) degrees of freedom, and non-centrality parameter V. X, Y, Z, and V must be scalars. The result is the value Q such that QFF(Q,Y,Z,V) = X.

X must lie in [0,1]. If X = 1, then MAXPOS is returned. Y and Z must be positive values. In statistical applications, Y and Z are integers related to sample sizes. V must be non-negative. If omitted, V defaults to zero, and QFI reduces to the inverse of the central F-distribution function. If $X \notin [0,1]$, $Y \leq 0$, $Z \leq 0$, or $V < 0$, then zero is returned, and a warning message is produced.

Recall the symmetry relation: $\mathrm{QFI}(X,Y,Z,0) = 1/\mathrm{QFI}(1-X,Z,Y,0)$.

18·6. BINOMI(X[,Y[,Z]])[8] inverse binomial distribution function

BINOMI(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input X of the inverse of the binomial distribution function with Y trials and probability of success Z. X, Y, and Z must be scalars.

X and Z must lie in [0,1]. Y must be non-negative. If Y = 0, then zero is always returned. If Y is omitted, it defaults to one. If X or $Z \notin [0,1]$ or $Y < 0$, then zero is returned and a warning message is produced. If Z is omitted, it defaults to 0.5.

The binomial distribution is discrete, with jumps occuring at the integer points $0, 1, \ldots$. Recall that BINOMF is the binomial distribution function. If BINOMF$(V, Y, Z) = X$ for some integer V, then BINOMI$(X, Y, Z) = V$, the usual inverse. Otherwise the result of BINOMI(X,Y,Z) is the integer value V, such that BINOMF$(V-1, Y, Z) < X \leq$ BINOMF(V, Y, Z).

18·7. POISSI(X[,Y])[8] inverse Poisson distribution function

POISSI(X[,Y]) computes a scalar which is the value corresponding to the input X of the inverse of the Poisson distribution function with mean Y. X and Y must be scalars.

X must lie in [0,1]. Y must be non-negative. If Y = 0, then zero is always returned. If Y is omitted, it defaults to one. If $X \notin [0, 1]$ or $Y < 0$, then MAXPOS is returned and a warning message is produced.

The Poisson distribution is discrete, with jumps occuring at non-negative integer points. Recall that POISSF is the Poisson distribution function. If POISSF$(V, Y) = X$ for some integer V, then POISSI$(X, Y) = V$, the usual inverse. Otherwise the result of POISSI(X,Y) is the integer value V, such that POISSF$(V-1, Y) < X \leq$ POISSF(V, Y).

18·8. GAMMAI(X[,Q[,R[,S]]])[8] inverse Gamma distribution function

GAMMAI(X[,Q[,R[,S]]]) computes a scalar which is the value corresponding to the input X of the inverse of the gamma distribution function with shape parameters Q,R, and S. X must be a scalar in the interval [0,1]. Q,R, and S are optional scalars which default to values of $Q = 1$, $R = 1$, and $S = 0$, respectively, if not provided. If provided, Q and R must be greater than zero or a warning message is printed and zero is returned.

18·9. BETAI(X[,A[,B[,C[,D[,E]]]]])[8] inverse non-central Beta distribution function

BETAI(X[,A[,B[,C[,D[,E]]]]]) computes a scalar which is the value corresponding to the input X of the inverse of the non-central Beta distribution function with shape parameters A and B, non-centrality parameter C, and range parameters D and E. X,A,B,C,D, and E must be scalars. The result lies in [D,E]. The parameters A,B,C,D, and E are optional. If A and/or B are omitted, they default to $A = 1$ and $B = 1$. If D and/or E are omitted, they default to $D = 0$ and $E = 1$. If C is omitted, it defaults to $C = 0$. X must lie in [0,1]. A and B must be positive. If $X \notin [0, 1]$, $A \leq 0$, or $B \leq 0$, then zero is returned and a warning message is produced.

The standard Beta distribution is defined in terms of the general, non-central Beta distribution function provided in MLAB as BETAF, as BETA$(X, A, B) =$ BETAF$(X, A, B, 0, 0, 1)$.

18·10. WEIBI(X[,Y[,Z[,V]]])[8] inverse Weibull distribution function

WEIBI(X[,Y[,Z[,V]]]) computes a scalar which is the value of the inverse of the Weibull distribution function for the input argument X, width parameter value Y, slope parameter value Z and displacement parameter value V. X, Y, Z and V must be scalars. The argument must lie in [0,1]. The parameters Y, Z, and V are optional. If the width parameter value Y is provided, it must be positive. If Y is omitted, it defaults to 1. If the slope parameter value Z is provided, it must be positive. If Z is omitted, it defaults to 1. If the displacement parameter value V is omitted, it defaults to 0. If $X \notin [0, 1]$, $Y \leq 0$, $Z \leq 0$, or $V \leq 0$, zero is returned and a warning message is produced.

The inverse of the Weibull distribution function is:

$$\text{WEIBI}(X, Y, Z, V) = V + Y(\log(\frac{1}{1-X}))^{(1/Z)}$$

18·11. NBINI(X[,Y[,Z]])[8] inverse negative binomial distribution function

NBINI(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input X of the inverse of the negative binomial distribution function with Y trials and probability of success Z. X, Y, and Z must be scalars.

X and Z must lie in [0,1]. Y must be non-negative. If Y = 0, then zero is always returned. If Y is not an integer, $\lfloor Y \rfloor$ is used. If Y is omitted, it defaults to one. If Z is omitted, it defaults to 0.5. If $X \notin [0,1]$ or $Z \notin [0,1]$ or $Y < 0$, then MAXPOS is returned and a warning message is produced.

The negative binomial distribution is discrete, with jumps occuring at the integer points $0, 1, \ldots$. Recall that NBINF is the negative binomial distribution function. If NBINF(V,Y,Z) = X for some integer V, then NBINI(X,Y,Z) = V, the usual inverse. Otherwise the result of NBINI(X,Y,Z) is the integer value V, such that $\text{NBINF}(V-1, Y, Z) < X \leq \text{NBINF}(V, Y, Z)$.

18·12. EXPI(X[,Y[,Z]])[8] inverse exponential distribution function

EXPI(X[,Y[,Z]]) computes a scalar which is the value corresponding to input X of the exponential distribution function with shifted mean parameter Y and displacement parameter Z. X, Y and Z must be scalars. The result is the value of the variate.

X must lie in [0,1]. If X = 0, then Z is returned. If X = 1, MAXPOS is returned. The optional shifted mean parameter Y must be non-negative. If omitted, Y defaults to one. If $Y < 0$, then zero is returned and a warning message is produced. The optional displacement parameter Z defaults to zero.

18·13. GEOMI(X[,Y])[8] inverse geometric distribution function

GEOMI(X[,Y]) computes a scalar which is the value of the inverse of the geometric distribution, corresponding to the argument X, with probability of success parameter Y. The optional parameter Y, if supplied, must lie in the interval [0,1], otherwise zero is returned and a warning message is produced. If omitted, Y defaults to .5.

The geometrical distribution is discrete, with jumps occuring at the integer points $0, 1, \ldots$. For $X, Y \notin [0,1]$ the result of GEOMI(X,Y) is the integer value V, such that $\text{GEOMF}(V-1, Y) < X \leq \text{GEOMF}(V, Y)$, where GEOMF is the geometric distribution function. If GEOMF(V,Y) = X for some integer V, then $\text{GEOMI}(X, Y) = V$, the usual inverse.

18·14. HGEOMI(X,Y,Z,V[,U])[8] inverse hypergeometric distribution function

HGEOMI(X,Y,Z,V[,U]) computes a scalar which is the value corresponding to the input X of the inverse of the hypergeometric distribution function. X, Y, Z, and V must be scalars. If U is omitted, it defaults to 1.

The central hypergeometric distribution (with U = 1) is the distribution of the number of defectives found when a sample of size Y is drawn without replacement from a lot of size Z actually containing V defectives. U is the non-centrality parameter.

18·15. LOGISI(X[,Y[,Z]])[8] inverse logistic distribution function

LOGISI(X[,Y[,Z]]) computes a scalar which is the value corresponding to input X of the inverse of the logistic distribution function with mean Y and slope Z. X, Y, and Z must be scalars.

X must lie in [0,1]. Y must be positive. If Y is omitted, it defaults to zero. If Z is omitted, it defaults to one. If $Y \leq 0$, zero is returned, and a warning message is produced.

The formula for the inverse of the logistic distribution function is:

$$\text{LOGISI}(X, Y, Z) = Y + (1/Z)\log[\exp(Z(X - Y))/(1 + \exp(Z(X - Y))]$$

18·16. CAUCHYI(X[,Y[,Z]])[8] inverse Cauchy distribution function

CAUCHYI(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input X of the inverse of the Cauchy distribution function with location parameter Y and scale parameter Z. X, Y, and Z must be scalars.

X must lie in [0,1]. If Y is omitted, it defaults to zero. Z must be positive. If omitted, Z defaults to one. If $X \notin [0, 1]$, or $Z \leq 0$, then zero is returned and a warning message is produced.

The formula for the inverse of the Cauchy distribution function is:

$$\text{CAUCHYI}(X, Y, Z) = Y + Z\tan(\pi(X - 0.5))$$

18·17. LAPLACEI(X[,Y[,Z]])[8] inverse Laplace distribution function

LAPLACEI(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input X of the inverse of the Laplace distribution function with displacement Y and width Z. X, Y, and Z must be scalars.

X must lie in [0,1]. If Y is omitted, it defaults to zero. If omitted, Z defaults to one. If $X \notin [0, 1]$ or $Z \leq 0$, then zero is returned and a warning message is produced.

18·18. PARETOI(X[,Y[,Z]])[8] inverse Pareto distribution function

PARETOI(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input X of the inverse of the Pareto distribution function with minimum parameter Y and slope Z. X, Y, and Z must be scalars.

X must lie in [0,1]. If provided, the value of the minimum parameter Y must be non-negative. If omitted, Y defaults to one. If provided, the argument of the slope parameter Z must be non-negative. If omitted, Z defaults to one. If $X \notin [0, 1]$, $Y < 0$, or $Z < 0$, then zero is returned and a warning message is produced.

The formula for the inverse of the Pareto distribution function is:

$$\text{PARETOI}(X, Y, Z) = Y/(1 - X)^{1/Z}$$

18·19. UNII(X[,Y[,Z]])[8] inverse uniform distribution function

UNII(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input X of the inverse uniform distribution function with lower limit Y and upper limit Z. X, Y, and Z must be scalars.

X must lie in [0,1]. If Y is omitted, it defaults to zero. If Z is omitted, it defaults to one. The relation $Y \leq Z$ must hold; if not, or if $X \notin [0, 1]$, then MAXPOS is returned and a warning message is produced.

The formula for the inverse of the uniform distribution function is:

$$\text{UNII}(X, Y, Z) = Y + (Z - Y)X.$$

18·20. TRII(X[,Y[,Z]])[8] inverse triangular distribution function

TRII(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input X of the inverse of the triangular distribution function. X, Y, and Z must be scalars.

X must lie in [0,1]. Y is the lower limit of the distribution; if omitted, it defaults to zero. Z is the upper limit of the distribution; if omitted, it defaults to one. Ordinarily, $Y \le Z$. If $X \notin [0, 1]$, then zero is returned and a warning message is produced.

The formula for the inverse function for the triangular distribution is:

$$\text{TRII}(X, Y, Z) = \begin{cases} Y + (Z - Y)\sqrt{X/2} & \text{for } X \in [0, .5] \\ Z - (Z - Y)\sqrt{(1 - X)/2)} & \text{for } X \in [.5, 1] \end{cases}.$$

18·21. LNORMI(X[,Y[,Z[,V]]])[8] inverse lognormal distribution function

LNORMI(X[,Y[,Z[,V]]]) computes a scalar which is the value corresponding to the input X of the inverse of the lognormal distribution function with log-related mean Y and log-related variance Z and displacement V. X, Y, Z, and V must be scalars.

X must lie in [0,1]. If Y is omitted, it defaults to zero. Z must be positive. If Z is omitted, it defaults to one. If V is omitted it defaults to 0. If $X \notin [0, 1]$ or if $Z \le 0$, then zero is produced and a warning message is produced.

The definition of the lognormal distribution function is given in the specification of the MLAB function LNORMF.

18·22. GUMBELI(X[,Y[,Z]])[8] inverse Gumbel distribution function

GUMBELI(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input X of the inverse of the Gumbel extreme value distribution function with displacement Y and width Z. X, Y, and Z must be scalars.

X must lie in [0,1]. Z must be positive. If Y is omitted, it defaults to zero. Z is the value of the width parameter. If omitted, Z defaults to one. If $X \notin [0, 1]$ or $Z \le 0$, then zero is returned and a warning message is produced.

The formula for the inverse of the Gumbel distribution function is:

$$\text{GUMBELI}(X, Y, Z) = Y - Z \log(-\log(X)).$$

18·23. HNORMI(X[,Y[,Z]])[8] inverse halfnormal distribution function

HNORMI(X[,Y[,Z]]) computes a scalar which is the value of the inverse of the halfnormal distribution function for the input argument X, with mean Y and variance Z. X, Y, and Z must be scalars. X must lie in [0,1]; otherwise a warning message is produced and 0 is returned. Y and Z are optional. If Y is omitted, it defaults to zero. If Z is omitted, it defaults to one. Z must be greater than 0 or else a warning message is printed and zero is returned.

The halfnormal distribution is the restriction of the normal distribution to positive values, so that, for $X > 0$, $\text{HNORMI}(X, Y, Z) = \text{GAUSSI}(1/2 + X/2, Y, Z)$. For $X \le 0$, $\text{HNORMI}(X, Y, Z) = 0$. Y and Z are the respective mean and variance of the underlying normal distribution.

19. Probability Density Functions

19·1. GAUSSD(X[,Y[,Z]])[8] normal density function

GAUSSD(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input X of the probability density function of a normally distributed random variable with mean Y and variance Z . X, Y, and Z must be scalars.

If Y is omitted, it defaults to zero. If Z is omitted, it defaults to one. If Z = 0, zero is returned and a warning message is produced. If $Z < 0$, then abs(Z) is used, and a warning message is produced.

Recall the definition of the (mean Y, variance Z) normal density function:

$$\text{GAUSSD}(X, Y, Z) = \frac{e^{-\frac{(X-Y)^2}{2Z}}}{\sqrt{2\pi Z}}$$

As a function of X, this is a bell-shaped curve of maximum height $(2\pi Z)^{-1/2}$ at X = Y.

19·2. CHISQD(X,Y[,Z])[8] non-central chi-squared density function

CHISQD(X,Y[,Z]) computes a scalar which is the value corresponding to the input X of the probability density function of a chi-squared distributed random variable with Y degrees of freedom and non-centrality parameter Z. X, Y, and Z must be scalars.

The chi-squared density function is defined to be zero if $X \leq 0$. Y must be a positive value. In statistical applications, Y is an integer related to sample size. The optional non-centrality argument Z, if supplied, must be non-negative. If omitted, Z defaults to zero. If $Y \leq 0$ or $Z < 0$, then zero is returned, and a warning message is produced.

Recall the definition of the probability density function of the central chi-squared distribution with Y degrees of freedom:

$$\text{CHISQD}(X, Y) = \frac{X^{\frac{Y}{2}-1}e^{-\frac{X}{2}}}{2^{\frac{Y}{2}}\Gamma(Y/2)}.$$

19·3. STUTD(X,Y[,Z])[8] non-central Student's t density function

STUTD(X,Y[,Z]) computes a scalar which is the value corresponding to the input argument X of the probability density function for a Student's t-distributed random variable with Y degrees of freedom and non-centrality parameter Z. X, Y, and Z must be scalars.

Y must be a positive value. In statistical applications, Y is an integer related to sample size. The optional non-centrality argument Z, if omitted, defaults to zero. In statistical applications, Z is non-negative. If $Y \leq 0$, or $Z < 0$ then zero is returned and a warning message is produced.

Let t be defined as $z/(c/d)^{1/2}$, where z and c are independent random variables such that z is normally distributed with mean 0 and variance 1, and c is chi-squared-distributed with Y degrees of freedom. Then t has a central t-distribution with Y degrees of freedom.

Recall the definition of the probability density function of the central Student's t distribution with Y degrees of freedom:

$$\text{STUTD}(X,Y) = \frac{(1+X^2/Y)^{-(Y+1)/2}}{\sqrt{Y}\beta(1/2,Y/2)} \text{ where } \beta \text{ is the complete beta function.}$$

19·4. ASTUTD(X,Y[,Z])[8] absolute non-central Student's t density

ASTUTD(X,Y[,Z]) computes a scalar which is the value corresponding to the input argument X of the probability density function for the absolute non-central Student's t-distribution with Y degrees of freedom and non-centrality parameter Z. X, Y, and Z must be scalars. The argument Y must be positive. In statistical applications, Y is an integer relating to sample size. The optional non-centrality parameter Z, if omitted, defaults to zero. In statistical applications, Z is non-negative. If $Y \leq 0$, or $Z < 0$ a warning message is produced and zero is returned.

Recall the definition of the probability density function of the absolute Student's t distribution with Y degrees of freedom:

$$\text{ASTUTD}(X,Y,0) = 0 \text{ for } X \leq 0; \text{ASTUTD}(X,Y,0) = 2 \cdot \text{STUTD}(X,Y,0) \text{ for } X > 0.$$

19·5. QFD(X,Y,Z[,W])[8] singly non-central F density function

QFD(X,Y,Z[,W]) computes a scalar which is the value corresponding to the input X of the probability density function of a non-central F-distributed random variable with (Y,Z) degrees of freedom and non-centrality parameter W. X, Y, Z, and W must be scalars.

Y and Z must be positive values. In statistical applications, Y and Z are integers related to sample size. W must be non-negative; if omitted, W defaults to zero, i.e. the central F-density. If $Y \leq 0$, $Z \leq 0$, or $W < 0$, then zero is returned, and a warning message is produced.

Let $F = (\frac{c_1}{Y})/(\frac{c_2}{Z})$, where c_1 and c_2 are independent random variables such that c_1 has a non-central chi-squared distribution with Y degrees of freedom and non-centrality parameter W, and c_2 has a central chi-squared distribution with Z degrees of freedom. Then, F has a non-central F-distribution with (Y,Z) degrees of freedom and non-centrality parameter W.

Recall the definition of the probability density function of the central F-distribution with (Y,Z) degrees of freedom:

$$\text{QFD}(X,Y,Z,0) = \text{QFD}(X,Y,Z) = \frac{X^{Y/2-1}Y^{Y/2}Z^{Z/2}}{\beta(Y/2,Z/2)(Z+YX)^{-(Y+Z)/2}}$$

where β is the complete beta function.

The definition of the probability density function of the singly non-central F-distribution with (Y,Z) degrees of freedom and non-centrality parameter W is given in Johnson & Kotz, Continuous Distributions 2, p 191.

19·6. BINOMD(X[,Y[,Z]])[8] binomial density function

BINOMD(X[,Y[,Z]]) computes a scalar which is the value corresponding to the input argument X of the probability density function for a binomially-distributed random variable S_{YZ} = the number of successes in Y trials with probability of success Z. The result is $\Pr[S_{YZ} = X]$; this is the probability of an event with probability Z occuring X times in Y independent trials. X, Y, and Z must be scalars.

The optional argument Y (the number of Bernoulli trials), if supplied, must be a non-negative integer. If omitted, Y defaults to 1. The optional argument Z (the probability of success), if supplied, must

be a number in the interval [0,1]. If omitted, Z defaults to 0.5. If $Y < 0$, or Y is not an integer, or $Z \notin [0, 1]$, then zero is returned and a warning message is produced. If X is not an integer, if $X < 0$, or if $X > Y$, zero is returned.

Recall the definition of the binomial density function:

$$\text{BINOMD}(X, Y, Z) = \Pr[S_{YZ} = X] = \binom{Y}{X} Z^X (1 - Z)^{Y-X}.$$

19·7. POISSD(X[,Y])[8] Poisson density function

POISSD(X[,Y]) computes a scalar which is the value of the Poisson density function with mean Y corresponding to the input X.

Y must be a non-negative value. If Y is omitted, it defaults to one. If Y = 0, zero is always returned. If $Y < 0$, then zero is returned and a warning message is produced.

Let S_Y be a mean-Y Poisson-distributed random variable. The definition of the Poisson density function is $\Pr[S_Y = X] = e^{-Y} Y^X / X!$ for X a non-negative integer. When X is not a non-negative integer, POISSD(X,Y) returns zero.

19·8. SPEARD(X[,Y])[8] Spearman rank-correlation test density function

SPEARD(X[,Y]) computes a scalar which is the value of the Spearman rank-correlation test density function with sample size parameter $Y = \lfloor Y \rfloor$ corresponding to the input X.

Y must be a non-negative value. If Y is omitted, it defaults to one. If Y = 0, zero is always returned. If $Y < 0$, then zero is returned and a warning message is produced.

The Spearman density function is discrete. It can take on $1 + Y(Y^2 - 1)/6$ equally-spaced values in the interval [-1,1]. The density function is symmetrical about zero.

The Spearman density function asymptotically approaches a Student's t-distribution with $Y - 2$ degrees of freedom for large values of Y. We use the asymptotic form for $Y > 10$.

19·9. WILCD(X,Y)[8] Wilcoxon signed-rank test density function

WILCD(X,Y) computes a scalar which is the value of the Wilcoxon signed-rank test density function with sample size parameter $n = \lfloor Y \rfloor$ corresponding to the input $\lfloor X \rfloor$. X and Y must be scalars. X and Y must be non-negative.

The Wilcoxon distribution is discrete. It can take on values only for integer values in the interval $[0, n(n + 1)/2]$. The distribution is symmetrical about its mean $u = n(n + 1)/4$.

The Wilcoxon distribution asymptotically approaches a normal distribution $N(u, v)$ for large sample sizes, where $v = n(n + 1)(2n + 1)/24$. We use the asymptotic form for $n > 15$, and the exact combinatorial calculation for $n \leq 15$.

19·10. MWD(X,Y,Z)[8] Mann-Whitney-Wilcoxon rank-sum test density function

MWD(X,Y,Z) computes a scalar which is the value of the Mann-Whitney-Wilcoxon rank-sum test density function with sample size parameters $s_1 = \lfloor Y \rfloor$ and $s_2 = \lfloor Z \rfloor$ corresponding to $\lfloor X \rfloor$. X, Y, and Z must be scalars. Y and Z must be non-negative.

Let $m = s_1 + s_2$, and $n = \min(s_1, s_2)$. The Mann-Whitney-Wilcoxon distribution is discrete. Jumps occur only for integer values in the interval $[m(m + 1)/2, m(2n - m + 1)/2]$. The distribution is symmetrical about its mean $u = m(n + 1)/2$.

The Mann-Whitney-Wilcoxon distribution asymptotically approaches a normal distribution $N(u, v)$ for large sample sizes, where $v = m(n+1)(n-m)/12$. We use the asymptotic form for $n > 16$, and the exact combinatorial calculation for $n \leq 15$.

19·11. BETAD(X[,A[,B[,C[,D[,E]]]]])[8] non-central beta density function

BETAD(X[,A[,B[,C[,D[,E]]]]]) computes a scalar which is the value of the non-central beta density function with shape parameters A and B, non-centrality parameter C, and range parameters D and E for the input argument X. X, A, B, C, D and E must be scalars. If A and/or B are omitted, they default to A = 1 and B = 1. If C is omitted, it defaults to 0. If D and/or E are omitted, they default to D = 0 and E = 1. The constraints $A > 0$, $B > 0$, and $E > D$ must hold. If $X \leq E$ or $X \geq E$, the result is zero.

The definition of the central beta density function is:

$$\text{BETAD}(X, A, B, 0, D, E) = \frac{(X-D)^{A-1}(E-X)^{B-1}}{(E-D)^{A+B+1}\beta(A,B)}$$

where β is the complete Beta function.

The function BETAD(X,0,1,0,.5,.5) is known as the arcsine density function.
The function BETAD(X,0,1,0,A,1-A) is sometimes called the generalized arcsine density function.

The non-central Beta density function is defined as the derivative with respect to the first argument of the non-central Beta distribution function:

$$\text{BETAD}(X, A, B, C, D, E) = \frac{\partial \text{BETAF}(X, A, B, C, D, E)}{\partial X}.$$

19·12. GAMMAD(X[,Q[,R[,S]]])[8] gamma density function

GAMMAD(X[,Q[,R[,S]]]) computes a scalar which is the value of the gamma density function with parameters Q,R, and S corresponding to the input argument X. X,Q,R, and S must be scalars. Q and R must be positive. If $X < S$, the result is zero.

If Q is omitted, it defaults to 1. If R is omitted, it defaults to 1. If S is omitted, it defaults to 0.

The definition of the density function of the gamma distribution is:

$$\text{GAMMAD}(X, Q, R, S) = (X-S)^{Q-1}e^{-[(X-S)/R]}R^{-Q}/\Gamma(Q).$$

Note the mean of a gamma-distributed random variable is: $RQ + S$.

19·13. WEIBD(X[,Y[,Z[,V]]])[8] Weibull density function

WEIBD(X[,Y[,Z[,V]]]) computes a scalar which is the value of the density function of the Weibull distribution with width parameter value Y, slope parameter value Z and and displacement parameter value V for the input argument X. X, Y, Z and V must be scalars. The parameters Y, Z, and V are optional. If the width parameter value Y is provided, it must be positive. If Y is omitted, it defaults to 1. If the slope parameter value Z is provided, it must be positive. If Z is omitted, it defaults to 1. If the displacement parameter value V is omitted, it defaults to 0. If $Y \leq 0$ or $Z \leq 0$, zero is returned and a warning message is produced.

The probability density function of the Weibull distribution is:

$$\text{WEIBD}(X, Y, Z, V) = (Z/Y)[(X - V)/Y]^{Z-1} \exp\{-[(X - V)/Y]^Z\} \text{ for } X \geq V, \text{ and 0 otherwise.}$$

19·14. NBIND(X[,Y[,Z]])[8] negative binomial distribution density function

NBIND(X[,Y[,Z]]) computes a scalar which is the value of the negative binomial density function with parameters Y, the number of successes required and Z, the probability of success on a single trial corresponding to the input argument X. X, Y, and Z must be scalars. If the optional argument Y is supplied, it must be a positive integer. If Y is omitted, it defaults to 1. If the optional argument Z is supplied, it must be a probability $\in [0, 1]$. If Z is omitted, it defaults to .5.

The negative binomial distribution is discrete. Let Z be the probability of a single success. Then the random variable B = the number of trials needed to obtain Y successes has the negative binomial distribution. The definition of the density function of the negative binomial distribution is:

$$\text{NBIND}(X, Y, Z) = \Pr[B = X] = \binom{X-1}{Y-1}(1 - Z)^{X-Y} Z^Y.$$

Note NBIND(X,1,Z) = GEOMD(X,Z).

19·15. EXPD(X[,Y[,Z]])[8] exponential density function

EXPD(X[,Y[,Z]]) computes a scalar which is the value of the exponential density function with standard deviation Y and displacement Z corresponding to the input argument X. X, Y, and Z must be scalars. If supplied, Y must be non-negative. If Y is zero, the result is always zero. If Y is omitted, it defaults to one. If the argument Z is omitted, it defaults to zero. If $X < Z$, then the result is zero.

The definition of the density function of the exponential distribution is:

$$\text{EXPD}(X, Y, Z) = \frac{1}{Y} e^{-\frac{X-Z}{Y}}$$

Note the mean of the exponential distribution is Y+Z, and its variance is Y^2.

19·16. GEOMD(X[,Y])[8] geometric density function

GEOMD(X[,Y]) computes a scalar which is the value of the geometric density function with probability of success parameter Y corresponding to the input argument X. If the optional argument Y is supplied, it must lie in the interval [0,1], otherwise zero is returned and a warning message is produced. If Y is omitted, it defaults to .5.

The geometric distribution is a discrete distribution with positive density (mass) only for the integers $0, 1, \ldots$. It describes the distribution of the number of failures X before the first success in an indefinite sequence of Bernoulli trials with probability of success Y. The name geometric distribution arises because the successive terms, corresponding to successive values of X form a geometric series with ratio Y/(1-Y).

The definition of the density function of the geometric distribution is:

$$\text{GEOMD}(X, Y) = Y(1 - Y)^X.$$

19·17. HGEOMD(X,Y,Z,V[,U])[8] hypergeometric distribution density function

HGEOMD(X,Y,Z,V[,U]) computes a scalar which is the value of the hypergeometric density function with the parameters Y, Z, V, and U corresponding to the input X. X represents the number of defectives found when a sample of size Y is drawn without replacement from a set of size Z containing V defectives, with non-centrality parameter U. If the optional non-centrality parameter U is omitted, it defaults to 1, corresponding to the central hypergeometric distribution function. If supplied, U must be a non-negative scalar. If the arguments are not non-negative integers, zero is returned and a warning message is produced.

The definition of the central density function of the hypergeometric distribution is:

$$\Pr[X = x] = \binom{Z}{x}\binom{V-Z}{Y-x}/\binom{V}{Y}.$$

19·18. LOGISD(X[,Y[,Z]])[8] logistic density function

LOGISD(X[,Y[,Z]]) computes a scalar which is the value of the density function of the logistic distribution with mean parameter value Y, and slope parameter value Z for argument X. X, Y, and Z must be scalars. The parameters Y and Z are optional. If Y is omitted, it defaults to 0. If the slope parameter value Z is provided, it must be positive. If Z is omitted, it defaults to 1. If $Z \leq 0$, 0 is returned and a warning message is produced.

The probability density function of the logistic distribution is:

$$\text{LOGISD}(X, Y, Z) = Z\exp(Z(X-Y))/[1+\exp(Z(X-Y))]^2.$$

19·19. CAUCHYD(X[,Y[,Z]])[8] Cauchy density function

CAUCHYD(X[,Y[,Z]]) computes a scalar which is the value of the density function of the Cauchy distribution with displacement parameter value Y, and width parameter value Z for the input argument X. X, Y, and Z must be scalars. The parameters Y and Z are optional. If Y is omitted, it defaults to 0. If the slope parameter value Z is provided, it must be positive. If Z is omitted, it defaults to 1. If $Z \leq 0$, zero is returned and a warning message is produced.

The probability density function of the Cauchy distribution is:

$$\text{CAUCHYD}(X, Y, Z) = 1/[\pi Z(1+\{(X-Y)/Z\}^2)].$$

19·20. LAPLACED(X[,Y[,Z]])[8] Laplace density function

LAPLACED(X[,Y[,Z]]) computes a scalar which is the value of the density function of the Laplace distribution with displacement parameter value Y, and width parameter value Z for the input argument X. X, Y and Z must be scalars. The parameters Y and Z are optional. If Y is omitted, it defaults to 0. If the slope parameter value Z is provided, it must be positive. If Z is omitted, it defaults to 1. If $Z \leq 0$, zero is returned and a warning message is produced.

The probability density function of the Laplace distribution is:

$$\text{LAPLACED}(X, Y, Z) = 1/(2Y)\exp(-|X-Y|/Z).$$

19·21. PARETOD(X[,Y[,Z]])[8] Pareto density function

PARETOD(X[,Y[,Z]]) computes a scalar which is the value of the density function of the Pareto distribution with minimum parameter value Y, and slope parameter value Z for the input argument X. X, Y, and Z must be scalars. The parameters Y and Z are optional. If the minimum parameter value Y is provided, it must be positive. If Y is omitted, it defaults to 1. If the slope parameter value Z is provided, it must be positive. If Z is omitted, it defaults to 1. If $Y \leq 0$ or $Z \leq 0$, zero is returned and a warning message is produced.

The probability density function of the Pareto distribution is:

$$\text{PARETOD}(X, Y, Z) = (Z/X) \cdot (Y/X)^Z, \text{ for } X > Y, \text{ and 0 otherwise.}$$

19·22. UNID(X[,Y[,Z]])[8] uniform density function

UNID(X[,Y[,Z]]) computes a scalar which is the value of the density function of the uniform distribution with minimum parameter value Y, and maximum parameter value Z for the input argument X. X, Y, and Z must be scalars. The parameters Y and Z are optional. If Y and Z are provided, $Y < Z$. If Y is omitted, it defaults to 0. If Z is omitted, it defaults to 1. If $Y \geq Z$, 0 is returned and a warning message is produced.

The probability density function of the uniform distribution is:

$$\text{UNID}(X, Y, Z) = 1/(Z - Y) \text{for } X \in [Y, Z], \text{ and 0 otherwise.}$$

19·23. TRID(X[,Y[,Z]])[8] triangular density function

TRID(X[,Y[,Z]]) computes a scalar which is the value of the density function of the triangular distribution with minimum parameter value Y, and maximum parameter value Z for the input argument X. X, Y, and Z must be scalars. The parameters Y and Z are optional. If Y and Z are provided, $Y < Z$. If Y is omitted, it defaults to 0. If Z is omitted, it defaults to 1. If $Y \geq Z$, 0 is returned and a warning message is produced.

The probability density function of the triangular distribution is given below. Let $m = (Y + Z)/2$:

$$\text{TRID}(X, Y, Z) = \begin{cases} (X - Y)/(Z - Y) & \text{for } X \in [Y, m] \\ 1 - (Z - X)/(Z - Y) & \text{for } X \in [m, Z] \end{cases}.$$

19·24. LNORMD(X[,Y[,Z[,V]]])[8] lognormal distribution density function

LNORMD(X[,Y[,Z[,V]]]) computes a scalar which is the value of the lognormal density function with log-related mean Y, log-related variance Z, and displacement V corresponding to the input argument X. X must be a scalar. If supplied, Y, Z, and V must be scalars. If Y is omitted, it defaults to 0. If Z is omitted, it defaults to 1; if supplied, Z must be greater than zero. However, for Z = 0 the result is always zero. If V is omitted, it defaults to 0.

The definition of the density function of the lognormal distribution is:

$$\text{LNORMD}(X, Y, Z, V) = \frac{e^{.5(\log(X-V)-Y)^2/Z^2}}{(X - V)\sqrt{2\pi}Z}.$$

Note that if L is a lognormally distributed random variable then log(L-V) is a Normal mean Y, variance Z random-variable.

19·25. GUMBELD(X[,Y[,Z]])[8] Gumbel (extreme value) density function

GUMBELD(X[,Y[,Z]]) computes a scalar which is the value of the Gumbel extreme value density function with displacement Y and width Z corresponding to the input argument X. X,Y, and Z must be scalars. Z must be greater than zero or a warning is printed and zero is returned. If Y and/or Z are omitted, they default to $Y = 0$ and $Z = 1$.

The definition of the density function of the Gumbel distribution is:

$$\text{GUMBELD}(X, Y, Z) = \exp(-(X - Y)/Z) \exp(-\exp(-(X - Y)/Z))/Z.$$

19·26. LOGSERD(X,Y)[8] logarithmic series distribution density function

LOGSERD(X,Y) computes a scalar which is the value of the logarithmic series density function with parameter Y corresponding to the input argument $\lfloor X \rfloor$. X and Y must be scalars. X must be ≥ 1, and Y must lie in (0,1).

The definition of the density function of the logarithmic series distribution is:

$$\text{LOGSERD}(X, Y) = -Y^X/[X \log(1 - Y)].$$

19·27. HNORMD(X[,Y[,Z]])[8] halfnormal density function

HNORMD(X[,Y[,Z]]) computes a scalar which is the value of the halfnormal density function with mean Y and variance Z for the input argument X. X, Y, and Z must be scalars. Y and Z are optional. If Y is omitted, it defaults to zero. If Z is omitted, it defaults to one. If supplied, Z must be greater than zero or else a warning message is printed and zero is returned.

The halfnormal density is the restriction of the normal density to positive values, so that, for $X > 0$, HNORMD(X,Y,Z) = 2 GAUSSD(X,Y,Z). For $X \leq 0$, HNORMD(X,Y,Z) = 0. Y and Z are the respective mean and variance of the underlying normal distribution.

---

20. Random Number Generators

The functions in this section compute "pseudo-random" numbers. They are all based upon an algorithm for uniform (0,1) random numbers derived from "Efficient And Portable Combined Random Number Generators" by Pierre l'Ecuyer, in Comm. ACM Vol. 31, pp. 742-749, 1988. Successive calls to the algorithm produce a very long sequence of numbers, which depends upon an initial seed value. Each seed produces a unique sequence. To obtain a given sequence again, its seed should be reused.

The seed, X, is the first argument of most of the generators described below. If X = 0, the result is the next number in the current pseudo-random sequence. If X is negative, the original startup seed established when MLAB is run is used. If X is positive, it is used as a seed for a new pseudo-random sequence and the result is the first number in that sequence. Repeated calls to any generator with the same non-zero value of X will yield the same number each time!

20·1. RAN(X[,Y[,Z]])[11] uniform random number

RAN(X[,Y[,Z]]) computes a scalar which is a uniform [Y,Z] pseudo-random number. X, Y, and Z must be scalars. X is the seed of the random number generator. The optional argument Y is the lower bound for the random numbers; it defaults to zero. The optional argument Z is the upper bound for the random numbers; it defaults to one.

20·2. NORMRAN(X[,Y[,Z]])[11] normal random number

NORMRAN(X[,Y[,Z]]) produces a scalar which is a pseudo-random sample from a mean Y, variance Z, normal distribution. X, Y, and Z must be scalars. X is the seed of the random number generator. The optional argument Y is the mean, which defaults to zero. The optional argument Z is the variance, which defaults to one. NORMRAN uses the polar transformation method to produce (mean 0, variance 1) normal random numbers, which are subsequently transformed to samples which are normally-distributed with the specified mean and variance.

20·3. POISRAN(X[,Y])[11] Poisson-distributed random number

POISRAN(X[,Y]) produces a single sample from the Poisson distribution with mean Y. Y and X must be scalars. X is the seed of the random number generator. The optional argument Y is the mean. Y must be non-negative, although Y = 0 always returns zero. If omitted, Y defaults to one. For $Y \geq 15$, the normal approximation to the Poisson distribution is more accurate than the summation scheme employed here so NORMRAN(X,Y,Y) should be used.

20·4. BINRAN(X[,Y[,Z]])[11] binomial-distributed random number

BINRAN(X[,Y[,Z]]) generates a scalar which is a single sample from the binomial distribution characterized by number-of-trials parameter Y and success probability parameter Z. The mandatory argument X is a scalar seed. The optional argument Y must be a positive integer and the optional argument Z must lie in [0,1]. If Y and/or Z are supplied and do not satisfy these conditions, a warning is printed and zero is returned. If Y and/or Z are not supplied, the default values $Y = 1$ and $Z = 0.5$ are used.

20·5. CAUCHYRAN(X[,Y[,Z]])[11] Cauchy-distributed random number

CAUCHYRAN(X[,Y[,Z]]) generates a scalar which is a single sample from the Cauchy distribution with location parameter Y and scale parameter Z. X is the seed. X,Y, and Z must be scalars. Z must be greater than zero or else a warning is printed and zero is returned. If Y and/or Z are omitted, Y defaults to zero and Z defaults to one.

20·6. GEOMRAN(X[,Y])[11] geometric-distributed random number

GEOMRAN(X[,Y]) generates a scalar which is a single sample from the geometric distribution with optional success parameter Y and mandatory seed X. If Y is omitted, it defaults to 0.5. If Y does not lie in the interval [0,1], then zero is returned and warning message is printed. X must be a scalar.

20·7. GUMBELRAN(X[,Y[,Z]])[11] Gumbel (extreme value) random number

GUMBELRAN(X[,Y[,Z]]) generates a scalar which is a single sample from the Gumbel extreme value distribution with displacement Y and width Z. The input X is the mandatory seed which must be a scalar. Y and Z are optional. If Y is not supplied it defaults to zero. If Z is not supplied it defaults to one. If Z is less than or equal to zero, a warning is printed and zero is returned.

20·8. LOGISRAN(X[,Y[,Z]])[11] logistic-distributed random number

LOGISRAN(X[,Y[,Z]]) generates a scalar which is a single sample from the logistic distribution with mean Y and slope Z. The input X is the mandatory seed which must be a scalar. Y and Z must also be scalars. If Y and/or Z is omitted, Y defaults to zero and Z defaults to one. If supplied, Z must be greater than zero or else a warning message is printed and zero is returned.

20·9. LNORMRAN(X[,Y[,Z[,V]]])[11] lognormal-distributed random number

LNORMRAN(X[,Y[,Z[,V]]]) generates a scalar which is a single sample from the lognormal distribution characterized by log-transformed mean Y, log-transformed standard deviation Z, and displacement V. The seed X must be a scalar. If supplied Y,Z, and V must be scalars. If omitted, Y defaults to zero, Z defaults to one, and V defaults to zero. If supplied, Z must be greater than zero or a warning message is printed and zero is returned.

20·10. NBINRAN(X[,Y[,Z]])[11] negative binomial-distributed random number

NBINRAN(X[,Y[,Z]]) generates a scalar which is a single sample from the negative binomial distribution characterized by number of events Y and probability of event Z. The seed X must be a scalar. If the optional argument Y is supplied, it must be a positive integer. If Y is omitted, it defaults to one. If the optional argument Z is supplied, it must be a scalar in the interval [0,1] or else a warning message is printed and zero is returned as the result. If omitted, Z defaults to 0.5.

20·11. PARETORAN(X[,Y[,Z]])[11] Pareto-distributed random number

PARETORAN(X[,Y[,Z]]) generates a scalar which is a single sample from the Pareto distribution characterized by minimum parameter value Y and slope parameter Z. The seed X must be a scalar. The optional parameter Y must be positive, if supplied, or a warning message is printed and zero is returned. If not supplied, Y defaults to one. The optional parameter Z must be positive or else a warning message is printed and zero is returned. If omitted, Z defaults to one.

20·12. TRIRAN(X[,Y[,Z]])[11] triangular-distributed random number

TRIRAN(X[,Y[,Z]]) generates a scalar which is a single sample from the triangular distribution characterized by minimum value Y and maximum value Z. The seed X must be a scalar. The parameters Y and Z are optional. If provided Y must be less than Z or a warning is printed and zero is returned. If omitted, Y defaults to zero and Z defaults to one.

20·13. WEIBRAN(X[,Y[,Z[,V]]])[11] Weibull-distributed random number

WEIBRAN(X[,Y[,Z[,V]]]) generates a scalar which is a single sample from the Weibull distribution characterized by width parameter Y, slope parameter Z, and displacement parameter V. The mandatory seed X must be a scalar. If supplied, the width parameter Y must be non-negative or else a warning message is printed and zero is returned. If omitted, the default value of Y is one. If supplied, the slope parameter Z must be positive or else a warning message is printed and zero is returned. If omitted, Z defaults to one. The displacement parameter V defaults to zero if omitted.

20·14. HNORMRAN(X[,Y[,Z]])[11] half normal-distributed random number

HNORMRAN(X[,Y[,Z]]) generates a scalar which is a single sample from the half normal distribution of mean Y and variance Z. The mandatory seed X must be a scalar. The mean Y must be a scalar. Y defaults to zero if not supplied. The variance Z must be a scalar greater than zero. If Z is not greater than zero, warning message is printed and zero is returned. If Z is not supplied, it defaults to one.

20·15. EXPRAN(X[,Y])[11] exponential-distributed random number

EXPRAN(X[,Y]) produces a single sample from the exponential distribution with mean Y. Y and X must be scalars. X is the seed of the random number generator. Y, the mean, must be non-negative, although Y = 0 always returns zero. If Y is omitted, it defaults to one. EXPRAN uses the log transformation method to generate unit mean exponential random samples which are subsequently scaled to correspond to the desired mean.

20·16. GAMRAN(X[,Q[,R[,S]]])[11] gamma-distributed random number

GAMRAN(X[,Q[,R[,S]]]) produces a single sample from the gamma distribution of order Q, shape R, and non-centrality S. The input X is the seed value. X,Q,R, and S must be scalars. The optional argument Q is the mean. Q and R must be non-negative, otherwise a warning message is printed and the result is zero. If omitted, Q defaults to one, R defaults to one, and S defaults to zero.

A GAMMA random variable with $R = 1$,$S = 0$ may be interpreted as the waiting time for the Q^{th} event in a Poisson process with unit mean.

20·17. BETARAN(X[,A[,B[,C[,D[,E]]]]])[11] non-central beta-distributed random number

BETARAN(X[,A[,B[,C[,D[,E]]]]]) produces a single sample from the non-central beta distribution with shape parameters A and B, non-centrality parameter C, and range parameters D and E. X is the seed. X,A,B,C,D, and E must be scalars. A and B must be positive or else a warning message is printed and zero is returned. If A and/or B is omitted, they default to $A = 1$ and $B = 1$. If C is omitted, it defaults to $C = 0$. If D and/or E is omitted, they default to $D = 0$ and $E = 1$.

Let x_a and x_b be gamma-distributed random variables with parameters a and b, respectively. Then $x_a/(x_a + x_b)$ is a central beta-distributed random variable with parameters (a, b).

20·18. CHISQRAN(X,Y[,Z])[11] non-central chi squared-distributed random number

CHISQRAN(X,Y[,Z]) produces a single sample from the non-central chi-squared distribution with Y degrees of freedom and non-centrality parameter Z. X is the seed. X,Y, and Z must be scalars. Y and W must be non-negative or else a warning is printed and zero is returned. If omitted, Z defaults to zero.

Let g_n be a gamma-distributed random variable with mean $n/2$. Then $2g_n$ is a central chi-squared distributed random variable with n degrees of freedom.

20·19. FRAN(X,Y,Z[,W])[11] singly non-central F-distributed random number

FRAN(X,Y,Z[,W]) produces a single sample from the non-central F distribution with (Y,Z) degrees of freedom and non-centrality parameter W. X is the seed. X,Y,Z, and W must be scalars. Y, Z and W must be non-negative or else a warning is printed and zero is returned. If Y and Z are both zero, the result is always zero. If W is omitted, it defaults to $W = 0$.

Let c_x be a non-central chi-squared distributed random variable with x degrees of freedom, and c_y be a central chi-squared distributed random variable with y degrees of freedom. Then $(c_x y)/(c_y x)$ is a singly non-central F-distributed random variable with (x, y) degrees of freedom.

20·20. STUTRAN(X,Y[,Z])[11] Student't t-distributed random number

STUTRAN(X,Y[,Z]) produces a single sample from the non-central Student's t distribution with Y degrees of freedom and non-centrality parameter Z. X is the seed. X,Y, and Z must be scalars. Y and Z must be non-negative or else a warning is printed and zero is returned. If Y is zero, the result is always zero. If Z is omitted, it defaults to $Z = 0$.

Let X be an N(0,1)-distributed random variable, and let c_Y be a chi squared-distributed random variable with Y degrees of freedom. Then $(X + Z)/\sqrt{c_Y/Y}$ is a non-central Student's t distributed random variable with Y degrees of freedom and non-centrality parameter Z.

20·21. IRAN(X[,Y[,Z]])[11] integer-valued random number

IRAN(X[,Y[,Z]]) produces a scalar which is a random integer in the closed interval $[\lfloor Y \rfloor, \lceil Z \rceil]$. The input X is the seed value of the random number generator. X, Y, and Z are scalars. Y is the lower limit for the random integers; if omitted, it defaults to zero. Z is the upper limit for the random integers; if omitted, it defaults to one.

20·22. RANPERM(V[,Y[,Z[,X]]])[11] random permutation of random combination

RANPERM(V[,Y[,Z[,X]]]) produces a matrix whose rows form a list of Z vectors, each of which is a random permutation of a random combination of V integers chosen from $\{1, 2, \ldots, Y\}$. X is the seed value of the generator. If X is omitted, it defaults to zero. V,Y, Z, and X must be scalars. The result list of vectors is returned as a matrix with Z rows and V columns.

V must be a positive integer which is the permutation length. Y is an optional integer greater than or equal to V . If Y is omitted, it defaults to V and each row of the result matrix is a permutation of 1:V . Z must be a positive integer which is the number of permutations that will be computed. If Z is omitted, it defaults to one.

---

21. Statistical Computations

21·1. MEAN(M[,X])[0] trimmed arithmetic mean

MEAN(M[,X]) computes the trimmed arithmetic means of the sets of numbers in the columns of M. M must be a matrix. The optional scalar parameter X is the symmetrical trimming fraction. If supplied, X must lie in [0,.5]. If omitted, X defaults to zero, corresponding to the usual arithmetic mean. If $X \notin [0, .5]$, a warning message is produced, and the arithmetic mean is computed.

Recall the definition of the ordinary arithmetic mean of a 1-column matrix M:

$$\text{MEAN}(M) = \frac{1}{n}\sum_{i=1}^{n} M[i], \text{ where } n = \text{NROWS}(M).$$

For the trimmed mean of a 1-column matrix M with $0 \le X < 0.5$, the elements of M are sorted, and FLOOR(NROWS(M)X) of the most extreme array elements are removed from both the high and low ends, after which the usual arithmetic mean is computed on the remaining elements. If X = 0.5, the result is the median. For a multi-column input matrix, the computation is applied separately to each column.

When M has a single column, MEAN produces a scalar result, which is the mean of that column. When M has $p > 1$ columns, MEAN produces a $p \times 1$ matrix, each row of which is the mean of the corresponding column of M. MEAN may be used in function bodies when its argument is a 1-column matrix.

21·2. MEDIAN(M)[9] median

MEDIAN(M) computes the median(s) of the set(s) of numbers in the columns of M. M must be a matrix. When M has a single column, MEDIAN produces a scalar result, which is the median of that column. When M has $p > 1$ columns, MEDIAN produces a $p \times 1$ matrix, each row of which is the median of the corresponding column of M. Thus, MEDIAN may be used in function bodies when its argument is a 1-column matrix.

Recall that MEDIAN(M) is defined in simplified terms to be a number z such that 50% of the elements of M are less than or equal to z, and 50% of the elements of M are greater than or equal to z. Thus, if NROWS(M) is odd, z is the middle element of SORT(M). If NROWS(M) is even, z will be the value midway between the two elements which define the middle of SORT(M).

21·3. MODE(M)[9] mode

MODE(M) computes the mode(s) of the set(s) of numbers in the columns of M. M must be a matrix. When M has a single column, MODE produces a scalar result, which is the mode of that column. When M has $p > 1$ columns, MODE produces a $p \times 1$ matrix, each row of which is the mode of the corresponding column of M. Thus, MODE may be used in function bodies when its argument is a 1-column matrix.

Recall that the mode of a collection of numbers is defined in simplified form as the most frequently occurring number in that collection. If there is more than one number which occurs with the maximum frequency (which must be greater than one), then the least such number is returned. If there are no numbers with frequency greater than one, the median is returned.

21·4. STDDEV(M)[0] standard deviation

STDDEV(M) computes the unbiased sample standard deviation(s) of the set(s) of numbers in the columns of M. M must be a matrix When M has a single column, STDDEV produces a scalar result, which is the standard deviation of that column. When M has $p > 1$ columns, STDDEV produces a $p \times 1$ matrix, each row of which is the standard deviation of the corresponding column of M. Thus, STDDEV may be used in function bodies when its argument is a 1-column matrix.

Recall the definition:

$$\text{STDDEV}(M) = [\frac{1}{\max(1, n-1)} \sum_{i=1}^{n} (M[i] - \text{MEAN}(M))^2]^{1/2}, \text{ where } n = \text{NROWS}(M).$$

21·5. VAR(M)[0] variance

VAR(M) computes the variance(s) of the set(s) of numbers in the columns of M. M must be a matrix. When M has a single column, VAR produces a scalar result, which is the variance of that column. When M has $p > 1$ columns, VAR produces a $p \times 1$ matrix, each row of which is the variance of the corresponding column of M. Thus, VAR may be used in function bodies when its argument is a 1-column matrix.

Recall the definition:

$$\text{VAR}(M) = \frac{1}{\max(1, n-1)} \sum_{i=1}^{n} (M[i] - \text{MEAN}(M))^2, \text{where } n = \text{NROWS}(M).$$

21·6. AVDEV(M)[9] average absolute deviation

AVDEV(M) computes the average absolute deviation(s) of the set(s) of numbers in the columns of M. M must be a matrix. When M has a single column, AVDEV produces a scalar result, which is the average absolute deviation of that column. When M has $p > 1$ columns, AVDEV produces a $p \times 1$ matrix, each row of which is the average absolute deviation of the corresponding column of M. Thus, AVDEV may be used in function bodies when its argument is a 1-column matrix.

Recall the definition:

$$\text{AVDEV}(M) = \frac{1}{n}\sum_{i=1}^{n} |M_i - \text{MEAN}(M)|, \text{ where } n = \text{NROWS}(M).$$

21·7. SKEW(M)[9] skew

SKEW(M) computes the skew(s) of the set(s) of numbers in the columns of M. M must be a matrix. When M has a single column, SKEW produces a scalar result, which is the skew of that column. When M has $p > 1$ columns, SKEW produces a $p \times 1$ matrix, each row of which is the skew of the corresponding column of M. Thus, SKEW may be used in function bodies when its argument is a 1-column matrix.

The definition of SKEW is

$$\text{SKEW}(M) = \frac{1}{n}\sum_{i=1}^{n} \frac{(M[i] - \text{MEAN}(M))^3}{\text{STDDEV}(M)^3}, \text{ where } n = \text{NROWS}(M).$$

21·8. KURT(M)[9] kurtosis

KURT(M) computes the kurtosis value(s) of the set(s) of numbers in the columns of M. M must be a matrix. When M has a single column, KURT produces a scalar result, which is the kurtosis of that column. When M has p columns, KURT produces a $p \times 1$ matrix, each row of which is the kurtosis of the corresponding column of M. Thus, KURT may be used in function bodies when its argument is a 1-column matrix. The definition of kurtosis is

$$\text{KURT}(M) = \frac{1}{n}\sum_{i=1}^{n} \frac{(M_i - \text{MEAN}(M))^4}{\text{STDDEV}(M)^4}, \text{ where } n = \text{NROWS}(M).$$

21·9. COV(M)[9] covariance matrix

COV(M) constructs a matrix which is the covariance matrix of the set of multi-dimensional observations in M. M must be a matrix. Each of the r rows of M is a single observation point, with c = NCOLS(M) components. The result is the $c \times c$ covariance matrix, COV(M). COV(M)$[i, j]$ is an unbiased sample covariance of M COL i and M COL j. Recall the definition:

$$\text{cov}(M)_{i,j} = (M \text{ COL } i) - \text{MEAN}(M \text{ COL } i))' * ((M \text{ COL } j) - \text{MEAN}(M \text{ COL } j))/(r-1)$$

21·10. CORR(M)[9] correlation matrix

CORR(M) constructs a matrix which is the multi-dimensional correlation matrix of a set of observations. M must be a matrix. Each row of M is a single observation, with c components, i.e. NCOLS(M) = c. The result is the $c \times c$ correlation matrix, CORR(M). The correlation matrix of M is related to the covariance matrix of M, so that:

$$\mathrm{CORR}(M)_{i,j} = \frac{\mathrm{COV}(M)_{i,j}}{[\mathrm{COV}(M)_{i,i} \cdot \mathrm{COV}(M)_{j,j}]^{1/2}}.$$

21·11. CDF(M[,N])[9] empirical cumulative distribution function

CDF(M[,N]) constructs a matrix which contains values of the cumulative empirical distribution function of the multi-dimensional data in the matrix M evaluated at a set of multi-dimensional points given in the optional matrix N. M must be a matrix. Let c = NCOLS(M). If N is provided, it must be a c-column matrix whose rows are the points at which evaluation of the c.d.f. is to be performed. If N is not given, it is taken to be the first argument, M, by default. The result of CDF is an NROWS(N) $\times$ ($c+1$) matrix, R, where the first c columns of R are a copy of N sorted in ascending lexicographic order, and R[j,NCOLS(N)+1] is the fraction of rows in M which have all their components less than or equal to the corresponding components in R ROW j COL 1:c.

21·12. HISTO(M[,N[,P]])[9] compute a histogram matrix

HISTO(M[,N[,P]]) computes a matrix representing the 1- or 2- dimensional histogram of the set of univariate or bivariate observations given in M. M must be a 1-column or 2-column matrix, which contains the data to be histogrammed. The optional argument N specifies the x-axis buckets to be used. N must be either a scalar, which is the number of x-axis buckets to be used, or a 1-column vector, the elements of which are the lower limits of the histogram x-axis buckets to be used. These lower-limit-values must be unique and in increasing order. When N is a scalar, a corresponding vector is implied which divides the x-range of the data in M into N equal bucket intervals. If N is omitted, it defaults to a vector which subdivides the x-range of M COL 1 into twenty x-axis buckets of equal size. In all cases, N may now be considered to be a vector.

When the argument P is not given, M must be a 1-row or 1-column matrix. In this univariate case, the elements of M are counted according to their membership in the half-open intervals $[N[1], N[2]), [N[2], N[3]), \ldots, [N[k-1], N[k]), [N[k], \mathrm{MAXPOS})$, where k = NROWS(N). If $N[1] \leq \min(M)$, the result is an NROWS(N) $\times$ 2 matrix H, where H col 1 is a copy of N. $H[i,2]$ is the number of elements of M which lie in the interval $[N[i], N[i+1])$, where $N[k+1]$ is taken to be MAXPOS.

If $N[1] > \min(M)$, then H will have one more row, and H[1,1] will contain the smallest element in M and H[1,2] will contain the number of elements of M which are less than the smallest element of N. The next NROWS(N) rows of H will be just as described above.

When P is given, M must be a 2-column matrix and P is interpreted in exactly the same manner as N to specify a set of y-axis buckets for the y-range. In this bivariate case, the result is an MSIZE(N)$\times$MSIZE(P) matrix H, where $H[i,j]$ is the number of rows, b, of M with $N[i] \leq M[b,1] < N[i+1]$, and $P[j] \leq M[b,2] < P[j+1]$.

21·13. RANKORDER(M)[9] rank ordering

RANKORDER(M) constructs a matrix which contains the rank ordering of the last column of the input matrix M. M must be an $r \times c$ matrix. The result is an $r \times (c+1)$ matrix Y. Y COL $1:c$ is a copy of M, sorted on column c. $Y[i, c+1]$ contains the order rank of Y ROW i, where ties are all given the same rank, namely the arithmetic mean of the ranks of the first and last tied rows.

Example:

Let $X = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \\ 2 \\ 4 \end{bmatrix}$. Then RANKORDER$(X) = \begin{bmatrix} 1 & 1.5 \\ 1 & 1.5 \\ 2 & 3.5 \\ 2 & 3.5 \\ 3 & 5 \\ 4 & 6 \end{bmatrix}$.

21·14. EWT(M)^2 estimated weights for curve-fitting

EWT(M) computes an estimated weight vector suitable for use in curve-fitting a model to the data in the matrix M. M must be a 2- or 3- column matrix representing a curve or surface. M must be in sort on column 1. EWT returns a 1-column matrix of estimated reciprocal variance values to be used as the weight vector for the data points in M. The standard deviation at each point is estimated based on the difference of the data from a smoothed form of the data. These differences are themselves smoothed to obtain an estimated standard-deviation curve. Thus, the error is assumed to be white noise with slowly changing variance. EWT-generated weights are often useful for the simultaneous fitting of two or more data sets with different magnitudes, with different error structures.

21·15. RMEAN(M[,X])* rank-based robust mean estimators

RMEAN(M) computes a scalar or matrix which is a robust estimator of the means of the sets of numbers in the columns of M. M must be a matrix. The optional parameter X selects which estimator should be computed. If omitted, X defaults to 1. If X is outside the valid range, a warning message is produced and the default value (1) of X is used.

Let $z_j(s)$ be the s^{th} percentile of the numbers in M COL j, i.e. $z_j(s)$ = the value in M COL j such that $100s$ percent of the values in M COL j are less than or equal to it. Linear interpolation between adjacent values is used if necessary.

When M has a single column, RMEAN produces a scalar result, which is the robust rank-based estimated mean of that column. When M has $p > 1$ columns, RMEAN produces a $p \times 1$ matrix, each row of which is the rank-based estimated of the corresponding column of M. RMEAN may be used in function bodies when its argument is a 1-column matrix.

Depending upon the value of X, one of the following computations is performed to produce the robust estimator of the mean of M COL j.

X = 1. The value $.3z_j(.33) + .4z_j(.5) + .3z_j(.67)$ is computed for M COL j. This is commonly called Gastworth's estimator.

X = 2. The value $(z_j(.25) + z_j(.75))/2$ is computed for M COL j (efficiency 81%).

X = 3. The value $(z_j(.17) + z_j(.5) + z_j(.83))/3$ is computed for M COL j (efficiency 88%).

X = 4. The value $(z_j(.125) + z_j(.375) + z_j(.625) + z_j(.875))/4$ is computed for M COL j(efficiency 91%).

X = 5. The value $(z_j(.1) + zj(.3) + z_j(.5) + z_j(.7) + z_j(.9))/5$ is computed for M COL j (efficiency 93%).

X = 10. The value $(z_j(.05) + z_j(.15) + \ldots + z_j(.95))/10$ is computed for M COL j (efficiency. 97%).

21·16. RDEV(M[,X])* rank-based robust standard deviation estimators

RDEV(M[,X]) computes a scalar or matrix which is a robust estimator the standard deviation(s) of the set(s) of numbers in the columns of M. M must be a matrix. The optional parameter X selects which estimator should be computed. If omitted, X defaults to 1.

Let $z_j(s)$ be the s^{th} percentile of the numbers in M COL j, i.e. $z_j(s)$ = the value in the range of M COL j such that $100s$ percent of the values in M COL j are less than or equal to it. Linear interpolation between adjacent values is used if necessary.

Depending upon the value of X, one of the following computations is performed to produce the robust estimator of the standard deviation of M COL j.

X = 1. The columns of M are sorted into the arrays q_j. The column medians of $q_j = m_j$ are computed. For each column, the array $y_j = |q_j[i] - m_j|$ is computed. The scalar quantity median$(y_j)/0.6745$ is a robust estimator of the standard deviation of M COL j.

X = 2. The scalar $y_j = .3388(z_j(.93) - .3z_j(.07))$ is computed for each column of M (efficiency 65%).

X = 3. The scalar $y_j = .1714(z_j(.97) + z_j(.85) - z_j(.15) - z_j(.03))$ is computed for each column of M (efficiency 81%).

X = 4. The scalar $y_j = .1180(z_j(.98) + z_j(.91) - z_j(.80) - z_j(.20) - z_j(.09) + z_j(.02))$ is computed for each column of M (efficiency 87%).

X = 5. The scalar $y_j = .0935(z_j(.98)+z_j(.93)+z_j(.86)+z_j(.77)+z_j(.33)+z_j(.14)+z_j(.07)+z_j(.02))$ is computed for each column of M (efficiency 90%).

X = 6. The scalar $y_j = .0739(z_j(.985) + z_j(.95) + z_j(.90) + z_j(.84) + z_j(.75) - z_j(.25) + z_j(.16) + z_j(.10) + z_j(.05) + z_j(.015))$ is computed for each column of M (efficiency 92%).

When M has a single column, RDEV produces a scalar result, which is the robust rank-based estimated standard deviation of that column. When M has $p > 1$ columns, rdev produces a $p \times 1$ matrix, each row of which is the rank-based estimated standard deviation of the corresponding column of M. rdev may be used in function bodies when its argument is a 1-column matrix.

21·17. GMEAN(M)* geometric mean

GMEAN(M) computes a scalar which is the geometric mean of the numbers in the matrix M. The data in M must be positive or a warning message is produced and zero is returned.

Recall that the logarithm of the geometric mean is defined to be arithmetic mean of the logarithms of the component numbers. Let r = NROWS(M). Then

$$\text{GMEAN}(M) = \text{EXP}(\text{SUM}(i, 1, r, \text{LOG}(M[i]))/r)$$

21·18. HMEAN(M[,N])* harmonic mean

HMEAN(M[,N]) computes a scalar which is the optionally weighted harmonic mean of the numbers in the 1-dimensional matrix M. The data in M must be non-zero or a warning message is produced and zero is returned. Let r = NROWS(M). The optional matrix N contains weights to be used in computing the harmonic mean. If N is omitted, the list of unit weights $1\hat{}\hat{}r$ will be used by default.

Recall that the log of the geometric mean is defined to be arithmetic mean of the logs of the component numbers. Then the weighted harmonic mean is

$$\text{HMEAN}(M, N) = \frac{\sum_{i=1}^{r} N[i]}{\sum_{i=1}^{r} \frac{N[i]}{M[i]}}$$

21·19. DMEAN(F,M)* discrete functional mean

DMEAN(F,M) computes a scalar which is the discrete functional mean of the numbers in the 1-column matrix M with respect to the function F. The function F should be monotonic. The data in M must be in the range of F or a warning message is produced and zero is returned.

The discrete functional mean is the argument value of F corresponding to the mean value of the function F over the elements of M. Let r = NROWS(M). Then the discrete functional mean is defined by the equation:

$$F(\text{DMEAN}(F, M)) = \frac{1}{r}\sum_{i=1}^{r} F(M[i]).$$

Concretely, DMEAN(F,M) is evaluated by root-finding. Let $V = \frac{1}{r}\sum_{i=1}^{r} F(M[i])$.
Then $\text{DMEAN}(F, M) = \text{ROOT}(Z, \text{MINV}(M), \text{MAXV}(M), V - F(Z))$.

Examples.

F(Z)	mean obtained
1	1
Z	arithmetic mean
LOG(Z)	geometric mean
1/Z	harmonic mean

21·20. FMEAN(F,X,Y)* functional mean

FMEAN(F,X,Y) computes a scalar which is the functional mean of the function F on the interval [X,Y]. The function F should be monotonic. If $Y < X$ a warning message is produced and zero is returned.

The functional mean is the argument value of F corresponding to the mean value of the integral of F over the interval [X,Y]. The functional mean of F is defined by the equation

$$F(\text{FMEAN}(F, X, Y)) = \frac{1}{Y - X}\int_{X}^{Y} F(t)dt.$$

Concretely, FMEAN(F,X,Y) is evaluated by root-finding. Let $V = \frac{1}{Y-X}\int_{X}^{Y} F(t)dt$.

Then FMEAN(F,M) = ROOT(Z,X,Y,V-F(Z)).

Examples:

F(t)	mean obtained
1	1
t	$(X + Y)/2$
t^p	$\frac{1}{p+1}\sum_{k=0}^{p} Y^{p-k}X^k$

21·21. MOMENT(M,X)* sample moment

MOMENT(M,X) computes a scalar which is the X-th sample central moment of the data in the matrix M about its mean. The scalar X must be positive or a warning message is produced and zero is returned. Let v be the mean of the values in M, and n = NROWS(M). Then the X-th sample moment is defined to be

$$\text{MOMENT}(M, X) = \frac{1}{n}\sum_{i=1}^{n} |M[i] - v|^X$$

21·22. ENTROPY(M)[9] entropy of a probability density function

ENTROPY(M) computes a scalar which is the entropy of the discrete distribution specified by the matrix M. M must be a 1-row or 1-column matrix. Let n = MSIZE(M).

Let X be a discrete random variable assuming values in $\{1, \ldots, n\}$ with $Pr[X = i] = p_i$ for $1 \le i \le n$, and $p_1 + p_2 + \ldots + p_n = 1$.

M represents the (possibly unnormalized) density function of the random variable X. Thus, $p_i = M[i] / \sum_{j=1}^{n} M[j]$.

The number of bits of information in a message that specifies the integer $i \in \{1, 2, \ldots, n\}$ with probability p_i is $H[X] = -\sum_{i=1}^{n} p_i \log_2(p_i)$. Thus, H[X] is the expected number of bits received when an optimally-encoded random sample of X is sent. H[X] is called the *information entropy* of X.

In defining entropy in physics, $\log_e$ is used in place of $\log_2$; this results in a constant factor of $1/\log(2)$ difference in the value of H[X].

Note if $p_i = 1$ and $p_1 = p_2 = \ldots = p_{i-1} = p_{i+1} = \ldots = p_n = 0$, then H[X] = 0; there is no information in a message reporting an X-sample. If $p_1 = \ldots = p_n = 1/n$, then $H[X] = \log_2(n)$. In general, $0 \le H[X] \le \log_2(n)$. The narrower the density function of X is, the less its entropy H[X] is.

ENTROPY(M) returns H[X], where M is the (possibly unnormalized) density function tabulation of X.

21·23. JENTROPY(M)[9] entropy of a contingency table

JENTROPY(M) computes a scalar which is the entropy of the discrete joint density function specified by the matrix M. M must be a matrix. Let n = NROWS(M) and m = NCOLS(M).

Let X be a discrete random variable assuming values in $\{1, \ldots, n\}$, and Y be a discrete random variable assuming values in $\{1, \ldots, m\}$. Let $\Pr[X = i \text{ and } Y = j] = r_{i,j}$. Thus, $r_{i,j}$ is a tabulation of the joint density function of (X,Y). Then the joint entropy of X and Y is:

$$H[XY] = -\sum_{i=1}^{n} \sum_{j=1}^{m} r_{i,j} \log_2(r_{i,j}).$$

In general, $H[XY] \in [0, \log_2(nm)]$.

The input is a 2D contingency table M[1:n,1:m]. The rows of M correspond to the n distinct values of the random variable X. The columns of M correspond to the m distinct values of the random variable Y. $M[i, j]$ is the number of observations where X = i occurred together with Y = j, or any fixed constant positive multiple of the relative frequency of the event X = i occurring together with Y = j. $M[i, j]$ is thus a contingency table form of the (possibly unnormalized) joint density function of (X,Y).

JENTROPY(M) returns the value H[XY], where the matrix M contains the (possibly unnormalized) joint density function of X and Y, given in contingency table form.

21·24. CENTROPY(M)[9] conditional entropy of a contingency table

CENTROPY(M) computes a scalar which is the conditional entropy based on the contingency table in the matrix M. M must be a matrix. Let n = NROWS(M) and m = NCOLS(M).

Let X be a discrete random variable assuming values in $\{1, \ldots, n\}$, and let Y be a discrete random variable assuming values in $\{1, \ldots, m\}$. Let $r_{i,j} = \Pr[x = i \text{ and } Y = j]$. Let C_j be a discrete random

variable assuming values in $\{1,\ldots,n\}$ with $\Pr[C_j = i] = \Pr[X = i|Y = j]$. Then the entropy of C_j is

$$H[C_j] = -\sum_{i=1}^{n} \frac{r_{i,j}}{r_{i,\cdot}} \log_2(\frac{r_{i,j}}{r_{i,\cdot}})$$

where $r_{i,\cdot} = \sum_{j=1}^{m} r_{i,j}$, (and $r_{\cdot,j} = \sum_{i=1}^{n} r_{i,j}$). We write $H[X|Y = j]$ in place of $H[C_j]$.

Now we define the entropy of X given Y to be: $H[X|Y] = \sum_{j=1}^{m} r_{\cdot,j} H[X|Y = j]$.

This is the expected value of $H[X|Y = j]$ over j with respect to the distribution of Y. The identities $H[XY] = H[X] + H[Y|X] = H[Y] + H[X|Y]$ hold.

The input is a 2D contingency table M[1:n,1:m]. The rows of M correspond to the n distinct values of a random variable X. The columns of M correspond to the m distinct values of a random variable Y. $M[i, j]$ is the number of observations where $X = i$ occurred together with $Y = j$, or any fixed constant positive multiple of the relative frequency of $X = i$ occurring together with $Y = j$. M is thus a contingency table form of the (possibly unnormalized) joint density function of (X,Y).

CENTROPY(M) = $H[X|Y]$ where M[1:n,1:m] is the (possibly unnormalized) joint density function of X and Y, given in contingency table form.

21·25. UNCERT(M)[9] uncertainty of a contingency table

UNCERT(M) computes a scalar which is the uncertainty of the contingency table in M. M must be a matrix. Let n = NROWS(M) and m = NCOLS(M).

Let X be a discrete random variable assuming values in $\{1,\ldots,n\}$, and let Y be a discrete random variable assuming values in $\{1,\ldots,m\}$.

Now we define the uncertainty of X given Y, in terms of the entropy $H[X]$ and the conditional entropy $H[X|Y]$, as: $U[X|Y] = (H[X] - H[X|Y])/H[X]$. $U[X|Y]$ lies in [0,1]. $U[X|Y]$ is a measure of the dependence of X upon Y. $U[X|Y]$ and $U[Y|X]$ are not necessarily equal!

The input is a 2D contingency table M[1:n,1:m]. The rows of M correspond to the n distinct values of a random variable X. The columns of M correspond to the m distinct values of a random variable Y. $M[i, j]$ is the number of observations where $X = i$ occurred together with $Y = j$, or any fixed constant positive multiple of the relative frequency of $X = i$ occurring together with $Y = j$. $M[i, j]$ is thus a contingency table form of the (possibly unnormalized) joint density function of (X,Y).

UNCERT(M) = U[X—Y] where $M[1 : n, 1 : m]$ is the (possibly unnormalized) joint density function of X and Y, given in contingency table form.

21·26. JUNCERT(M)[9] joint uncertainty of a contingency table

JUNCERT(M) computes a scalar which is the joint uncertainty of the contingency table in M. M must be a matrix. Let n = NROWS(M) and m = NCOLS(M).

The symmetric joint uncertainty of X and Y is:

$$U[XY] = \frac{H[X]U[X|Y] + H[Y]U[Y|X]}{H[X] + H[Y]}.$$

U[XY] is a measure of the association of X and Y akin to the correlation between X and Y.

The input is a 2D contingency table M[1:n,1:m]. The rows of M correspond to the n distinct values of a random variable X. The columns of M correspond to the m distinct values of a random variable Y. M$[i, j]$ is the number of observations where X = i occurred together with Y = j, or any fixed

constant positive multiple of the relative frequency of X = i occurring together with Y = j. M[i, j] is thus a contingency table form of the (possibly unnormalized) joint density function of (X,Y).

JUNCERT(M) = U[XY] where M[1:n,1:m] is the (possibly unnormalized) joint density function of X and Y, given in contingency table form.

---

22. Statistical Tests

Each of the statistical tests described below produces a matrix of results. In addition, each test optionally types on the display a description of the results which were obtained. To obtain this output, the MLAB control variable STATPSW must have a non-zero value (e.g. by having been set to TRUE). To suppress this output, STATPSW must have the value zero (e.g. by having been set to FALSE). STATPSW is TRUE (1) by default.

22·1. TMT(M[,X])[9] Student's t test against a postulated constant mean

TMT(M[,X]) performs a Student's t test for a postulated population mean for the data in M. M must be a 1-row or 1-column matrix. The data are assumed to be drawn from an approximately normal population with unknown variance. X is an optional scalar representing the value of the hypothetical mean. If omitted, X defaults to 0.

The null hypothesis is $H_0 : mean(M) = X$. The 1-sided alternative hypotheses are $H_1^- : mean(M) < X$ and $H_1^+ : mean(M) > X$ and the 2-sided alternative hypothesis is $H_1 : mean(M) \neq X$.

We compute $t_0 = \sqrt{r}(\text{MEAN}(M) - X)/\text{STDDEV}(M)$, where r is the number of elements of M. Under H_0, t_0 is a sample of a Student's t random variable with $r - 1$ degrees of freedom.

This function returns a 4 row matrix, R. R[1] is the Student's t statistic sample value, t_0; $R[2]$ is the 1-tailed probability, $P[t < t_0]$; R[3] is the 1-tailed probability, $P[t > t_0]$; and R[4] is the 2-tailed probability $P[|t| > |t_0|]$.

If R[2] is small, H_0 is unlikely and we may reject H_0 in favor of H_1^-.
If R[3] is small, H_0 is unlikely and we may reject H_0 in favor of H_1^+.
If R[4] is small, H_0 is unlikely and we may reject H_0 in favor of H_1.

22·2. TST(M1,M2)[9] Student's t test for equal means (equal variances)

TST(M1,M2) performs a Student's t test for equality of means on the data arrays M1 and M2. M1 and M2 must be 1-row or 1-column matrices. Let n_1 = MSIZE($M1$) and n_2 = MSIZE($M2$). It is assumed that the data in M1 and M2 are drawn from two approximately normal populations with equal, but unknown, variances.

The null hypothesis is $H_0 : \mu_1 = \mu_2$, with $\sigma_1^2 = \sigma_2^2$, where μ_1 is the mean and σ_1^2 is the variance of the population of which the values in M1 are samples, and μ_2 is the mean and σ_2^2 is the variance of the population of which the values in M2 are samples.

The alternate hypotheses are $H_1^- : \mu_1 < \mu_2$, $H_1^+ : \mu_1 > \mu_2$, and $H_1 : \mu_1 \neq \mu_2$.

Let T_d be a Student's t-distributed random variable with $d = n_1 + n_2 - 2$ degrees of freedom. We compute $s = [\text{mean}(M1) - \text{mean}(M2)]/\sqrt{(1/n_1 + 1/n_2)v}$, where v is the pooled variance of the combined sample M1&M2. Under H_0, s is a sample of the random variable T_d.

Let p_1 be the probability $P[T_d \leq s]$, p_2 be the probability $P[T_d \geq s]$, and p_3 be the probability $P[|T_d| \geq |s|]$. The symmetry of the Student's t distribution implies that $P[T_d \leq -s] = P[T_d \geq s]$.

If p_1 is small, the corresponding negative value s is unlikely under H_0 and we can reject H_0 in favor of H_1^-. If p_2 is small, the corresponding positive value s is unlikely under H_0 and we can reject H_0

in favor of H_1^+. If p_3 is small, such an extreme value of s is unlikely under H_0 and we can reject H_0 in favor of H_1.

TST returns a 4-row matrix R with $R[1] = s$; $R[2] = p_1$; $R[3] = p_2$; and $R[4] = p_3$.

22·3. TDT(M1,M2)[9] Student's t test for equal means (unequal variances)

TDT(M1,M2) performs a Student's t test for equality of means on the data in M1 and M2. M1 and M2 must be 1-row or 1-column matrices. Let $n_1 = \text{MSIZE}(M1)$ and $n_2 = \text{MSIZE}(M2)$. M1 and M2 are assumed to be drawn from approximately normal populations. No assumption is made about the variances of the populations from which the data in M1 and M2 are drawn.

The null hypothesis is $H_0 : \mu_1 = \mu_2$, with σ_1^2 not necessarily equal to σ_2^2, where μ_1 is the mean and σ_1^2 is the variance of the population of which the values in M1 are samples, and μ_2 is the mean and σ_2^2 is the variance of the population of which the values in M2 are samples.

The alternate hypotheses are $H_1^- : \mu_1 < \mu_2$, $H_1^+ : \mu_1 > \mu_2$, and $H_1 : \mu_1 \neq \mu_2$.

Let T_d be a Student's t-distributed random variable with

$$d = \frac{[\text{var}(M1)/n_1 + \text{var}(M2)/n_2]^2}{[(\text{var}(M1)/n_1)^2/(n_1 - 1) + (\text{var}(M2)/n_2)^2/(n_2 - 1)]}$$

degrees of freedom. We compute

$$s = \frac{\text{mean}(M1) - \text{mean}(M2)}{\sqrt{var(M1)/n_1 + var(M2)/n2}}$$

where the denominator is the weighted variance estimate of the difference in means. Under H_0, s is approximately a sample of the random variable T_d.

Let p_1 be the probability $P[T_d \leq s]$, p_2 be the probability $P[T_d \geq s]$, and p_3 be probability $P[|T_d| \geq |s|]$. The symmetry of the Student's t distribution implies that $P[T_d \leq -s] = P[T_d \geq s]$.

If p_1 is small, the corresponding negative value s is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^-. If p_2 is small, the corresponding positive value s is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^+. If p_3 is small, such an extreme value of s is unlikely to arise under H_0 and we can reject H_0 in favor of H_1.

TDT returns a 5-row matrix R with $R[1] = s$; $R[2] = d$; $R[3] = p_1$; $R[4] = p_2$; and $r[5] = p_3$.

22·4. TPT(M1,M2)[9] Student's t test for equal means (paired samples)

TPT(M1,M2) performs a Student's t test for equality of means on the data in M1 and M2. M1 and M2 must be 1-row or 1-column matrices. Let $n_1 = \text{MSIZE}(M1)$ and $n_2 = \text{MSIZE}(M2)$. It is assumed that $M1[i]$ and $M2[i]$ are a paired sample from an approximately normal bivariate distribution with a possibly non-zero covariance.

The null hypothesis is $H_0 : \mu_1 = \mu_2$, where μ_1 is the mean of the population of which the values in μ_1 are samples, and μ_2 is the mean of the population of which the values in μ_2 are samples.

The alternate hypotheses are $H_1^- : \mu_1 < \mu_2$, $H_1^+ : \mu_1 > \mu_2$, and $H_1 : \mu_1 \neq \mu_2$.

Let T_d be a Student's t-distributed random variable with $d = n_1 - 1$ degrees of freedom. We compute

$$s = \frac{\text{mean}(M1) - \text{mean}(M2)}{\sqrt{(\text{var}(M1) + \text{var}(M2) - \text{cov}(M1, M2))/n}}$$

where the denominator is the weighted-variance estimate of the difference in means. Under H_0, s is a sample of Student's t with d degrees of freedom.

Let p_1 be the probability $P[T_d \le s]$, p_2 be the probability $P[T_d \ge s]$, and p_3 be the probability $P[|T_d| \ge |s|]$. The symmetry of the Student's t distribution implies that $P[T_d \le -s] = P[T_d \ge s]$.

If p_1 is small, the corresponding negative value s is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^-. If p_2 is small, the corresponding positive value s is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^+. If p_3 is small, such an extreme value of s is unlikely to arise under H_0 and we can reject H_0 in favor of H_1.

TPT returns a 4-row matrix R with $R[1] = s$; $R[2] = p_1$; $R[3] = p_2$; and $R[4] = p_3$.

22·5. QFT(M1,M2)[9] F-test for equal variances

QFT(M1,M2) performs an F-test for equality of the underlying variances of the data in M1 and M2. M1 and M2 must be 1-row or 1-column matrices. Let $n_1 = \text{MSIZE}(M1)$ and $n_2 = \text{MSIZE}(M2)$. It is assumed that the M1 values are independent samples of a normal random variable with mean μ_1 and variance σ_1^2, and that the M2 values are independent samples of a normal random variable with mean μ_2 and variance σ_2^2.

The null hypothesis is $H_0 : \sigma_1^2/\sigma_2^2 = 1$.

The alternate hypotheses are $H_1^- : \sigma_1^2/\sigma_2^2 < 1$, $H_1^+ : \sigma_1^2/\sigma_2^2 > 1$, and $H_1 : \sigma_1^2/\sigma_2^2 \ne 1$.

Let X be an F-distributed random variable with $(n_1 - 1, n_2 - 1)$ degrees of freedom. We compute $v_1 = \text{var}(M1)$ and $v_2 = \text{var}(M2)$. Under H_0, $v = v_1/v_2$ is a sample of X.

Let p_1 be the probability $\Pr[X \le v]$, let p_2 be the probability $\Pr[X \ge v]$, and let p_3 be the probability $\Pr[X \le \min(v, 1/v) \text{ or } X \ge \max(v, 1/v)]$.

If p_1 is small, a value as small as v is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^-. This is a 1-sided test.

If p_2 is small, a value as large as v is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^+. This is a 1-sided test.

If p_3 is small, a value as extreme as v is unlikely to arise under H_0 and we can reject H_0 in favor of H_1. This is a 2-sided test.

QFT returns a 4-row matrix R with $R[1] = v$; $R[2] = p_1$; $R[3] = p_2$; and $R[4] = p_3$.

22·6. CHISQ1T(M[,E[,X[,V[,Z]]]])[9] chi-square test on data versus expected values

CHISQ1T(M[,E[,X[,V[,Z]]]]) performs a chi-square test to test if each of the samples $M[i]$ was drawn from an approximately normal population with mean E[i] and variance V[i], for $1 \le i \le k$, where $k = \text{MSIZE}(M)$. M, E, and V must be 1-row or 1-column matrices of the same size, k. Each value $M[i]$ is taken to be a sample of an approximately-normally-distributed random variable Y_i for $1 \le i \le k$.

The optional matrix argument E contains estimates of the expected values of the random variables $Y_1, \ldots, Y_k$. If E is omitted, it is assumed that equal frequencies are expected for each of the k outcomes in M, so that $E_i = \sum_{j=1}^{k} \frac{M[j]}{k}$, for $1 \le i \le k$. The optional scalar argument X is the number of constraints on the data in M. If X is omitted, it is assumed that there is a single constraint, i.e. X = 1 by default.

The optional matrix argument V contains estimates of the variance values of the random variables $Y_1, \ldots, Y_k$. If V is omitted, it is assumed that the Poisson model applies, so that $V[i] = E[i]$, for $1 \le i \le k$.

The optional scalar argument Z is the Yates' continuity correction switch value. If Z is omitted, or if Z is supplied and the value of Z is $\neq 0$ (true), then Yates' correction is to be used. If the value of Z is 0 (FALSE), then Yates' correction is not to be used.

Yates' continuity correction compensates for the error introduced by approximating a discrete distribution (typically binomial) by a continuous normal distribution. The correction consists of adding 1/2 to values of $M[i]$ less than the expected and subtracting 1/2 from values of $M[i]$ greater than the expected.

The null hypothesis is H_0: The samples $M[i]$ were drawn from approximately normal populations with mean $E[i]$, and variance $V[i]$ for $i = 1, 2, \ldots, k$.

The alternate hypothesis is H_1: H_0 is false.

Let S_d be a chi-squared distributed random variable with $d = k - X$ degrees of freedom. Given H_0, $c = \frac{(M[1]-E[1])^2}{V[1]} + \ldots + \frac{(M[k]-E[k])^2}{V[k]}$ is a sample of a random variable whose distribution is similar to the distribution of S_d.

We compute the probability $p = \Pr[S_d \geq c]$. If p is small, such a large value of S_d is unlikely to arise under H_0 and we can reject H_0 in favor of H_1. This is a 1-sided test.

CHISQ1T returns a 3-row matrix R with $R[1] = c$, $R[2] = p$, and $R[3] = d$.

A common situation where CHISQ1T applies is the following. Suppose the random variable T assumes the category values $c_1, \ldots, c_k$ with probabilities $p_1, \ldots, p_k$, and that we have n independent samples of T. Let the count random variables Y_i be the number of samples with value c_i. Then Y_i is Poisson-distributed with $\Pr[Y_i = v] = (np_i)^v \exp(-np_i)/v!$, where $p_i = e_i/n$, and $n = \sum e_j$. In this Poisson case, $e_i = \text{mean}(Y_i) = \text{var}(Y_i) = np_i$. The Y_i are independent, except for the single constraint $\sum Y_i = n$. In various other cases, additional constraints may exist, or the usual constraint above may be absent.

Let $D = \sum_{1 \leq i \leq k} \frac{(Y_i - e_i)^2}{e_i}$. D is approximately chi-squared distributed with $k-1$ degrees of freedom. The approximation arises because D is a sum of squared Poisson random variables, while the chi-squared random variable is a sum of squared Gaussian random variables. However, when $e_i = np_i$ is large enough (≥ 4), the two distributions are similar. Given samples $y_1, \ldots, y_k$ of $Y_1, \ldots, Y_k$, if these samples are not consistent with $p_i = e_i / \sum e_j$, then CHISQ1T can be used to detect this situation, whence we may reject the hypothesis that each Y_i is Poisson-distributed with mean e_i.

22·7. CHISQFT(M[,F[,X[,Z]]])[9] chi-square test on histogram vs. cdf

CHISQFT(M[,F[,X[,Z]]]) performs a chi-square test to test if the unnormalized histogram (i.e. the numerical density function) tabulated in the matrix M is based on samples of a random variable V distributed according to the distribution function F. If F is omitted, the uniform distribution spanning the range of values in M COL 1 is assumed. Let n = NROWS(M). M must be a 2-column matrix. M COL 1 contains values in the range of V and M COL 2 contains counts of the number of samples lying in the intervals defined by M COL 1.

The null hypothesis is H_0: The histogram given by M is based on samples of a random variable with the distribution function F.

The alternate hypothesis is H_1: H_0 is false.

Let $E_i = n(F(M[i+1, 1]) - F(M[i, 1]))$. Given H_0, E_i is the expected value of $M[i]$.

The optional scalar argument X is the number of constraints on the data in M. If X is omitted, it is assumed that there is a single constraint, i.e. X = 1 by default.

It is assumed that the Poisson approximation for the binomial distribution applies, so that $\text{var}(M[i]) = E_i$, for $1 \leq i \leq n$.

The optional scalar argument Z is the Yates' continuity correction switch value. If Z is omitted, or if Z is supplied and the value of Z is 1 (TRUE), then Yates' correction is to be used. If the value of Z is 0 (FALSE), then Yates' correction is not to be used.

Yates' continuity correction compensates for the error introduced by approximating a discrete distribution (typically binomial) by a continuous normal distribution. The correction consists of adding 1/2 to values of $M[i,2]$ less than the expected and subtracting 1/2 from values of $M[i,2]$ greater than the expected.

Let S_d be a chi-squared distributed random variable with $d = n - X$ degrees of freedom. Given the null hypothesis, H_0, $c = \frac{(M[1,2]-E_1)^2}{E_1} + \ldots + \frac{(M[n,2]-E_n)^2}{E_n}$ is a sample of a random variable whose distribution is similar to the distribution of S_d.

We compute the probability $p = \Pr[S_d \geq c]$. If p is small, such a large value of S_d is unlikely to arise under H_0 and we can reject H_0 in favor of H_1. This is a 1-sided test.

CHISQFT returns a 3-row matrix R with $R[1] = c$, $R[2] = p$, and $R[3] = d$.

22·8. CHISQ2T(M1,M2[,X])[9] chi-square test on two data sets

CHISQ2T(M1,M2[,X]) performs a chi-square test for statistical identity of the distribution functions underlying the data in M1 and M2. Let k = MSIZE(M1) and MSIZE(M2). M1 and M2 must be equal-sized 1-row or 1-column matrices. The values in M1 and M2 counts of the values which fall in fixed histogram buckets; The particular buckets need not be known.

If the optional scalar argument X is supplied, then X is the number of constraints on the data in M1 and M2. If X is omitted, it defaults to 2, i.e. it is assumed that there are two constraints (one one each data set).

The null hypothesis is H_0: The samples in M1 and the samples in M2 were drawn from populations with the same distribution.

The alternate hypothesis is H_1: H_0 is false.

Suppose the random variables S and T assume the category values $c_1, \ldots, c_k$ with probabilities $p_1, \ldots, p_k$, and that we have n_1 independent samples of S and n_2 independent samples of T. Let the count random variables $Y_{1,i}$ = the number of the S samples with value c_i for $i = 1, 2, \ldots, k$. Let the count variables $Y_{2,i}$ = the number of the T samples with value c_i for $i = 1, 2, \ldots, k$. Then $Y_{1,i}$ and $Y_{2,i}$ are Poisson-distributed with $\Pr[Y_{j,i} = v] = (n_j p_i)^v \exp(-n_j p_i)/v!$, for $j = 1, 2$.

The $Y_{j,i}$ are usually independent, except for the two constraints $\sum Y_{1,i} = n_1$, and $\sum Y_{2,i} = n_2$. However, in addition to the usual constraints specified above, additional constraints may exist, or those constraints may be absent. In any case the total number of constraints is X.

Let $S_d = \frac{(Y_{1,1}-rY_{2,1})^2}{Y_{1,1}+rY_{2,1}} + \ldots + \frac{(Y_{1,k}-rY_{2,k})^2}{Y_{1,k}+rY_{2,k}}$, where $r = \frac{n_1}{n_2}$. S_d is approximately a chi-square-distributed random variable with $d = k - X$ degrees of freedom.

Let $v = \frac{(M1[1]-rM2[1])^2}{M1[1]+rM2[1]} + \ldots + \frac{(M1[k]-rM2[k])^2}{M1[k]+rM2[k]}$. v is a sample of an approximately chi-squared-distributed random variable with d degrees of freedom. The approximation arises because v is a sum of squared Poisson random samples, while the chi-squared random variable is a sum of squared Gaussian random variables. We compute the probability $p = \Pr[S_d \geq v]$.

If p is small, such a large value as v is unlikely to arise under H_0 and we can reject H_0 in favor of H_1. This is a 1-sided test.

CHISQ2T returns a 3-row matrix R with $R[1] = v$, $R[2] = p$, and $R[3] = d$.

22·9. CTABT(M[,X])[9] chi-square test on contingency table

CTABT(M[,X]) performs a chi-square test for lack of treatment effect in the contingency table in M. Let r = NROWS(M) and c = NCOLS(M). The rows of M correspond to the "treatments", while the columns of M correspond to the "outcomes". Thus $M[i,j]$ is the number of subjects that were given treatment i and achieved outcome j. The optional argument X is the value used in M to denote missing data or "structural zeros" in M. A "structural zero" is a cell of M which must be a zero value because of a logical constraint requiring it. For example, the number of pregnant men is necessarily zero. The same value is used for both missing data and structural zeros because they are both treated equivalently, i.e. by ignoring the corresponding cells in M and reducing the number of degrees of freedom by the number of such data cells.

The null hypothesis is H_0: The outcome is independent of the choice of treatment, i.e. the data in the rows of M are each samples from the same distribution.

The alternate hypothesis is H_1: H_0 is false.

Yates' continuity correction is applied to the data in M. Yates' continuity correction compensates for the error introduced by approximating a discrete distribution (typically binomial) by a continuous normal distribution .The correction consists of adding .5 to counts less than the expected value and subtracting .5 from counts greater than the expected value.

Let S_d be a chi-squared distributed random variable with $d = (r-1)(c-1) - z$ degrees of freedom, where z is the number of structural-zero cells in M.

Let $R_i = \sum_j M[i,j]$, $C_j = \sum_i M[i,j]$, $t = \sum_i \sum_j M[i,j]$, and $f_j = C_j/t$. The value f_j is the observed overall frequency of outcome-j.

Let $\delta = \sum_i \sum_j \frac{|M[i,j]-R_i f_j|^2}{R_i f_j}$. Given H_0, δ is approximately a sample of S_d. The approximation arises because δ is a sum involving samples of Poisson random variables, while the chi-squared random variable S_d is a sum of squared Gaussian random variables. However, when R_i is large enough (≥ 4), the chi-squared distribution can be used. We compute the probability $p = \Pr[S_d \geq \delta]$.

If p is small, such a large value as δ is unlikely to arise under H_0 and we can reject H_0 in favor of H_1. This is a 1-sided test.

We also compute two related measures of dependency: Cramer's v and the contingency coefficient g.

Cramer's v is defined as $v = \sqrt{\frac{\delta}{t \cdot \min(r-1,g-1)}}$. $0 \leq v \leq 1$. $v = 0$ if the outcome is independent of the treatment. $v = 1$ if the outcome is completely dependent on the treatment.

The contingency coefficient $g = \sqrt{\frac{\delta}{\delta+t}}$. $0 \leq g < 1$. $g = 0$ if the outcome is independent of the treatment. g approaches 1 if the outcome is completely dependent on the treatment.

CTABT returns a 5-row matrix R with $R[1] = \delta$, $R[2] = p$, $R[3] = d$, $R[4] = V$, and $R[5] = g$.

22·10. KSF1T(M,F)[9] Kolmogorov-Smirnov 1-sample test on a cdf

KS1FT(M,F) performs the Kolmogorov-Smirnov test to test if the empirical cumulative distribution step-function tabulated in M is a consistent estimate of the cumulative distribution specified by F.

M must be a 2-column matrix representing an empirical cumulative distribution function denoted G_M. Let n = NROWS(M). F must be a function of a single variable which is a strictly monotone increasing, continuous cumulative distribution function.

The null hypothesis is H_0: The empirical cumulative distribution function given by M is a consistent estimate of F.

The alternate hypothesis is H_1: H_0 is false.

Let K be a Kolmogorov-Smirnov-distributed random variable with n degrees of freedom. We compute $k^+ = \sqrt{n}\max(G_M - F)$, and $k^- = \sqrt{n}\max(F - G_M)$, and $v = \max(k^+, k^-)$. Under H_0, v is a sample of K. Let x be the value such that $\text{abs}(G_M(x) - F(x)) = v/\sqrt{n}$.

Let p be the probability $\Pr[K \geq v]$. If p is small, such a large value of K as v is unlikely to arise under H_0 and we can reject H_0 in favor of H_1.

KSF1T returns a 5-row matrix R with $R[1] = v$, $R[2] = p$, $R[3] = k^+$, $R[4] = k^-$, and $R[5] = x$.

22·11. KSF2T(M1,M2)[9] Kolmogorov-Smirnov 2-sample test on 2 cdf's

KSF2T(M1,M2) performs the Kolmogorov-Smirnov test for the identity of two cumulative distribution functions F_1 and F_2 for which the matrices M1 and M2 are empirical cumulative distribution function estimates. M1 and M2 must be 2-column matrices which correspond to empirical cumulative distribution functions denoted G_1 and G_2, respectively. Let $n_1 = \text{NROWS}(M1)$, and $n_2 = \text{NROWS}(M2)$.

The null hypothesis is $H_0 : F_1 = F_2$.

The alternate hypothesis is H_1: H_0 is false.

Let K be a Kolmogorov-Smirnov-distributed random variable with $n = n_1 n_2/(n_1 + n_2)$ degrees of freedom. We compute $k^+ = \sqrt{n}\max(G_1 - G_2)$, and $k^- = \sqrt{n}\max(G_2 - G_1)$, and $v = \max(k^+, k^-)$. Under H_0, v is a sample of K. Let x be the value such that $\text{abs}(G_1(x) - G_2(x)) = v/\sqrt{n}$.

Let p be the probability $\Pr[K \geq v]$. If p is small, such a large value of K as v is unlikely to arise under H_0 and we can reject H_0 in favor of H_1.

KS2T returns a 5-row matrix R with $R[1] = v$, $R[2] = p$, $R[3] = k^+$, $R[4] = k^-$, and $R[5] = x$.

22·12. KS1T(M,F)[9] Kolmogorov-Smirnov 1-sample test

KS1T(M,F) performs the Kolmogorov-Smirnov test to test if the population of which the data in M are samples has the cumulative distribution specified by F.

M must be a 1-row or l-column matrix containing independent samples of a random variable X. Let $n = \text{MSIZE}(\text{M})$. F must be a function of a single variable which is a strictly monotone increasing, continuous cumulative distribution function.

The null hypothesis is H_0: The random variable X sampled in M is distributed according to F.

The alternate hypothesis is H_1: H_0 is false. Let G denote the empirical cummulative distribution function of the data in M. Let K be a Kolmogorov-Smirnov-distributed random variable with n degrees of freedom. We compute $k^+ = \sqrt{n}\max(\text{cdf}(M) - F)$, and $k^- = \sqrt{n}\max(F - G)$, and $v = \max(k^+, k^-)$. Under H_0, v is a sample of K. Let x be the value such that $\text{abs}(G(x) - F(x)) = v/\sqrt{n}$.

Let p be the probability $\Pr[K \geq v]$. If p is small, such a large value of K as v is unlikely to arise under H_0 and we can reject H_0 in favor of H_1.

KS1T returns a 5-row matrix R with $R[1] = v$, $R[2] = p$, $R[3] = k^+$, $R[4] = k^-$, and $R[5] = x$.

22·13. KS2T(M1,M2)[9] Kolmogorov-Smirnov 2-sample test

KS2T(M1,M2) performs the Kolmogorov-Smirnov test for the equality of the cumulative distribution functions F_1 and F_2 of the populations from which the data in M1 and M2 are samples. M1 and M2 must be 1-row or 1-column matrices containing F_1-samples and F_2-samples, respectively. Let $n_1 = \text{MSIZE}(M1)$, and $n_2 = \text{MSIZE}(M2)$.

The null hypothesis is $H_0 : F_1 = F_2$. Let G_1 and G_2 denote the empirical cummulative distribution functions of the data in the matices M1 and M2 respectively. The alternate hypothesis is H_1: H_0 is false.

Let K be a Kolmogorov-Smirnov-distributed random variable with $n = n_1 n_2/(n_1 + n_2)$ degrees of freedom. We compute $k^+ = \sqrt{n}\max(G_1 - G_2)$, and $k^- = \sqrt{n}\max(G_2 - G_1)$, and $v = \max(k^+, k^-)$. Under H_0, v is a sample of K. Let x be the value such that $\text{abs}(G_1(x) - G_2(x)) = v/\sqrt{n}$.

Let p be the probability $\Pr[K \geq v]$. If p is small, such a large value of K as v is unlikely to arise under H_0 and we can reject H_0 in favor of H_1.

KS2T returns a 5-row matrix R with $R[1] = v$, $R[2] = p$, $R[3] = k^+$, $R[4] = k^-$, and $R[5] = x$.

22·14. PEART(M1,M2)[9] Pearson correlation test on bivariate normal data

PEART(M1,M2) performs Pearson's correlation-coefficient test on the data arrays M1 and M2. Let $n = \text{MSIZE(M1)} = \text{MSIZE(M2)}$. M1 and M2 must be 1-row or 1-column matrices.

The null hypothesis is H_0: The samples $(M1[i], M2[i])$ for $1 \leq i \leq n$ were drawn from a bivariate normal population with correlation coefficient r equal to zero.

The alternate hypotheses are $H_1^- : r < 0$, $H_1^+ : r > 0$, and $H_1 : r \neq 0$.

We compute the sample corrrelation coefficient value c for the two data arrays:

$$c = \frac{cov(M1, M2)}{\sqrt{\text{var}(M1)\text{var}(M2)}}.$$

c is an estimate of r. Let S be a Student's t-distributed random variable with $d = n - 2$ degrees of freedom. We use the fact that, given H_0, $v = c\sqrt{\frac{n-2}{1-c^2}}$ is a sample of S.

We compute the probability $p_1 = \Pr[S < v]$, the probability $p_2 = \Pr[S > v]$, and the probability $p_3 = 1 - \Pr[S < -v \text{ or } S > v]$.

If p_1 is small, such a large negative value of S as v is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^-. This is a 1-sided test.

If p_2 is small, such a large value of S as v is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^+. This is a 1-sided test.

If p_3 is small, such an extreme value of S as v is unlikely to arise under H_0 and we can reject H_0 in favor of H_1. This is a 2-sided test.

PEART returns a 5-row matrix R with $R[1] = c$, $R[2] = v$, $R[3] = p_1$, $R[4] = p_2$, and $R[5] = p_3$.

22·15. CORR2T(R1,R2,N1[,N2])[9] correlation-coefficient test

CORR2T(R1,R2,N1[,N2]) tests if two correlation coefficient values r_1 and r_2, estimated by R1 and R2, based on samples of size N1 and N2, are equal. R1, R2, N1, and N2 must be scalars. R1 and R2 are estimates of the true population correlation coefficients r_1 and r_2, except, if N2 is omitted, then R2 is assumed to be an exact value equal to r_2.

The null hypothesis is $H_0 : r_1 = r_2$.

The alternate hypotheses are $H_1^- : r_1 < r_2$, $H_1^+ : r_1 > r_2$, and $H_1 : r_1 \neq r_2$.

Given H_0, R1 (and R2 when N2 is given) are samples of the correlation-coefficient sampling random variables S_1 and S_2.

Let $A_1 = .5\log(\frac{1+S_1}{1-S_1})$ and $A_2 = .5\log(\frac{1+S_2}{1-S_2})$. This is Fisher's z-transformation.

Given H_0, A_1 is approximately normally-distributed with mean $m_1 = .5\log(\frac{1+r_1}{1-r_1}) + .5\frac{r_1}{n_1-1}$, and variance $v_1 = \frac{1}{n_1-3}$.

Given H_0, if R2 is not a fixed constant, A_2 is approximately normally-distributed, with mean $m_2 = .5\log(\frac{1+r_2}{1-r_2})$, and variance $v_2 = \frac{1}{n_2-3}$. If R2 is a fixed constant assumed equal to r_2, then $v_2 = 0$ and $A_2 = m_2$.

Let $D = A_1 - A_2$. Let $a_1 = .5\log(\frac{1+R1}{1-R1})$, and $a_2 = .5\log(\frac{1+R2}{1-R2})$. a_1 and a_2 are samples of A_1 and A_2, and $a_d = a_1 - a_2$ is a sample of D.

Let Q be a normally-distributed random variable with mean $m_1 - m_2$ and variance $v_1 + v_2$. Recall that $v_2 = 0$ if $R2 = r_2$.

We compute the probabilities: $p_1 = \Pr[Q < a_d]$, $p_2 = \Pr[Q > a_d]$, and $p_3 = \Pr[|Q| > |a_d|]$.

If p_1 is small, such a large negative value of Q as a_d is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^-. This is a 1-sided test.

If p_2 is small, such a large value of Q as a_d is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^+. This is a 1-sided test.

If p_3 is small, such an extreme value of Q as a_d is unlikely to arise under H_0 and we can reject H_0 in favor of H_1. This is a 2-sided test.

CORR2T returns a 4-row matrix R with $R[1] = a_d$, $R[2] = p_1$, $R[3] = p_2$, and $R[4] = p_3$.

22·16. SPEART(M1,M2)[9] Spearman rank-correlation test on two data sets

SPEART(M1,M2) performs Spearman's rank-correlation test on the data arrays M1 and M2. Let n = MSIZE(M1) = MSIZE(M2). M1 and M2 must be 1-row or 1-column matrices.

Let r be the true correlation of the populations of which M1 and M2 are samples. The null hypothesis is $H_0 : r = 0$. When the two populations are independent, r = 0 holds.

The alternate hypotheses are $H_1^- : r < 0$, $H_1^+ : r > 0$, and $H_1 : r \neq 0$.

We compute the Spearman rank-correlation coefficient sample value:
c = corr(ranks(M1), ranks(M2)). This is the ordinary correlation coefficient, computed for the rank-values of the values in M1 and the rank-values of the values in M2.

Let S be a Spearman rank-correlation random variable with sample size parameter n. Given H_0, c is a sample of S. We compute the probabilities $p_1 = \Pr[S < c]$, $p_2 = \Pr[S > c]$, and $p_3 = \Pr[|S| > |c|] = 1 - \Pr[-|c| < S < |c|]$.

If p_1 is small, such a large negative value of S as c is unlikely under H_0 and we can reject H_0 in favor of H_1^-. This is a 1-sided test.

If p_2 is small, such a large value of S as c is unlikely under H_0 and we can reject H_0 in favor of H_1^+. This is a 1-sided test.

If p_3 is small, such an extreme value of S as c is unlikely under H_0 and we can reject H_0 in favor of H_1. This is a 2-sided test.

SPEART returns a 4-row matrix R with $R[1] = c$, $R[2] = p_1$, $R[3] = p_2$, and $R[4] = p_3$.

22·17. WIL1T(M[,X])[9] Wilcoxon 1-sample signed-rank test

WIL1T(M[,X]) performs the 1-sample Wilcoxon signed-rank test on the data array M and the postulated median value X. Let n = MSIZE(M). M must be a 1-row or 1-column matrix and X must be a scalar. If X is omitted, it defaults to zero.

The null hypothesis is H_0: The median v of the distribution, of which the elements of M are samples, is equal to X.

The alternate hypotheses are $H_1^- : v < X$, $H_1^+ : v > X$, and $H_1 : v \neq X$.

Let W_e be a Wilcoxon signed-rank random variable with effective sample size parameter e. The mean of $W_e = e(e+1)/4$.

To apply the Wilcoxon 1-sample signed-rank test, we compute the values t^+ and t^- as follows. First, we compute the differences between each value $M[i]$ and X. The absolute values of the differences are ranked $1, 2, \ldots, n$, where the smallest value gets 1 and the largest value gets n. Zero differences are discarded. The number of non-zero differences is the effective sample size, e. Tied differences are assigned the average of the tied ranks. The sign of each difference is then assigned to its rank. t^- is then the absolute value of the sum of negative ranks and t^+ is the sum of positive ranks.

The values t^+ and t^- are linearly related sample values of the Wilcoxon signed rank statistic W_e if H_0 is true. The linear relation is $t^+ + t^- = e(e+1)/2$.

We compute the probability $p_1 = \Pr[W_e > t^+]$, the probability $p_2 = \Pr[W_e > t^-]$, and the probability $p_3 = 1 - \Pr[\min(t^+, t^-) < W_e < \max(t^+, t^-)]$.

If p_1 is small, a value of W_e as large as t^+ is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^-. This is a 1-sided test.

If p_2 is small, a value of W_e as large as t^- is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^+. This is a 1-sided test.

If p_3 is small, a partition of the integer $e(e+1)/2$ as extreme as t^+ and t^- is unlikely to arise under H_0 and we can reject H_0 in favor of H_1. This is a 2-sided test.

WIL1T returns a 6-row matrix R with $R[1] = e$, $R[2] = t^+$, $R[3] = t^-$, $R[4] = p_1$, $R[5] = p_2$, and $R[6] = p_3$.

22·18. WIL2T(M1,M2)[9] Wilcoxon 2-sample paired data signed-rank test

WIL2T(M1,M2) performs the 2-sample paired-data Wilcoxon signed-rank test on the data arrays M1 and M2. Let MSIZE(M1) and MSIZE(M2) = n. M1 and M2 must be 1-row or 1-column matrices.

The null hypothesis is H_0: The medians v_1 and v_2, of the distributions of which the paired data M1 and M2 are samples, are equal.

The alternate hypotheses are $H_1^- : v_1 < v_2$, $H_1^+ : v_1 > v_2$, and $H_1 : v_1 \neq v_2$.

Let W_e be a Wilcoxon signed-rank random variable with effective sample size parameter e. The mean of $W_e = e(e+1)/4$.

To apply the Wilcoxon 2-sample paired-data signed-rank test, we compute the values t^+ and t^- as follows. First the n differences between the pairs $M1[i]$ and $M2[i]$ are computed. The absolute values of the differences are ranked $1, 2, \ldots, n$, where the smallest value gets 1 and the largest value gets n. Differences equal to zero are discarded. The number of non-zero differences is the effective sample size, e. Tied, non-zero, differences are assigned the average of the corresponding ranks. The sign of each difference is assigned to its rank. t^+ is then the sum of the positive ranks and t^- is the absolute value of the sum of negative ranks.

The values t^+ and t^- are linearly related sample values of the Wilcoxon signed rank statistic W_e if H_0 is true. The linear relation is $t^+ + t^- = e(e+1)/2$.

We compute the probabilities: $p_1 = \Pr[W_e > t^+]$, $p_2 = \Pr[W_e > t^-]$, and $p_3 = 1 - \Pr[\min(t^+, t^-) < W_e < \max(t^+, t^-)]$.

If p_1 is small, such a large value of W_e as t^+ is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^+.

If p_2 is small, such a large value of W_e as t^- is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^-.

If p_3 is small, a partition of the integer $e(e+1)/2$ as extreme as t^+ and t^- is unlikely to arise under H_0 and we can reject H_0 in favor of H_1.

WIL2T returns a 6-row matrix R with $R[1] = e$, $R[2] = t^+$, $R[3] = t^-$, $R[4] = p_1$, $R[5] = p_2$, and $R[6] = p_3$.

22·19. MWT(M1,M2)[9] Mann-Whitney-Wilcoxon rank-sum test

MWT(M1,M2) performs the Mann-Whitney-Wilcoxon signed-rank test on the data arrays M1 and M2. M1 and M2 must be 1-row or 1-column matrices. Let $n_1 = \min(\text{MSIZE}(M1), \text{MSIZE}(M2))$ and $n_2 = \max(\text{MSIZE}(M1), \text{MSIZE}(M2))$.

The data in M1 and M2 are pooled, sorted into increasing order, and assigned ranks so that the least value receives rank 1, etc. The ranks of tied values are averaged and assigned to each tied value. We compute $w =$ the sum of the ranks for the smallest array. If the arrays are the same size, we choose the array with the smallest rank-sum.

Let v_1 denote the median of the population to which the computed value w corresponds, and let v_2 be the median of the other population. If v_1 turns-out to be the median of the population associated with the M1 data (because w is the rank-sum associated with this data), then the indicator code c is defined to be 1. Otherwise c is defined to be 2.

The null hypothesis is H_0: The medians v_1 and v_2, of the distributions of which the elements of M1 and M2 (or M2 and M1) are samples, are equal.

The alternate hypotheses are $H_1^- : v_1 < v_2$, $H_1^+ : v_1 > v_2$, and $H_1 : v_1 \neq v_2$.

Let W_{n_1,n_2} be a Mann-Whitney-Wilcoxon-distributed random variable with sample-size parameters (n_1, n_2). Under H_0, w is a sample-value from this distribution.

We compute the probabilities $p_1 = \Pr[W_{n_1,n_2} > w]$, $p_2 = \Pr[W_{n_1,n_2} < w]$; and $p_3 = \Pr[|W_{n_1,n_2}| > |w|]$, which we estimate as $2\min(p_1, p_2)$.

If p_1 is small, such a large value of W_{n_1,n_2} as w is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^+. This is a 1-sided test.

If p_2 is small, such a small value of W_{n_1,n_2} as w is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^-. This is a 1-sided test.

If p_3 is small, such an extreme value of W_{n_1,n_2} as w is unlikely to arise under H_0 and we can reject H_0 in favor of H_1. This is a 2-sided test.

MWT returns a 5-row matrix R with $R[1] = w$, $R[2] = c$, $R[3] = p_1$, $R[4] = p_2$, and $R[5] = p_3$.

22·20. KEN1T(M1,M2)[9] Kendall's paired-sample tau correlation coefficient test

KENT(M1,M2) performs the Kendall's tau correlation coefficient test on the data arrays M1 and M2. Let $n = \text{MSIZE(M1)} = \text{MSIZE(M2)}$. M1 and M2 must be 1-row or 1-column matrices. The data in M1 and M2 may be considered to be counts, i.e. $M1[i]$ and $M2[i]$ are the numbers of occurrences of the i-th of n possible ordered outcomes in the first and second samples. Alternatively, the data may be considered as a pair of uniformly-spaced time series, with $M1[i]$ and $M2[i]$ being measured at the same "time".

Kendall's tau (τ) is a non-parametric measure of correlation. τ is computed as the fraction of concordant inversion pairs in M1 and M2, minus the fraction of discordant inversion pairs in M1 and M2. τ must lie in [-1,1]. Positive τ means the two samples tend to increase (or decrease) together. Negative τ means that when one sample increases, the other tends to decrease, and conversely.

The null hypothesis is $H_0 : \tau = 0$. This may be interpreted to mean that there is no correlation between the sequences of values in M1 and M2.

The alternate hypotheses are $H_1^- : \tau < 0$, $H_1^+ : \tau > 0$, and $H_1 : \tau \neq 0$.

We compute n_1, n_2, and n_d, where:
n_1 = the number of times that $M1[i] - M1[j]$ and $M2[i] - M2[j]$ have the same sign,
n_2 = the number of times that $M1[i] - M1[j]$ and $M2[i] - M2[j]$ have the opposite sign,
n_d = the number of times that $M1[i] - M1[j]$ and $M2[i] - M2[j]$ are both non-zero, for all $i, j \in \{1, 2, \ldots, n\}$.

Let T_n be a sampling-Kendall's tau-distributed random variable with sample size parameter n. Given H_0, the mean of T_n is zero, and asymptotically, for large n, T_n is normally-distributed, with mean zero and variance $(4n + 10)/(9n(n - 1))$. We use this asymptotic approximation for all n.

Let $w = \frac{n_d}{\sqrt{n_1 n_2}}$. The value w is a sample value of T_n.

We compute the probabilities: $p_1 = \Pr[T_n > w]$, $p_2 = \Pr[T_n < w]$, and $p_3 = \Pr[|T_n| > |w|]$.

If p_1 is small, such a large value of T_n as w is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^+. This is a 1-sided test.

If p_2 is small, such a large negative value of T_n as w is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^-. This is a 1-sided test.

If p_3 is small, such an extreme value of T_n as w is unlikely to arise under H_0 and we can reject H_0 in favor of H_1. This is a 2-sided test.

KEN1T returns a 4-row matrix R with $R[1] = w$, $R[2] = p_1$, $R[3] = p_2$, and $R[4] = p_3$.

22·21. KEN2T(M)[9] Kendall's tau test on a contingency table

KEN2T(M) performs the Kendall's tau correlation coefficient test on the data array $M[1 : r, 1 : c]$ in M. The data in M may be considered to be "counts" of bivariate data, i.e. $M[i, j]$ is the number of simultaneous occurrences of the i-th (of r possible ordered outcomes) for the first variable and the j-th (of c possible ordered outcomes) for the second variable. Let $n = \sum M[i, j]$, the total number of counts in M; n corresponds to the number of bivariate sample pairs.

Kendall's tau (τ) is a non-parametric measure of correlation. τ is computed as the fraction of concordant inversion pairs in M1 and M2, minus the fraction of discordant inversion pairs in M1 and M2. τ must lie in [-1,1]. Positive τ means the two varibles tend to increase (or decrease) together. Negative τ means that when one variable increases, the other tends to decrease, and conversely.

The null hypothesis is $H_0 : \tau = 0$. This may be interpreted to mean that there is no correlation between the two variables.

The alternate hypotheses are $H_1^- : \tau < 0$, $H_1^+ : \tau > 0$, and $H_1 : \tau \neq 0$.

The $r \times c$ contingency table M may be (in effect) reconstructed into separate paired count data stored in the two 1-column matrices M1 and M2, where the common row-size of the constructed matrices M1 and M2 is rc. This data is then analyzed as in KEN1T.

We compute n_1, n_2, and n_d, where:
n_1 = the number of times that $M1[i] - M1[j]$ and $M2[i] - M2[j]$ have the same sign,
n_2 = the number of times that $M1[i] - M1[j]$ and $M2[i] - M2[j]$ have the opposite sign,
n_d = the number of times that $M1[i] - M1[j]$ and $M2[i] - M2[j]$ are both non-zero, for all $i, j \in \{1, 2, \ldots, rc\}$.

Let T_n be a sampling-Kendall's tau-distributed random variable with sample size parameter n. Given H_0, the mean of T_n is zero. Asymptotically, for large n, T_n is normally-distributed, with mean zero and variance $(4n + 10)/(9n(n - 1))$. We use this asymptotic approximation for all n. Let $w = \frac{n_d}{\sqrt{(n_1 n_2)}}$. The value w is a sample value of T_n.

We compute the probabilities $p_1 = \Pr[T_n > w]$, $p_2 = \Pr[T_n < w]$, and $p_3 = \Pr[|T_n| > |w|]$.

If p_1 is small, such a large value of T_n as w is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^+. This is a 1-sided test.

If p_2 is small, such a large negative value of T_n as w is unlikely to arise under H_0 and we can reject H_0 in favor of H_1^-. This is a 1-sided test.

If p_3 is small, such an extreme value of T_n as w is unlikely to arise under H_0 and we can reject H_0 in favor of H_1. This is a 2-sided test.

KEN2T returns a 4-row matrix R with $R[1] = w$, $R[2] = p_1$, $R[3] = p_2$, and $R[4] = p_3$.

22·22. SKEWT(M)[9] D'Agostino's skewness test for normality

SKEWT(M) performs D'Agostino's skewness test for normality on the data in M. M must be a 1-row or 1-column matrix. Let n = MSIZE(M).

The data in M are considered to be independent samples of a random variable Y with a continuous distribution. D'Agostino's skewness test is sensitive to departures of this distribution from normality due to skewness (crudely, asymmetry).

For a random variable X, the skewness, $skew(X)$, is defined to be $\frac{E(X-E(X))^3}{\mathrm{var}(X)^{3/2}}$. For a normal distribution, the skewness is equal to zero.

For the set of samples $M[1:n]$, the estimated skewness is $s = m_3/m_2^{3/2}$, where m_j is the j-th central sample moment, $m_j = \sum_{i=1}^{n} \frac{[M_i - \mathrm{mean}(M)]^j}{n}$.

D'Agostino's skewness statistic Z is a certain transformation t of the sample skewness with sample size parameter n. $z = t(s)$ is a sample of Z.

The null hypothesis is $H_0 : \mathrm{skew}(Y) = 0$, This holds when Y is normally-distributed.

The alternate hypotheses are:
$H_1^- : \mathrm{skew}(Y) < 0$; the distribution is skewed left towards lower values.
$H_1^+ : \mathrm{skew}(Y) > 0$; the distribution is skewed right towards higher values.
$H_1 : \mathrm{skew}(Y) \neq 0$; the distribution is skewed either left or right.

When H_0 is true, Z is approximately a normally-distributed random variable with mean 0 and variance 1.

SKEWT computes the probabilities $p_1 = \Pr[Z < z]$, $p_2 = \Pr[Z > z]$, and $p_3 = \Pr[|Z| > |z|]$.

If p_1 is small, a negative value of Z as large in magnitude as z is unlikely under H_0 and we can reject H_0 in favor of H_1^-. This is a 1-sided test.

If p_2 is small, a value of Z as large as z is unlikely under H_0 and we can reject H_0 in favor of H_1^+. This is a 1-sided test.

If p_3 is small, a value of Z as extreme as z is unlikely under H_0 and we can reject H_0 in favor of H_1. This is a 2-sided test.

SKEWT returns a 4-row matrix R with $R[1] = z$, $R[2] = p_1$, $R[3] = p_2$, and $R[4] = p_3$.

22·23. KURTT(M)[9] D'Agostino's kurtosis test for normality

KURTT(M) performs D'Agostino's kurtosis test for normality on the data in M. M must be a 1-row or 1-column matrix. Let n = MSIZE(M).

The data values in M are considered to be independent samples of a random variable Y with a continuous distribution. D'Agostino's kurtosis test is sensitive to departures of this distribution from normality due to kurtosis (crudely, peakedness).

For a random variable X, the kurtosis, $kurtosis(X)$, is defined to be $E[(X - E(X))^4]/\mathrm{var}(X)^2$.

For the set of samples $M[1:n]$, the estimated kurtosis is computed as $\beta = m_4/m_2^2$, where m_j is the j-th central sample moment, $m_j = \sum_{i=1}^{n} \frac{[M_i - \mathrm{mean}(M)]^j}{n}$. For a normal distribution with arbitrary

mean and variance, the kurtosis is equal to 3. For n samples from a normal distribution, the expected estimated kurtosis value is $3(n-1)/(n+1)$.

D'Agostino's kurtosis statistic K is a certain transformation s of the sample kurtosis with sample size parameter n. $k = s(\beta)$ is a sample of K.

The null hypothesis is H_0 : kurtosis$(Y) = 0$. When the distribution of Y is normal, kurtosis$(Y) = 0$.

The alternate hypotheses are:

H_1^- : kurtosis$(Y) < 0$; the distribution of Y is less peaked than a normal distribution.
H_1^+ : kurtosis$(Y) > 0$; the distribution of Y is more peaked than a normal.
H_1 : kurtosis$(Y) \neq 0$; the distribution of Y is more or less peaked than a normal.

When H_0 is true, K is approximately a normally-distributed random variable with mean zero and variance one.

KURTT computes the probabilities $p_1 = \Pr[K < k]$, $p_2 = \Pr[K > k]$, and $p_3 = \Pr[|Z| > |k|]$.

If p_1 is small, a negative value of K as large as k in magnitude is unlikely under H_0 and we can reject H_0 in favor of H_1^-. This is a 1-sided test.

If p_2 is small, a value of K as large as k is unlikely under H_0 and we can reject H_0 in favor of H_1^+. This is a 1-sided test.

If p_3 is small, a value of K as extreme as k is unlikely under H_0 and we can reject H_0 in favor of H_1. This is a 2-sided test.

KURTT returns a 4-row matrix R with $R[1] = k$, $R[2] = p_1$, $R[3] = p_2$, and $R[4] = p_3$.

22·24. NORMT(M)[9] D'Agostino's omnibus test for normality

NORMT(M) performs D'Agostino's omnibus test for normality on the data array M. Let n = MSIZE(M). M must be a 1-row or 1-column matrix.

The data values in M are considered to be independent samples of a random variable Y with a continuous distribution. This test involves computing a combination of the sample values z and k, based on M, which are samples of D'Agostino's skewness and kurtosis statistics, Z and K, described previously in connection with SKEWT and KURTT.

D'Agostino's omnibus test statistic is $C = Z^2 + K^2$. The value of $c = z^2 + k^2$ is a sample of C obtained from the sample values z and k.

The null hypothesis is H_0: The distribution of Y has a skewness and kurtosis consistent with a normal distribution.

The alternate hypotheses is: H_1: the distribution is not normal due to skewness or kurtosis.

When H_0 is true, C is approximately a chi-squared random variable with 2 degrees of freedom. We compute D'Agostino's kurtosis statistic sample value k, and D'Agostino's skewness statistic value z. Then c is computed as $k^2 + z^2$.

We compute the probability $p = \Pr[C > c]$.

If p is small, a value of C as extreme as c is unlikely to arise under H_0 and we can reject H_0 in favor of H_1. This is a 1-sided test.

NORMT returns a 4-row matrix R with $R[1] = z$, $R[2] = k$, $R[3] = c$, and $R[4] = p$.

22·25. KWT(M[,X])[9] Kruskal-Wallis multi-sample rank-sum test

KWT(M,X) computes a matrix resulting from performing the multivariate extension of the Mann-Whitney-Wilcoxon rank-sum test. M is a $k \times c$ matrix of multivariate data. The k rows of M

contain the c performance scores of k subjects with respect to c treatments. The optional scalar X is the value used to denote the code for missing data in the matrix M. If X is omitted, it defaults to MAXPOS. Let m_i denote the median of the values in M COL i

The null hypothesis is H_0: Each of the medians m_i of the population of which the entries in M COL i are samples is the same for all i. That is $m_1 = m_2 = \ldots = m_c$.

The alternate hypothesis is H_1: H_0 is false.

Let n be the number of values in M, not counting the missing values. $n \leq kc$. The computation for the Kruskal-Wallis test is to pool the data in M and compute the ranks $r_1, r_2, \ldots, r_n$ of the n elements in the pooled data. Tied ranks are given the average rank of the tied group to which they belong. Denote the number of non-missing entries in M COL i (i.e. not equal to X) by n_i. Next the value h is computed:

$$h = -3(n+1) + \frac{12}{n(n+1)} \sum_{i=1}^{c} \frac{r_i^2}{n_i}$$

Let g be the number of groups of tied data values in the pooled data. We correct h to compensate for the presence of ties by computing

$$h' = \frac{h}{1 - \sum_{k=1}^{g} \frac{t_k^3 - t_k}{n^3 - n}}$$

where t_k is the number of entries in the k-th tied group. If $g = 0$, then $h' = h$.

Let K be a Kruskal-Wallis-distributed random variable with parameters: $n_1, n_2, \ldots, n_c$, where n_i is the number of members in the i-th group. Under H_0, h' is a sample of K. Asymptotically, for large values of c, K is chi squared-distributed with $c - 1$ degrees of freedom.

We compute $p = \Pr[K > h']$. If p is small, a value of K as large as h' is unlikely to arise under H_0 and we can reject H_0 in favor of H_1.

KWT returns a 2-row matrix R with $R[1] = h'$ and $R[2] = p$.

22·26. LEVENET(M[,X])[9] Levene's test for equality of variances

LEVENET(M[,X]) performs Levene's test for equality of variances. M is an $n \times k$ matrix whose columns represent the data groups the homogenity of variances of which is being tested. k must be ≥ 2. X is an optional scalar representing missing values in M. If omitted, X defaults to MAXPOS.

Suppose there are n_i samples in the i-th data group. Then there must be $n - n_i$ occurrences of the missing value code X in M COL i.

The null hypothesis is H_0: the variance v_i of population i of which the entries in M COL i are samples is the same for all i. The alternate hypothesis is H_1: H_0 is false.

Define d_i = mean(M COL i) (excluding the missing value elements in each column). Let $z_{ij} = |M[i,j] - d_i|$, except when $M[i,j]$ is a missing value code, take $z_{ij} = d_i$. Let a be the overall mean of the all the values in M excluding the missing values. Let $N = \sum n_i$ be the total number of data values in M, excluding the missing values. Then we compute the statistic value:

$$W = \frac{\sum_{i=1}^{k} \frac{n_i (d_i - a)^2}{k-1}}{\sum_{i=1}^{k} \sum_{j=1}^{n} \frac{(z_{ij} - d_i)^2}{N-k}}$$

Under H_0, W is approximately a sample of the random variable $F_{k-1,N-k}$.

We compute $p = \Pr[F_{k-1,N-k} > W]$. If p is small, a value of $F_{k-1,N-k}$ as large as W is unlikely to arise under H_0 and we can reject H_0 in favor of H_1. This is a 1-sided test.

LEVENET returns a 2-row matrix R. $R[1] = W$ and $R[2] = p$.

---

23. Signal Processing Functions

23·1. REALDFT(M)[6] amplitude/phase real discrete Fourier transform

REALDFT(M) computes a matrix which contains the amplitude and phase spectrum for a discrete, real-valued, periodic function F of a real argument, defined over one period by the input M which must be a matrix. Let n = NROWS(M). M specifies values of the real-valued function F computed at n equally-spaced points over one period. There are three alternatives for M.

- If NCOLS(M) = 1, the domain of F over one period is assumed to be $0, 1, \ldots, (n-1)$. M COL 1 contains the real values of F, namely F ON $0 : (n-1)$.
- If NCOLS(M) = 2, M COL 1 is assumed to hold n equally-spaced real domain values of F over one period. Thus, the period is $(M[2,1] - M[1,1])n$. M COL 2 contains the corresponding real values of F, namely F ON (M COL 1).

  For NCOLS(M) = 1 or 2, the result of REALDFT(M) is an $a \times 3$ matrix, Q, containing the phase and amplitude spectrum of F, such that

$$F(t) = \sum_{h=1}^{a} Q_{h,2}\cos(2\pi Q_{h,1}t + Q_{h,3} \text{ where } a = \text{NROWS}(Q) = 1 + 2^{\lfloor \log_2 n \rfloor - 1}.$$

  The frequencies produced in Q COL 1 are in units of cycles/unit (i.e. Hertz).

  A power-of-2 FFT algorithm is used. If n is not a power of 2, interpolation on a power-of-2-sized grid is done prior to the FFT-computation.

- If NCOLS(M) = 3, then M COL 1:2 contains the points of a regular grid organized as produced by the CROSS function, for some 2-dimensional rectangular domain. Thus M COL 1:2 is the cross-product of two equally-spaced lists of real values. Such a tensor product basis may be built by use of the CROSS operator. M COL 3 contains the real values of F ON M COL 1:2. The result of REALDFT(M) is an $a \times 6$ matrix, Q, such that

$$F(r,t) = \sum_{h=1}^{a} [Q_{h,3}\cos(2\pi(Q_{h,1}r + Q_{h,2}t + Q_{h,4})) + Q_{h,5}\cos(2\pi(Q_{h,1}r + Q_{h,2}t + Q_{h,6}))]$$

  where a = NROWS(Q).

  This relation is only approximate if the dimensions of the regular grid in M COL 1:2 are not powers of 2; interpolation on a power-of-2 grid is done if necessary.

23·2. DFT(M)[6] discrete Fourier transform

DFT(M) constructs a matrix which contains the discrete Fourier transform of its input. M must be a matrix. The rows of M are taken to specify n = NROWS(M) equally-spaced sample values over a single period of a discrete, periodic, real or complex-valued function F defined on one period by the input M. The function F has a single, real argument. There are four alternatives for M.

- If NCOLS(M) = 1, the domain of F over one period is assumed to be $0, 1, \ldots, n-1$, and M COL 1 contains the real values of the function F, namely F ON $(0 : n-1)$.
- If NCOLS(M) = 2, the domain of F over one period is assumed to be $0, 1, \ldots, n-1$, and M COL 1:2 contains the real and imaginary parts, respectively, of the complex-valued function F, namely F ON $0 : (n-1)$.

  A power-of-2 FFT algorithm is used. If N is not a power of 2, interpolation on a power-of-2-sized grid is done prior to the FFT.
- If NCOLS(M) = 3, M COL 1 is assumed to hold n equally-spaced real domain values of F over one period. M COL 2:3 contain the real and imaginary parts, respectively, of the values of the complex-valued function F, namely F ON (M COL 1).

  In each of these cases, DFT produces a 3-column matrix Q. Q holds the complex coefficients of the discrete Fourier representation of the periodic extension of the specified function F, sampled at $2\hat{}\lfloor \log_2 n \rfloor$ points. Q COL 1 holds the frequency values given in cycles/unit (Hertz); Q COL 2:3 holds the associated complex Fourier coefficients, so that

$$F(t) = \sum_{h=1}^{b} (Q[h,2] + iQ[h,3]) e^{2\pi i Q[h,1] t}, \text{ where } b = \text{NROWS}(Q) = 2^{\lfloor \log_2 n \rfloor}$$

  A power-of-2 FFT algorithm is used. If n is not a power of 2, interpolation on a power-of-2-sized grid is done prior to the FFT.
- If NCOLS(M) = 4, M COL 1:2 contains a tensor product basis for a $p \times q$ two-dimensional rectangular domain, while M COL 3:4 are taken to be the real and imaginary parts, respectively, of the values of a complex-valued function F tabulated on (M COL 1:2). Note $pq = n$. Such a tensor product basis may be obtained by use of the CROSS operator.

  In this case, DFT produces an $ab \times 4$ matrix, Q. Q holds the 2-dimensional discrete Fourier transform of $F(x, y)$, so that

$$F(x,y) = \sum_{h=1}^{ab} (Q[h,3] + iQ[h,4]) e^{2\pi i (Q[h,1]x + Q[h,2]y)}$$

  where $a = 2^{\lfloor \log_2 p \rfloor}$, $b = 2^{\lfloor \log_2 q \rfloor}$. Note that $ab \leq n$, with equality holding only when the dimensions fo the input grid are both powers of two.

23·3. IDFT(M)[6] discrete inverse Fourier transform

IDFT(M) constructs a matrix which contains the inverse discrete Fourier transform. M must be a matrix. Let n = NROWS(M). There are two cases:

- If NCOLS(M) = 3, M COL 1 contains the n equally-spaced frequencies (measured in cycles per unit (Hertz)) corresponding to the complex Fourier coefficients of a real-valued discrete periodic function F. M COL 2:3 contains the real and imaginary parts, respectively, of the complex Fourier coefficients of the complex-valued function F. Thus M is in the form of an output matrix of the DFT function.

  In this case IDFT produces a $b \times 2$ matrix Q. Q holds the real and imaginary values of the function F determined by M, sampled at b equally-spaced intervals on a single period, so that

$$F(t) = \sum_{h=1}^{b} (M[h,2] + iM[h,3]) e^{2\pi i M[h,1] t}, \text{ where } b = 2^{\lfloor \log_2 n \rfloor}$$

A power of 2 FFT algorithm is used. If n is not a power of 2, interpolation on a power of 2-sized mesh is done prior to the FFT.

- If NCOLS(M) = 4, M COL 1:2 contains a $p \times q$ grid of the equally-spaced x- and y- frequencies (measured in cycles per period) corresponding to the Fourier coefficients of a complex-valued function of two real arguments, $F(x, y)$. M COL 3:4 contain the real and imaginary parts, respectively, of the Fourier coefficients of the complex-valued function F on the regualar $p \times q$ grid. Thus M has the same form as an output matrix of the DFT function, so that

$$F(x,y) = \sum_{k=1}^{ab} (M[k,3] + iM[k,4]) e^{2\pi i(M[k,1]x + M[k,2]y)}.$$

where $a = 2^{\lfloor \log_2 p \rfloor}$, $b = 2^{\lfloor \log_2 q \rfloor}$. Note that $ab \leq n$, with equality holding only when p and q are powers of two.

In this case IDFT produces an $ab \times 4$ matrix Q. Q ROW j COL 3:4 holds the real and imaginary values $F(Q[j,1], Q[j,2])$ of F determined by M. The output values of F given in Q sampled at the $p \times q$ equally-spaced points over a single period in both the x- and y- directions.

Recall that, when NROWS(M) = 2^k, IDFT(DFT(M)) = M, nearly exactly, but only approximately if not.

23·4. INTERPOLATE(M,N[,A])[10] 2D and 3D function interpolation
INTERP(M,N[,A])[10] 2D and 3D function interpolation

INTERPOLATE(M,N[,A]) constructs a matrix which contains the result of 2D or 3D function interpolation. M must be a 2- or 3-column matrix containing the 2D or 3D function data. Thus the rows of M are points on a 2D function or points on a 3D surface. N must be a 1- or 2-column matrix of interpolation points. The optional argument A is either a matrix of explicit slope values, or a scalar tangent estimation switch. A is used only in the case where M is a 2-column matrix. There are two cases.

- If M is a 2-column matrix, the rows of M contain points of a proper (i.e. single-valued) function of a single variable, $y = f(x)$. M COL 1 contains the x-values. M COL 2 contains the corresponding function values. N is a 1-column matrix of values on which interpolation is desired. N is usually in sort order, but this is not required. The result is a 2-column matrix, V. V COL 1 is a copy of N. V COL 2 is the cubic spline interpolation of the input data M at the corresponding points of N.

  When the optional argument A is a scalar, it is the tangent estimation switch, which selects the method of estimating tangents as follow. If A is omitted, it defaults to 0. The options for the scalar A are:

  - A = 0: average distance-weighted estimated tangent vectors
  - A = 1: Chordal estimated tangent vectors
  - A = 2: average distance-weighted estimated tangent vectors with zero-slope at local minimum and local maximum points (no overshoot)
  - A = 3: Chordal estimated tangent vectors with zero-slope at local minimum and local maximum points (no overshoot)
  - A = 4: global second-derivative-continuous tangent vectors with natural end-conditions.

  - A = 5: global second-derivative-continuous tangent vectors with end-conditions clamped to be the unit-tangents in the direction of the line segment from point 1 to point 2 and from the next-to-last to the last point.

  When the optional argument A is a matrix, both A and M must have 2 columns and NROWS(A) must equal NROWS(M). Then, each row of A is the 2-component tangent vector of the curve of M at the corresponding point.

- If M is a 3-column matrix, the rows of M contain points of a proper (i.e. single valued) function of two variables $z = f(x, y)$. M COL 1:2 contains the (x, y)-values, which may be arbitrarily positioned in the (x, y)-plane, and which need not be in any special order. M COL 3 contains the corresponding function values. N is a 2-column matrix of (x, y)-values on which interpolation is desired. N is frequently a tensor product set, perhaps created using the CROSS operator. The points in N do not have to appear in any particular order. The result is a 3-column matrix, V. V COL 1:2 is a copy of N. V COL 3 is a modified 5-nearest neighbor interpolation of the input data M at the corresponding points of N. In this case the input argument A is ignored.

23·5. DINTERP(M,N[,A])[10] 2D function derivative interpolation

DINTERP(M,N[,A]) constructs a matrix which contains the derivative of the 2D function that interpolates the data in the matrix M. M must be a 2-column matrix. N must be a 1-column matrix of interpolation points at which the estimated derivative values are to be computed. The optional argument A is either a matrix of explicit slope values, or a scalar tangent estimation switch.

The rows of the 2-column matrix M contain points of a proper (i.e. single-valued) function of a single variable, $y = f(x)$. M COL 1 contains the x-values. M COL 2 contains the corresponding function values. N is a 1-column matrix of values on which derivative values of the interpolating function is desired. N is usually in sort order, but this is not required. The result is a 2-column matrix, V. V COL 1 is a copy of N. V COL 2 contains the estimated derivative values based on the cubic spline interpolation of the input data M at the corresponding points of N.

When the optional argument A is a scalar, it is the tangent estimation switch, which selects the method of estimating tangents as follow. If A is omitted, it defaults to 0. The options for the scalar A are:

- A = 0: scaled average distance-weighted estimated tangent vectors
- A = 1: scaled chordal estimated tangent vectors
- A = 2: scaled average distance-weighted estimated tangent vectors with zero-slope at local minimum and local maximum points (no overshoot)
- A = 3: scaled chordal estimated tangent vectors with zero-slope at local minimum and local maximum points (no overshoot)
- A = 4: global second-derivative-continuous tangent vectors with natural end-conditions.
- A = 5: global second-derivative-continuous tangent vectors with end-conditions clamped to be the unit-tangents in the direction of the line segment from point 1 to point 2 and from the next-to-last to the last point.

When the optional argument A is a matrix, it must be the same size as M. Then, each row of A is the 2-component tangent vector of the curve of M at the corresponding point.

23·6. LOOKUP(M,X)[10] interpolate linearly into a tabular function of 1 variable

LOOKUP(M,X) computes a scalar which is the linear interpolation into a tabular function of a single variable of the form: $y = f(x)$. M must be a matrix with two or more columns, whose first two columns are treated as (x, y) coordinate pairs. M COL 1 must be in sorted order, either increasing or decreasing. X must be a scalar. If X is within the range of x-values in M COL 1, linear interpolation is performed between the two elements of M COL 1 which bracket X, and the linearly interpolated y-value corresponding to X is computed and returned. If X is outside the range of x-values in M COL 1, the appropriate upper or lower limiting y-value in M COL 2 is returned.

Examples: Let `FCT F(X) = LOOKUP(M,X)`, where.

$M = \begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 10 \end{bmatrix}$ Then, $\mathtt{F}(-5) = 0, \mathtt{F}(11) = 10, \mathtt{F}(2.5) = 7.5.$

23·7. SMOOTH(M[,W[,X[,N]]])[10] smooth a tabular function of 1 or 2 variables

SMOOTH(M[,W[,X[,N]]]) constructs a matrix which smooths an input matrix representing 1- or 2-dimensional "noisy" data. M must be a 2- or 3-column matrix which is in sort on column-1. For 1-dimensional data where M is a 2-column matrix, W is an optional scalar which defaults to five. W is the number of points used in smoothing the data. X is an optional scalar which defaults to one. X is the number of smoothing iterations which are to be performed. N is an optional 1- or 2-column matrix of interpolation points at which the smoothed data is to be sampled (by linear interpolation). If N is omitted, it defaults to M COL 1 if NCOLS(M) = 2 and to M COL 1:2 if NCOLS(M) = 3. The result is a matrix which is "smoother" than the input.

If M has 2 columns, it is assumed to represent a "noisy" sample of a function of one variable. The values of the argument are in M COL 1 and the values of the function are in M COL 2. A W-point, variable-interval smoothing algorithm is applied for each smoothing iteration. W has a default value of 5. The points at the ends of the interval are less strongly smoothed than the interior points.

If M has 3 columns, it is assumed to represent a "noisy" sample of a function of two variables. The values of the argument pairs are in M COL 1:2 and the values of the function are in M COL 3. For a single smoothing iteration, the twelve nearest neighbors to each point are used in an inverse-square smoothing operation. Points on the boundaries of the domain are smoothed less strongly than the interior points.

23·8. SPLINE(D,S[,F[,T]])[10] spline interpolation of plane or space curve

SPLINE(D,S[,F[,T]]) constructs a matrix which contains the result of interpolation into the plane or space curve which is implicitly parameterized by relative chordal arc-length ranging from 0 to 1 overall. D must be a 2- or 3-column matrix containing the curve data. S must be a 1-column matrix, which holds the interpolation chordal arc-length parameter values. F is an optional scalar equal to the flatness parameter. The optional argument T must be either a matrix of tangent vectors or a scalar tangent estimation control code. The result is an NROWS(S) × NCOLS(D) matrix R, containing the spline-interpolated coordinates of the curve D at the parameter values in the matrix S.

When D has 2 columns, the rows of D are a sequential list of (x, y) coordinate points on a plane curve. When D has 3 columns, the rows of D are a sequential list of (x, y, z) coordinate points on a space curve.

S is a 1-column matrix of relative chordal arc-length parameter values with respect to the implicitly parameterized curve passing through the points given in D. The entire curve defined by D has relative chordal arc-length values ranging from 0 to 1. The result matrix R will have one row for each entry of S.

The optional flatness parameter F defaults to 1. As F becomes smaller, the interpolating curve becomes increasingly "flat". In the limit, as F goes to 0, the interpolating curve becomes piecewise straight between the input data points.

When the optional argument T is a scalar, it is the tangent estimation switch, which selects the method of estimating tangents as follow. If T is omitted, it defaults to 0. The options for the scalar T are:

- T = 0: scaled average distance-weighted estimated tangent vectors
- T = 1: scaled chordal estimated tangent vectors
- T = 2: scaled average distance-weighted estimated tangent vectors with zero-slope at local minimum and local maximum points (no overshoot)
- T = 3: scaled chordal estimated tangent vectors with zero-slope at local minimum and local maximum points (no overshoot)
- T = 4: global second-derivative-continuous tangent vectors with natural end-conditions.
- T = 5: global second-derivative-continuous tangent vectors with end-conditions clamped to be the unit-tangents in the direction of the line segment from point 1 to point 2 and from the next-to-last to the last point.

When the optional argument T is a matrix, it must have the same dimensions as D. Then, each row of T is the 2- or 3- component tangent vector of the curve of D at the corresponding point.

The result is an NROWS(S) × NCOLS(D) matrix R, the rows of which are the coordinate points of the interpolating spline curve of D at the parameter values in S. The derivative of the interpolating spline curve of D is discontinuous at any point which is duplicated in adjacent rows of D.

23·9. SPLINEP(D,S[,F[,T]])[10] spline interpolate parametric plane or space curve

SPLINEP(D,S[,F[,T]]) constructs a matrix which contains the result of interpolation into an explicitly parametrically-specified curve. D must be a 3- or 4-column matrix containing the curve data. S must be a 1-column matrix which holds the interpolation parameter values. The optional argument F must be a scalar which is the flatness parameter. The optional argument T is an either a matrix of tangent vectors or a scalar tangent estimation control code. The result is an NROWS(S) × (2- or 3- column) matrix R, containing the spline-interpolated coordinates of the curve D at the parameter values in the matrix S.

When D has 3 columns, D COL (1,2) are a set of (x, y) coordinate points on a plane curve (in two dimensions), with the associated parameter values in D COL 3. When D has 4 columns, D COL (1,2,3) are a set of (x, y, z) coordinate points on a space curve (in 3 dimensions), with the associated monotone-increasing parameter values of some parametric specification in D COL 4.

S is a 1-column matrix of parameter values with respect to the parametrically defined curve in D. The result matrix R will have one row for each row of S.

The optional flatness parameter F defaults to 1.0. As F becomes smaller, the interpolating curve becomes increasingly "flat". In the limit, as F goes to zero, the interpolating curve becomes piecewise straight between the input data points.

When the optional argument T is a scalar, it is the tangent estimation switch, which selects the method of estimating tangents as follows: The default value of T is 0. The options for the scalar T are:

- T = 0: scaled average distance-weighted estimated tangent vectors

- T = 1: scaled chordal estimated tangent vectors
- T = 2: scaled average distance-weighted estimated tangent vectors with zero-slope at local minimum and local maximum points (no overshoot)
- T = 3: scaled chordal estimated tangent vectors with zero-slope at local minimum and local maximum points (no overshoot)
- T = 4: global second-derivative-continuous tangent vectors with natural end-conditions.
- T = 5: global second-derivative-continuous tangent vectors with end-conditions clamped to be the unit-tangents in the direction of the line segment from point 1 to point 2 and from the next-to-last to the last point.

When the optional argument T is a matrix, it must be of size NROWS(D) × (NCOLS(D)-1). Each row of T is the 2- or 3- component (depending on whether D is a plane curve or a space curve) tangent vector of the curve of D at the specified points.

The result is an NROWS(S) × (NCOLS(D)-1) matrix R, the rows of which are the computed interpolated coordinate points of the interpolating spline curve of D corresponding to the parameter values in S. The derivative of the interpolating spline curve of d is discontinuous at any point which is duplicated in adjacent rows of d.

23·10. CROSSCORR(X[,Y[,L]])[6] cross-correlation of sets of vectors
CROSSCOV(X[,Y[,L]])[6] covariance of sets of vectors

CROSSCORR(X[,Y[,L])) constructs a matrix that contains the multivariate stationary cross correlation function for the columns of two matrices X and Y. CROSSCOV(X[,Y[,L]]) constructs a matrix that contains the multivariate stationary cross-covariance function for the columns of the two matrices X and Y.

X is a matrix with n rows and a columns. Y is a matrix with m rows and b columns. If Y is omitted, a copy of X is used by default, thus yielding an autocorrelation or autocovariance computation. If provided, the scalar L must be an integer; L is the maximum lag to be used. If L is omitted, it defaults to $nm/2$.

For CROSSCORR the result is a matrix V with 2L+1 rows and ab columns. V COL k is the stationary cross-correlation function tabulated for the lags $-L \le j \le L$ of the sample sequences X COL i and Y COL j, where $i = \text{ceiling}((k-1)/b)$, and $j = \lfloor (k-1)/b \rfloor$. That is, V COL k ROW $(L+1+h)$ is the value $\hat{R}(h)$ for X COL i and Y COL j, where $\hat{R}(h)$ is defined below.

The stochastic process $X = \ldots, X_{-1}, X_0, X_1, \ldots$ is *second order stationary* if each component has the same mean and variance, i.e. if $E(X_i) = E(X_j)$, and $\text{Var}(X_i) = \text{Var}(X_j)$, for all i, j.

Given the stochastic processes $X = \ldots, X_{-1}, X_0, X_1, \ldots$ and $Y = \ldots, Y_{-1}, Y_0, Y_1, \ldots$, the cross-covariance functions of X and Y are defined by $C_i(h) = \text{cov}(X_i, Y_{i+h})$ for $i = \ldots, -1, 0, 1, \ldots$, and $h = \ldots, -1, 0, 1, \ldots$. The processes X and Y are jointly second order stationary when $C_i(h)$ is independent of i; we write $C(h) = \text{cov}(X_i, Y_{i+h})$ for any i, and $C(h)$ is called the *stationary cross-covariance function* of X and Y.

Similarly, the functions $R_i(h) = \text{corr}(X_i, Y_{i+h}) = C_i(h)/(\text{Var}(X_i)\text{Var}(Y_i + h))^{1/2}$ are the cross-correlation functions of X and Y, and when X and Y are jointly second order stationary, $R_i(h)$ is independent of i and we write $R(h) = \text{corr}(X_i, Y_{i+h})$ for any i. The function $R(h)$ is called the *stationary cross-correlation function* of X and Y.

Given finite sample sequences $x_1, x_2, \ldots, x_n$, and $y_1, y_2, \ldots, y_m$, the estimated stationary cross-correlation function is

$$\hat{R}(h) = \frac{1}{r} \sum_{i=s(h)}^{e(h)} \frac{(x_i - m_x)(y_{i+h} - m_y)}{\sigma_x \sigma_y}$$

where $r = \min(m, n)$, $s(h) =$ if $h < 0$ then $1 - h$ else 1, $e(h) =$ if $h < 0$ then r else $r - h$,

$$m_x = \sum_{i=1}^{n} \frac{x_i}{n}, m_y = \sum_{i=1}^{m} \frac{y_i}{m} \text{ are the estimated means, and}$$

$$\sigma_x = \left[\sum_{i=1}^{n} \frac{(x_i - m_x)^2}{n} \right]^{1/2}, \text{ and } \sigma_y = \left[\sum_{i=1}^{m} \frac{(y_i - m_y)^2}{m_y} \right]^{1/2}$$

are the estimated standard deviations for the first and second sequences, respectively. The value $\hat{R}(h)$ is 0 for $h \leq -r$ and for $h \geq r$. For $-r < h < r$, $\hat{R}(h)$ contains $r - h$ cross terms, and is successively less reliable as $|h| \to r$.

The estimated stationary cross-covariance function is

$$\hat{V}(h) = \frac{1}{r} \sum_{i=s(h)}^{e(h)} (x_i - m_x)(y_{i+h} - m_y)$$

which differs from the cross-correlation only by the omission of the normalizing standard deviations.

23·11. DECONV(M,N[,X])[6] inverse convolution (deconvolution)

DECONV(M,N[,X]) computes a 1-column matrix which is a sequence that satisfies a convolution equation. M and N must be 1-column matrices containing, respectively, the sequences:
$m_1, m_2, \ldots, m_k$, and $n_1, n_2, \ldots, n_k$. The optional scalar argument X, determines which of the following computations is performed. If X is omitted, it defaults to 0. X = 0 corresponds to zero-extension convolution, and X = 2 corresponds to the usual periodic extension convolution.

If X = 0, m_1 must be non-zero, and DECONV returns the 1-column matrix Q which satisfies the modified convolution equation: $MQ = N$, which corresponds to the lower triangular linear system:

$$\begin{bmatrix} m_1 & 0 & \ldots & 0 \\ m_2 & m_1 & \ldots & 0 \\ m_3 & m_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ m_k & m_{k-1} & \ldots & m_1 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_k \end{bmatrix} = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ \vdots \\ n_k \end{bmatrix}.$$

When m_i = P(A=i-1) and n_i = P(A+B=i-1), then the X=0 result is such that q_i = P(B=i-1)m where A and B are independent discrete random variables.

If X = 1, DECONV returns the 1-column matrix Q which satisfies the modified convolution equation: $MQ = N$, which corresponds to the linear system:

$$\begin{bmatrix} m_1 & m_2 & m_3 & \ldots & m_k \\ m_2 & m_1 & m_2 & \ldots & m_{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_k & m_{k-1} & m_{k-2} & \ldots & m_1 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_k \end{bmatrix} = \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_k \end{bmatrix}.$$

If X = 2, DECONV returns the 1-column matrix Q which satisfies the standard convolution equation: $MQ = N$, which corresponds to the linear system:

$$\begin{bmatrix} m_1 & m_k & m_{k-1} & \dots & m_2 \\ m_2 & m_1 & m_k & \dots & . \\ . & m_2 & m_1 & \dots & . \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_k & m_{k-1} & m_{k-2} & \dots & m_1 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_k \end{bmatrix} = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ \vdots \\ n_k \end{bmatrix}.$$

If X = 3, (the Toeplitz matrix case), DECONV returns the 1-column matrix Q which satisfies the linear system:

$$\begin{bmatrix} m_1 & m_{k+1} & m_{k+2} & \dots & m_{2k-1} \\ m_2 & m_1 & m_{k+1} & \dots & m_{2k-2} \\ . & m_2 & . & \dots & . \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_k & m_{k-1} & m_{k-2} & \dots & m_1 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_k \end{bmatrix} = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ \vdots \\ n_k \end{bmatrix}$$

Here, M must have $2k - 1$ elements.

23·12. CONVOLVE(M,N[,X[,Y]])[6] convolution

CONVOLVE(M,N[,X[,Y]]) constructs a matrix which contains a convolution of the data in the matrices M and N. If M and N are 1-dimensional (single rows or single columns), we will do a 1-dimensional convolution. Otherwise, if M or N has more than 1 row and more than 1 column, we will do a 2-dimensional convolution.

• If M and N are both 1-row or 1-column matrices, then CONVOLVE computes the 1-dimensional convolution of M and N. Let p = MSIZE(M), and q = MSIZE(N). Suppose that M contains the sequence of values: $m_1, m_2, \dots, m_p$, and that N contains the sequence of values: $n_1, n_2, \dots, n_q$. Internally, the matrix M or N, whichever has the fewest rows, is extended with zeros so that both matrices have $s = \max(p, q)$ rows.

Now, the length-s sequences given by M and N are further extended to length-r sequences. The optional scalars X and Y define the extension rule for dealing with the ends of the length-r sequences given by M and N, respectively. If X or Y is omitted, it defaults to 1. In the discussion below, the sequence {b} represents either the M-sequence $m_1, \dots, m_s$ or the N-sequence $n_1, \dots, n_s$, depending on whether the argument X or Y, respectively, is taken as the extension code. There are seven options:

0 : zero extension. The sequence $b_1, b_2, \dots, b_s$ is suffixed with s elements with the value 0 and then treated as a periodic sequence with length $r = 2s$.

1 : periodic extension. If both X and Y are 1, both the sequences are treated as though they were periodic with period length $r = s$. The resulting convolution sequence is of length r. Otherwise, the sequence $b_1, b_2, \dots, b_s$, whose extension rule code is 1, is suffixed with a replicate of itself and treated as a periodic sequence with length $r = 2s$.

2 : bi-constant extension. The sequence $b_1, b_2, \dots, b_s$ is suffixed with $\lfloor s/2 \rfloor$ elements with the value b_s, and then further suffixed with $\lceil s/2 \rceil$ elements with the value b_1 and then treated as a periodic sequence with length $r = 2s$.

3 : odd replicate extension. The sequence $b_1, b_2, \dots, b_s$ is suffixed with the sequence: $-b_s, -b_{s-1}, \dots, -b_1$, and then treated as a periodic sequence with length $r = 2s$.

4 : even replicate extension. The sequence $b_1, b_2, \ldots, b_s$ is suffixed with the sequence: $b_s, b_{s-1}, \ldots, b_1$, and then treated as a periodic sequence with length $r = 2s$.

5 : odd extension. The sequence $b_1, b_2, \ldots, b_s$ is suffixed with $-b_{s-1}, -b_{s-2}, \ldots, -b_2$ and then treated as a periodic sequence of length $r = 2s - 2$.

6 : even extension. The sequence $b_1, b_2, \ldots, b_s$ is suffixed with $b_{s-1}, b_{s-2}, \ldots, b_2$ and then treated as a periodic sequence of length $r = 2s - 2$.

Now in all cases, we have constructed two sequences of length r, which are taken as periodic sequences with period length r.

Recall the definition of the cyclic convolution of two discrete sequences $m = \{m_1, m_2, \ldots, m_r\}$ and $n = \{n_1, n_2, \ldots, n_r\}$. The cyclic convolution, g, of m and n, denoted by $m \cdot n$, is defined as the sequence:

$$g_j = \sum_{k=1}^{r} m_k n_{[(j-k+r) \bmod r]+1} \text{ for } j = 1, 2, \ldots, r$$

The 1-dimensional convolution operation may be represented in terms of a circulant matrix, as shown below.

$$\begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_r \end{bmatrix} = \begin{bmatrix} n_1 & n_r & n_{r-1} & \ldots & n_2 \\ n_2 & n_1 & n_r & \ldots & n_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ n_r & n_{r-1} & n_{r-2} & \ldots & n_1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_r \end{bmatrix}$$

• If M or N are 2-dimensional matrices, CONVOLVE(M,N[,X[,Y]]) constructs a matrix which contains the convolution of the discrete 2-dimensional pattern $\{m_{i,j}\}$ given by M and the discrete 2-dimensional pattern $\{n_{i,j}\}$ given by N. M is an $r_m \times c_m$ matrix, which contains the pattern values for $\{m_{i,j}\}$, i.e. $M[i, j] = m_{i,j}$. N is an $r_n \times c_n$ matrix which contains the pattern values for $\{n_{i,j}\}$ i.e. $N[i, j] = n_{i,j}$.

Let $r_o = \max(r_m, r_n)$, and $c_o = \max(c_m, c_n)$. Internally, the matrix M and/or N is extended with zeros if necessary to size $r_o \times c_o$, so we have two patterns of size $r_o \times c_o$.

The optional scalars X and Y define the extension rule for periodic extension in the row and column directions, respectively, of the patterns given by both M and N. If X or Y is omitted, it defaults to 1. There are seven options:

0 : zero extension.

The pattern is suffixed with an $r_o \times c_o$ pattern of 0's in the row direction to produce a pattern which is $r_o \times 2c_o$. Then this pattern is suffixed with an $r_o \times 2c_o$ pattern of 0's in the covlumn direction. The resulting pattern is then treated as doubly-periodic with period length $(r, c) = (2r_o, 2c_o)$.

1 : periodic extension.

If both X and Y are 1, both the patterns are treated as though they were periodic in the directions defined by the rows and columns with period lengths $(r, c) = (r_o, c_o)$. The resulting convolution pattern is doubly-periodic with period (r, c). Otherwise, if X is 1 and Y is not 1, or Y is 1 and X is not 1, then the pattern whose extension rule code is 1 is suffixed in the directions defined by the rows and columns with replicates of itself and treated as a doubly-periodic pattern in the directions defined by the rows and columns of period $(r, c) = (2r_o, 2c_o)$.

2 : bi-constant extension.

If X = 2, the M-pattern is suffixed in the column direction $\lfloor c_o/2 \rfloor$ times with the pattern in M COL c_o, and then suffixed in the column direction $\lceil c_o/2 \rceil$ times with the pattern in M COL 1. The M-pattern is then suffixed in the row direction $\lfloor r_o/2 \rfloor$ times with the pattern in M ROW r_o and then further suffixed in the row direction $\lceil r_o/2 \rceil$ times with the pattern in M COL 1.

If Y = 2, the n-pattern is suffixed in the column direction $\lfloor c_o/2 \rfloor$ times with the pattern in N COL c_o, and then suffixed in the column direction $\lceil c_o/2 \rceil$ times with the pattern in N COL 1. The N-pattern is then suffixed in the row direction with $\lfloor r_o/2 \rfloor$ times with the pattern in N ROW r_o and then further suffixed in the row direction $\lceil r_o/2 \rceil$ times with the pattern in N COL 1.

The resulting pattern is doubly-periodic with period lengths $(r, c) = (2r_o, 2c_o)$.

3 : odd replicate extension.

If X = 3, the M-pattern is suffixed in the column direction with the pattern: $\{-M$ COL $c_o, -M$ COL $(c_o-1), \ldots, -M$ COL $1\}$, and then suffixed in the row direction with the pattern $\{-M$ ROW $r_o, -M$ ROW $(r_o - 1), \ldots, -M$ ROW $1\}$.

If Y= 3, the N-pattern is suffixed in the column direction with the pattern: $\{-N$ COL $c_o, \ldots, -N$ COL $1\}$, and then suffixed in the row direction with the pattern: $\{-N$ ROW $r_o, -N$ ROW $(r_o - 1), \ldots, -N$ ROW $1\}$.

The resulting pattern is doubly-periodic with period lengths $(r, c) = (2r_o, 2c_o)$.

4 : even replicate extension.

If X = 4, the M-pattern is suffixed in the column direction with the pattern: $\{M$ COL c_o, M COL $(c_o - 1), \ldots, M$ COL $1\}$, and then suffixed in the row direction with the pattern $\{M$ ROW r_o, M ROW $(r_o - 1), \ldots, M$ ROW $1\}$.

If Y = 4, the N-pattern is suffixed in the column direction with the pattern: $\{N$ COL c_o, N COL $(c_o - 1), \ldots, N$ COL $1\}$, and then suffixed in the row direction with the pattern: $\{N$ ROW r_o, N ROW $(r_o - 1), \ldots, N$ ROW $1\}$.

The resulting pattern is doubly-periodic with period lengths $(r, c) = (2r_o, 2c_o)$.

5 : odd extension.

If X = 5, the M-pattern is suffixed in the column direction with the pattern: $\{-M$ COL $(c_o-1), -M$ COL $(c_o-2), \ldots, -M$ COL $2\}$, and then suffixed in the row direction with the pattern $\{-M$ ROW $(r_o - 1), -M$ ROW $(r_o - 2), \ldots, -M$ ROW $2\}$.

If $y = 5$, the N-pattern is suffixed in the column direction with the pattern: $\{-N$ COL $(c_o - 1), \ldots, -N$ COL $2\}$, and then suffixed in the row direction with the pattern: $\{-N$ ROW $(r_o - 1), -N$ ROW $(r_o - 2), \ldots, -N$ ROW $2\}$.

The resulting pattern is doubly-periodic with period lengths $(r, c) = (2r_o - 2, 2c_o - 2)$.

6 : even extension.

If X = 6, the M-pattern is suffixed in the column direction with the pattern: $\{M$ COL $(c_o-1), M$ COL $(c_o-2), \ldots, M$ COL $2\}$, and then suffixed in the row direction with the pattern $\{M$ ROW $(r_o - 1), M$ ROW $(r_o - 2), \ldots, M$ ROW $2\}$.

If Y = 6, the N-pattern is suffixed in the column direction with the pattern: $\{N$ COL $(c_o - 1), \ldots, N$ COL $2\}$, and then suffixed in the row direction with the pattern: $\{N$ ROW $(r_o - 1), \ldots, N$ ROW $2\}$.

The resulting pattern is doubly-periodic with period lengths $(r, c) = (2r_o - 2, 2c_o - 2)$.

Following the extension, we have two doubly-periodic patterns, each with period lengths (r, c).

Recall the definition of the convolution of the two discrete 2-dimensional (r, c) doubly-periodic patterns M and N. Then g, the convolution of M and N, denoted by $M * N$, is defined as the 2-dimensional (r, c) doubly-periodic pattern:

$$g_{p,q} = \sum_{k=1}^{r} \sum_{j=1}^{c} m_{kj} n_{(p+r+1-k) \bmod r, (q+c+1-j) \bmod c} \text{ for } 1 \le p \le r, \text{ and } 1 \le q \le c.$$

23·13. LMIN(M[,X])[6] local minima of 2D curve

LMIN(M[,X]) computes a matrix consisting of those rows of the $n \times c$ input matrix M for which the last column of M hold the local column-c minima values with threshold X. M must be a matrix. The optional scalar X defaults to zero if it is omitted.

Given a list of values $v_1, v_2, ..., v_n$, the value v_i is a threshold-t local minimum if there exists a neighborhood $v_j, v_{j+1}, \ldots, v_i, v_{i+1}, \ldots, v_h$ of v_i with $j < i < h$ such that

(1) $v_i = \min(v_j, \ldots, v_h)$

(2) $v_i < v_k$ for $k = j, \ldots, i-1$

(3) $v_k - v_i \le t$ for $k = j+1, \ldots, h-1$

(4) $v_j - v_i > t$ and $v_h - v_i > t$

23·14. LMAX(M[,X])[6] local maxima of 2D curve

LMAX(M[,X]) computes a matrix consisting of those rows of the $n \times c$ input matrix M for which the last column of M hold the local column-c maxima values with threshold X. M must be a matrix. The optional scalar X defaults to zero if it is omitted.

Given a list of values $v_1, v_2, \ldots, v_n$, the value v_i is a threshold-t local maximum if there exists a neighborhood $v_j, v_{j+1}, \ldots, v_i, v_{i+1}, \ldots, v_h$ of v_i with $j < i < h$ such that

(1) $v_i = \max(v_j, \ldots, v_h)$

(2) $v_i > v_k$ for $k = j, \ldots, i-1$

(3) $v_i - v_k \le t$ for $k = j+1, \ldots, h-1$

(4) $v_i - v_j > t$ and $v_i - v_h > t$

23·15. MMEAN(M[,X[,Y[,Z[,Q]]]])[9] moving mean

MMEAN(M[,X[,Y[,Z[,Q]]]]) computes a matrix P which is the X-point moving mean of the numbers in the matrix M. The optional arguments X, Y, and Z are made integer values by application of the floor function. The computation is done on the individual columns of M so that the result P will have the same shape as M. The remaining discussion will treat an example of M with a single column.

The optional scalar X is the width of the window within which the mean is computed. If supplied, X must be ≥ 1 or an error message is produced. If X is omitted, it defaults to 5.

When $X = 5$, $P[i]$ is computed to be the mean of the numbers in the window $\{M[i-2], M[i-1], M[i], M[i+1], M[i+2]\}$, for $i = 1, 2, \ldots, s$, where s is the number of rows in M. For some values of i, the computation of $P[i]$ may require non-existent elements of M. For example when $X = 5$, the calculation of P[1] requires M[-1] and M[0]. In such cases, the missing elements of M are found by using M[1] where elements of M with zero or negative subscripts are required and $M[s]$ for calculations requiring elements of M with subscripts greater than s.

The optional scalar Y is the window offset value. It shifts the indices of the numbers in the window. If Y is omitted, it defaults to 0 and $P[i]$ is computed from X successive numbers starting with $M[i - \lfloor X/2 \rfloor]$. If Y is supplied, the window for determining $p[i]$ extends from $M[i - \lfloor X/2 \rfloor + Y]$ to $M[i + \lfloor (X-1)/2 \rfloor + Y]$.

The optional scalar Z determines the values to be used for non-existent elements of M when they are required for a computation. If Z is omitted, it defaults to -1. The following table describes how non-existent elements of M are found for different values of Z:

< 0 : constant extension: use $M[1]$ for $M[0]$, $M[-1]$, $M[-2]$,...; use $M[s]$ for $M[s+1]$, $M[s+2]$, $M[s+3]$, ...

0 : zero extension: use 0 for all non-existent elements of M.

1 : periodic extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is treated as a periodic sequence with length s.

2 : even extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with $M[s-1]$, $M[s-2]$, ..., $M[2]$ and then treated as a periodic sequence of length $2s-2$.

3 : odd extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with $-M[s-1]$, $-M[s-2]$, ..., $-M[2]$ and then treated as a periodic sequence of length $2s-2$.

4 : even replicate extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with the sequence $M[s]$, $M[s-1]$, ..., $M[1]$, and then treated as a periodic sequence of length $2s$.

5 : odd replicate extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with the sequence $-M[s]$, $-M[s-1]$, ..., $-M[1]$, and then treated as a periodic sequence of length $2s$.

The optional matrix Q is the weight matrix. It enables the user to stress or de-emphasize window elements in the calculation of means. If supplied, Q must have X rows. If Q has one column, $P[i]$ is computed as

$$P[i] = \frac{\sum_{j=1}^{X} Q[j] M[i + j - \lfloor \frac{X+1}{2} \rfloor]}{\sum_{j=1}^{X} Q[j]} \text{ for } i = 1, 2, \ldots, s.$$

If Q is not supplied, a default weight matrix of X rows containing ones is used. If Q has more than one column, successive columns of Q are used to weight corresponding columns of M.

23·16. MMEDIAN(M[,X[,Y[,Z[,Q]]]])[9] moving median

MMEDIAN(M[,X[,Y[,Z[,Q]]]]) computes a matrix P which is the X-point moving median of the data in the matrix M. The optional arguments X, Y, and Z are made integer values by application of the floor function. The computation is done on the individual columns of M so that P will have the same shape as M. The remaining discussion will treat an example of M with a single column.

By default, $P[i]$ is computed to be the median of the numbers in the window $\{M[i-2], M[i-1], M[i], M[i+1], M[i+2]\}$, for $i = 1, 2, \ldots, s$, where s is the number of rows in M. For some values of i, the computation of $P[i]$ may require non-existent elements of M. For example, the calculation of P[1] requires M[-1] and M[0]. In such cases, the missing elements of M are found by using M[1] where elements of M with zero or negative subscripts are required and $M[s]$ for calculations requiring elements of M with subscripts greater than s.

The optional scalar X is the width of the window within which the moving median is computed. If supplied, X must be ≥ 1 or an error message is produced. If X is omitted, it defaults to 5.

The optional scalar Y is the window offset value. It shifts the indices of the numbers in the window. If Y is omitted, it defaults to 0 and P[i] is computed from X successive numbers starting with $M[i - \lfloor X/2 \rfloor]$. If Y is supplied, the window for determining P[i] extends from $M[i - \lfloor X/2 \rfloor + Y]$ to $M[i + \lfloor (X-1)/2 \rfloor + Y]$.

The optional scalar Z determines the values to be used for non-existent elements of M when they are required for a computation. If Z is omitted, it defaults to -1. The following table describes how non-existent elements of M are found for different values of Z:

< 0 : constant extension: use $M[1]$ for $M[0]$, $M[-1]$, $M[-2]$,...; use $M[s]$ for $M[s+1]$, $M[s+2]$, $M[s+3]$,...

0 : zero extension: use 0 for all non-existent elements of M.

1 : periodic extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is treated as a periodic sequence with length s.

2 : even extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with $M[s-1]$, $M[s-2]$, ..., $M[2]$ and then treated as a periodic sequence of length $2s-2$.

3 : odd extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with $-M[s-1]$, $-M[s-2]$, ..., $-M[2]$ and then treated as a periodic sequence of length $2s-2$.

4 : even replicate extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with the sequence $M[s]$, $M[s-1]$, ..., $M[1]$, and then treated as a periodic sequence of length $2s$.

5 : odd replicate extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with the sequence $-M[s]$, $-M[s-1]$, ..., $-M[1]$, and then treated as a periodic sequence of length $2s$.

The optional matrix Q is the weight matrix. Q must have at least X rows. The values Q ROW 1:X COL 1:MOD($j-1$,NROWS(Q)) are normalized so that they sum to one and then are used to weight the values in each window of size X of the data in M COL j, for $j = 1, 2, \ldots,$ NCOLS(M). These normalized weights are used as probabilities in computing expectations.

If Q is not supplied, a default weight matrix of X rows containing ones is used. If Q has more than one column, successive columns of Q are used to weight corresponding columns of M.

23·17. MVAR(M[,X[,Y[,Z[,Q]]]])[9] moving variance

MVAR(M[,X[,Y[,Z[,Q]]]]) computes a matrix P which is the X-point moving variance of the data in the matrix M. The optional arguments X, Y, and Z are made integer values by application of the floor function. The computation is done on the individual columns of M so that the result P will have the same shape as M. The remaining discussion will treat an example of M with a single column.

By default, $P[i]$ is computed to be the variance of the numbers in the window $\{M[i-2], M[i-1], M[i], M[i+1], M[i+2]\}$, for $i = 1, 2, \ldots, s$, where s is the number of rows in M. For some values of i, the computation of $P[i]$ may require non-existent elements of M. For example, the calculation of P[1] requires M[-1] and M[0]. In such cases, the missing elements of M are found by using M[1] where elements of M with zero or negative subscripts are required and $M[s]$ for calculations requiring elements of M with subscripts greater than s.

The optional scalar X is the width of the window within which the moving variance is computed. If supplied, X must be ≥ 1 or an error message is produced. If X is omitted, it defaults to 5.

The optional scalar Y is the window offset value. It shifts the indices of the numbers in the window. If Y is omitted, it defaults to 0 and $P[i]$ is computed from X successive numbers starting with

$M[i - \lfloor X/2 \rfloor]$. If Y is supplied, the window for determining $P[i]$ extends from $M[i - \lfloor X/2 \rfloor + Y]$ to $M[i + \lfloor (X-1)/2 \rfloor + Y]$.

The optional scalar Z determines the values to be used for non-existent elements of M when they are required for a computation. If Z is omitted, it defaults to -1. The following table describes how non-existent elements of M are found for different values of Z:

< 0 : constant extension: use $M[1]$ for $M[0]$, $M[-1]$, $M[-2]$,...; use $M[s]$ for $M[s+1]$, $M[s+2]$, $M[s+3]$,....

0 : zero extension: use 0 for all non-existent elements of M.

1 : periodic extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is treated as a periodic sequence with length s.

2 : even extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with $M[s-1]$, $M[s-2]$, ..., $M[2]$ and then treated as a periodic sequence of length $2s-2$.

3 : odd extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with $-M[s-1]$, $-M[s-2]$, ..., $-M[2]$ and then treated as a periodic sequence of length $2s-2$.

4 : even replicate extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with the sequence $M[s]$, $M[s-1]$, ..., $M[1]$, and then treated as a periodic sequence of length $2s$.

5 : odd replicate extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with the sequence $-M[s]$, $-M[s-1]$, ..., $-M[1]$, and then treated as a periodic sequence of length $2s$.

The optional matrix Q is the weight matrix. Q must have at least X rows. The values Q ROW 1:X COL 1:MOD($j-1$,NROWS(Q)) are normalized so that they sum to one and then are used to weight the values in each window of size X of the data in M COL j, for $j = 1, 2, \ldots,$ NCOLS(M). These normalized weights are used as probabilities in computing expectations.

If Q is not supplied, a default weight matrix of X rows containing ones is used. If Q has more than one column, successive columns of Q are used to weight corresponding columns of M.

23·18. MSTDDEV(M[,X[,Y[,Z[,Q]]]])[9] moving standard deviation

MSTDDEV(M[,X[,Y[,Z[,Q]]]]) computes a matrix P which is the X-point moving standard deviation of the data in the matrix M. The optional arguments X, Y, and Z are made integer values by application of the floor function. The computation is done on the individual columns of M so that the result P will have the same shape as M. The remaining discussion will treat an example of M with a single column.

By default, P[i] is computed to be the standard deviation of the numbers in the window $\{M[i-2], M[i-1], M[i], M[i+1], M[i+2]\}$, for $i = 1, 2, \ldots, s$, where s is the number of rows in M. For some values of i, the computation of $P[i]$ may require non-existent elements of M. For example, the calculation of P[1] requires M[-1] and M[0]. In such cases, the missing elements of M are found by using M[1] where elements of M with zero or negative subscripts are required and $M[s]$ for calculations requiring elements of M with subscripts greater than s.

The optional scalar X is the width of the window within which the moving standard deviation is computed. If supplied, X must be ≥ 1 or an error message is produced. If X is omitted, it defaults to 5.

The optional scalar Y is the window offset value. It shifts the indices of the numbers in the window. If Y is omitted, it defaults to 0 and $P[i]$ is computed from X successive numbers starting with $M[i - \lfloor X/2 \rfloor]$. If Y is supplied, the window for determining $P[i]$ extends from $M[i - \lfloor X/2 \rfloor + Y]$ to $M[i + \lfloor (X-1)/2 \rfloor + Y]$.

The optional scalar Z determines the values to be used for non-existent elements of M when they are required for a computation. If Z is omitted, it defaults to -1. The following table describes how non-existent elements of M are found for different values of Z:

< 0 : constant extension: use $M[1]$ for $M[0]$, $M[-1]$, $M[-2]$,...; use $M[s]$ for $M[s+1]$, $M[s+2]$, $M[s+3]$,....

0 : zero extension: use 0 for all non-existent elements of M.

1 : periodic extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is treated as a periodic sequence with length s.

2 : even extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with $M[s-1]$, $M[s-2]$, ..., $M[2]$ and then treated as a periodic sequence of length $2s-2$.

3 : odd extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with $-M[s-1]$, $-M[s-2]$, ..., $-M[2]$ and then treated as a periodic sequence of length $2s-2$.

4 : even replicate extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with the sequence $M[s]$, $M[s-1]$, ..., $M[1]$, and then treated as a periodic sequence of length $2s$.

5 : odd replicate extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with the sequence $-M[s]$, $-M[s-1]$, ..., $-M[1]$, and then treated as a periodic sequence of length $2s$.

The optional matrix Q is the weight matrix. Q must have at least X rows. The values Q ROW 1:X COL 1:MOD($j-1$,NROWS(Q)) are normalized so that they sum to one and then are used to weight the values in each window of size X of the data in M COL j, for $j = 1, 2, \ldots,$ NCOLS(M). These normalized weights are used as probabilities in computing expectations.

If Q is not supplied, a default weight matrix of X rows containing ones is used. If Q has more than one column, successive columns of Q are used to weight corresponding columns of M.

23·19. MAVDEV(M[,X[,Y[,Z[,Q]]]])[9] moving absolute average deviation

MAVDEV(M[,X[,Y[,Z[,Q]]]]) computes a matrix P which is the X-point moving average deviation of the data in the matrix M. The optional arguments X, Y, and Z are made integer values by application of the floor function. The computation is done on the individual columns of M so that the result P will have the same shape as M. The remaining discussion will treat an example of M with a single column.

By default, $P[i]$ is computed to be the average deviation of the numbers in the window $\{M[i-2], M[i-1], M[i], M[i+1], M[i+2]\}$, for $i = 1, 2, \ldots, s$, where s is the number of rows in M. For some values of i, the computation of $P[i]$ may require non-existent elements of M. For example, the calculation of P[1] requires M[- 1] and M[0]. In such cases, the missing elements of M are found by using M[1] where elements of M with zero or negative subscripts are required and $M[s]$ for calculations requiring elements of M with subscripts greater than s.

The optional scalar X is the width of the window within which the moving average deviation is computed. If supplied, X must be ≥ 1 or an error message is produced. If X is omitted, it defaults to 5.

The optional scalar Y is the window offset value. It shifts the indices of the numbers in the window. If Y is omitted, it defaults to 0 and $P[i]$ is computed from X successive numbers starting with $M[i - \lfloor X/2 \rfloor]$. If Y is supplied, the window for determining $P[i]$ extends from $M[i - \lfloor X/2 \rfloor + Y]$ to $M[i + \lfloor (X-1)/2 \rfloor + Y]$.

The optional scalar z determines the values to be used for non- existent elements of M when they are required for a computation. If Z is omitted, it defaults to -1. The following table describes how non-existent elements of M are found for different values of Z:

< 0 : constant extension: use $M[1]$ for $M[0]$, $M[-1]$, $M[-2]$,...; use $M[s]$ for $M[s+1]$, $M[s+2]$, $M[s+3]$,....

0 : zero extension: use 0 for all non-existent elements of M.

1 : periodic extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is treated as a periodic sequence with length s.

2 : even extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with $M[s-1]$, $M[s-2]$, ..., $M[2]$ and then treated as a periodic sequence of length $2s-2$.

3 : odd extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with $-M[s-1]$, $-M[s-2]$, ..., $-M[2]$ and then treated as a periodic sequence of length $2s-2$.

4 : even replicate extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with the sequence $M[s]$, $M[s-1]$, ..., $M[1]$, and then treated as a periodic sequence of length $2s$.

5 : odd replicate extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with the sequence $-M[s]$, $-M[s-1]$, ..., $-M[1]$, and then treated as a periodic sequence of length $2s$.

The optional matrix Q is the weight matrix. Q must have at least X rows. The values Q ROW 1:X COL 1:MOD($j-1$,NROWS(Q)) are normalized so that they sum to one and then are used to weight the values in each window of size X of the data in M COL j, for $j = 1, 2, \ldots,$ NCOLS(M). These normalized weights are used as probabilities in computing expectations.

If Q is not supplied, a default weight matrix of X rows containing ones is used. If Q has more than one column, successive columns of Q are used to weight corresponding columns of M.

23·20. MQUANTILE(M[,H[,X[,Y,[Z,[Q]]]]])[9] moving quantile

MQUANTILE(M[,H[,X[,Y[,Z[,Q]]]]]) computes a matrix P which is the X-point moving H-quantile value of the data in the matrix M weighted by the weights in Q. The optional arguments X, Y, and Z are made integer values by application of the floor function. The computation is done separately on each of the individual columns of M so that the result P will have the same shape as M. The remaining discussion will treat an example of M with a single column.

By default, $P[i]$ is computed to be the .5-quantile value for the numbers in the window $\{M[i-2], M[i-1], M[i], M[i+1], M[i+2]\}$ for $i = 1, 2, \ldots, s$, where s is the number of rows in M. For some values of i, the computation of $P[i]$ may require non-existent elements of M. For example, the calculation of P[1] requires M[-1] and M[0]. In such cases, the missing elements of M are found by using M[1] where elements of M with zero or negative subscripts are required and $M[s]$ for calculations requiring elements of M with subscripts greater than s.

The optional scalar argument H is the desired quantile level value in [0,1]. H is .5 by default. If H is outside the interval [0,1], an error message is produced.

The optional scalar argument X is the width of the window within which the moving H-quantile is computed. X must be >= 1 or an error message is produced. The default value of X is 5.

The optional scalar argument Y is the window offset value. It shifts the indices of the numbers in the window. If Y is omitted, it defaults to 0 and $P[i]$ is computed from X successive numbers starting with $M[i-\lfloor X/2 \rfloor]$. If Y is supplied, the window for determining $P[i]$ extends from $M[i-\lfloor X/2 \rfloor + Y]$ to $M[i+\lfloor (X-1)/2 \rfloor + Y]$.

The optional scalar Z determines the values to be used for non-existent elements of M when they are required for a computation. If Z is omitted, it defaults to -1. The following table describes how non-existent elements of M are found for different values of Z:

< 0 : constant extension: use $M[1]$ for $M[0]$, $M[-1]$, $M[-2]$,...; use $M[s]$ for $M[s+1]$, $M[s+2]$, $M[s+3]$,....

0 : zero extension: use 0 for all non-existent elements of M.

1 : periodic extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is treated as a periodic sequence with length s.

2 : even extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with $M[s-1]$, $M[s-2]$, ..., $M[2]$ and then treated as a periodic sequence of length $2s-2$.

3 : odd extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with $-M[s-1]$, $-M[s-2]$, ..., $-M[2]$ and then treated as a periodic sequence of length $2s-2$.

4 : even replicate extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with the sequence $M[s]$, $M[s-1]$, ..., $M[1]$, and then treated as a periodic sequence of length $2s$.

5 : odd replicate extension: the sequence $M[1]$, $M[2]$, ..., $M[s]$ is suffixed with the sequence $-M[s]$, $-M[s-1]$, ..., $-M[1]$, and then treated as a periodic sequence of length $2s$.

The optionally supplied weight matrix Q is a matrix with X rows and c columns. The values in Q col $(((j-1) \bmod c)+1)$ are normalized to sum to 1 and used to weigh the values in M COL j when computing expectations involving them.

If Q is not supplied, a default weight matrix of X rows containing ones is used. If Q has more than one column, successive columns of Q are used to weight corresponding columns of M.

23·21. MONOT(M[,D])[9] Monotonic smoothing filter

MONOT(M[,D]) Monotonic Smoothing

M is an $n \times 2$ matrix of data points with M COL 1 in increasing order. The points in M are assumed to be noisy samples of a monotonic (increasing or decreasing) function. Note, however, that due to the assumed noise, the data points in M need not be monotonic.

The optional argument D is a scalar direction code. $D \geq 0$ means "increasing", and $D < 0$ means "decreasing". If D is not given the default value used is 1.

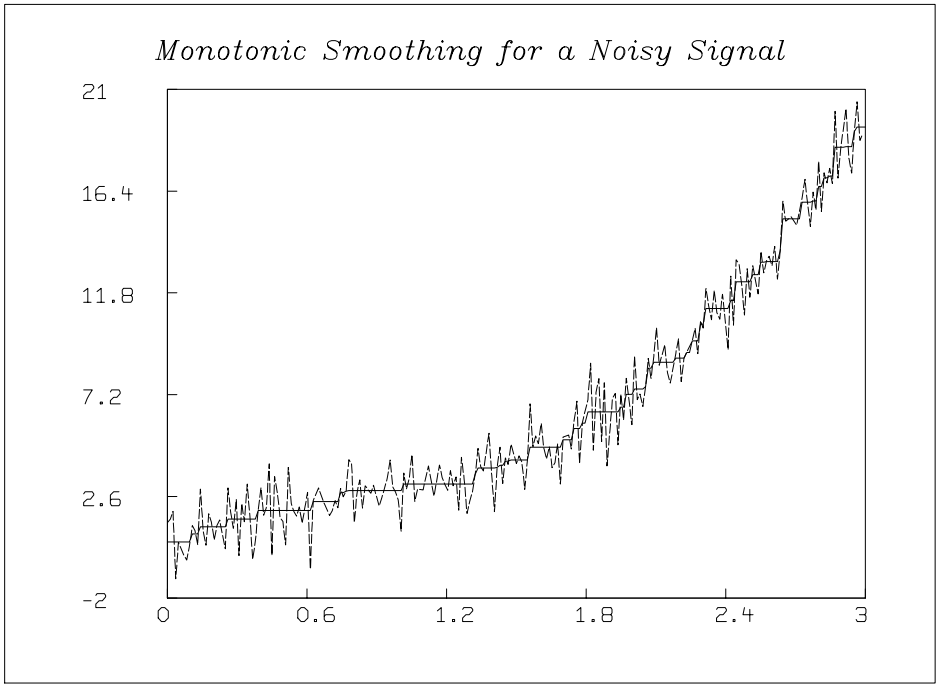
Let the points in the matrix M be denoted by the sequence of points $(x_1, y_1), \ldots, (x_n, y_n)$ with $x_1 < x_2 < \ldots < x_n$; these points are assumed to be noisy samples from the graph of a "hidden" *monotonic* function m, with $y_i = m(x_i) + \epsilon_i$. The errors $\epsilon_1, \ldots, \epsilon_n$ generally cause the sequence of values $y_1, y_2, \ldots, y_n$ to fail to be monotonic.

We may adjust the values $y_1, y_2, \ldots, y_n$ by adding various positive or negative numbers $s_1, s_2, \ldots, s_n$ to define $z_i = y_i + s_i$ for $1 \leq i \leq n$. If $D \geq 0$, we will choose $s_1, s_2, \ldots, s_n$ such that $y_1 + s_1 \leq y_2 + s_2 \leq \ldots \leq y_n + s_n$, *i.e.*, so that the points $(x_1, z_1), \ldots, (x_n, z_n)$ are monotonically increasing, and if $D < 0$, we will choose $s_1, s_2, \ldots, s_n$ such that $y_1 + s_1 \geq y_2 + s_2 \geq \ldots \geq y_n + s_n$, *i.e.*, so that the points $(x_1, z_1), \ldots, (x_n, z_n)$ are monotonically decreasing.

The result of MONOT is an $n \times 2$ matrix R containing the *monotonized* data points $(x_1, z_1), \ldots, (x_n, z_n)$. This is a form of smoothing of the input data points that enforces monotonicity.

The values $s_1, s_2, \ldots, s_n$ are computed so as to minimize $s_1^2 + s_2^2 + \ldots + s_n^2$, subject, of course, to the $n-1$ constraints $y_1 + s_1 \leq y_2 + s_2$, $y_2 + s_2 \leq y_3 + s_3$, ..., $y_{n-1} + s_{n-1} \leq y_n + s_n$ (when $D \geq 0$) or $y_1 + s_1 \geq y_2 + s_2$, $y_2 + s_2 \geq y_3 + s_3$, ..., $y_{n-1} + s_{n-1} \geq y_n + s_n$ (when $D < 0$).

The figure below shows a noisy signal $(x_1, y_1), \ldots, (x_n, y_n)$ with $n = 255$, together with the corresponding monotonized signal.



The mean of the values $y_1, \ldots, y_n$ and the mean of the computed values $z_1, \ldots, z_n$ are always identical. Therefore the "bias" $\sum_{1 \le i \le n}(y_i - m(x_i)) = \sum_{1 \le i \le n} \epsilon_i$ in the original data values $y_1, \ldots, y_n$ is exactly preserved in the monotonized values $z_1, \ldots, z_n$.

The variance of the errors $z_1 - m(x_1), \ldots, z_n - m(x_n)$ is generally reduced in comparison to the variance of the original errors $\epsilon_1, \ldots, \epsilon_n$. Thus the monotonizing transformation is a filter which passes the d.c. term and generally reduces the power at the other frequencies of the error signal.

23·22. SMOOTHSPLINE(M[,V[,R[,G,[D]]]])[9] optimal smoothing spline

SMOOTHSPLINE(M[,V[,R[,G[,D]]]])

M is an $n \times 2$ matrix of planar functional data points (*i.e.*, drawn, with error, from the graph of a function) for which an optimal smoothing spline or its derivative or its integral is to be computed.

V is an $m \times 1$ vector of x-values whereat we want points of our optimal smoothing spline to be generated and returned. If V is not given, M col 1 will be used by default.

R is a scalar specifying the smoothing factor denoted by ρ below. If R≥ 0, then R is the smoothing factor to be used. If R$= -1$, then the smoothing factor is estimated by requiring our optimal smoothing spline to fit a weighted moving mean of the sequence of values in M col 2. If R$= -2$, then the smoothing factor is estimated by requiring our optimal smoothing spline to fit a weighted moving median of the sequence of values in M col 2. the window-size of the moving mean or moving median is taken as $\max(5, 2\lfloor n/20 \rfloor + 1)$. If R is not given, the value -1 is used by default.

G is an $n \times 1$ vector of weight values denoted below by $\lambda_1, \ldots, \lambda_n$. Generally G[$i$] should be chosen as the reciprocal of an estimate of the variance of the random variable of which the data value M[$i, 2$] is a sample; the MLAB function The MLAB operator EWT can be used to compute such a vector of weight-values. If G is not given, the vector of ones $1 \wedge \wedge n$ is used by default.

D is a scalar specifying the derivative/integral-switch control value. If D$= 0$, the requested points on the requested optimal smoothing spline are returned. Otherwise, if $-1 \le$D< 0, the requested points of the derivative of the requested optimal smoothing spline are returned, if D< -1, the requested

points of the second derivative of the requested optimal smoothing spline are returned, and, if D> 0, the requested points of the integral of the requested optimal smoothing spline are returned; the value of the integral of the optimal smoothing spline g correponding to an x-value v is $\int_{M[1,1]\leq x\leq v} g(x)dx$.

The result is an $m \times 2$ matrix S such that S col 1 = V, and S$[i, 2]$ is the value of the requested optimal smoothing spline for the data in M computed at the value V$[i]$, or of its derivative, if D< 0, or of its integral, if D> 0.

Suppose we have a sequence of points that are known to be noisy samples from the graph of a "hidden" function f We may seek some form of smooth-curve that appears to fit the data without regard for its functional form. Such a non-parametric regression model may be used for descriptive purposes, or sometimes even, with due care, for predictive purposes. For example, such a non-parametric regression curve can be used as an "input" function in a system of algebraic or differential equations for which we need to provide some suitable input function in order for us to obtain their solution and to use them in subsequent modeling. Sometimes we are seeking the derivative of the non-parametric model of our data, and we require that the non-parametric model we pick be smooth enough to have an appropriately nice derivative function for our purposes.

There are many smoothing algorithms such as weighted moving means or kernel smoothing that may be employed to obtain a non-parametric regression curve. One of the best approaches in many circumstances is the use of a so-called optimal smoothing cubic spline curve. An optimal smoothing cubic spline has several advantages. First, it is a piecewise-cubic polynomial function which is twice continuously-differentiable at its join-points, and hence its form can be differentiated or integrated directly when desired. Second, it can be adjusted to have any desired reduction in the amount of observed oscillatory variation by appropriately choosing the so-called smoothing factor associated with the optimal smoothing spline objective function.

Constructing an optimal smoothing spline is a somewhat elaborate undertaking. We consider the data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, with $x_1 < x_2 < \ldots < x_n$, to be points of R^2 sampled with error from the graph of some unknown function $f : R \to R$ such that $y_j = f(x_j)+\epsilon_j$ for $j = 1, \ldots, n$. If the form of the function f were known, except for some unknown parameter values, we could then determine those values by the least-squares method where the desired parameter values are those parameter values which minimize $E = \sum_{j=1}^{n} \lambda_j (y_j - f(x_j))^2$ where $\lambda_1, \ldots, \lambda_n$ are weight values which are inversely proportional to the variances of the errors in the points. When the function f is not known, we may seek a smooth function g which "fits" the data points without regard to any particular functional form. A polynomial or a cubic spline can fit the data exactly, but we may obtain a less-oscillatory approximating function if we take g to be the function which minimizes the functional $E(f) + \rho J(f)$ over all suitable functions f, where ρ is a positive constant and

$$E(f) = \sum_{j=1}^{n} \lambda_j (y_j - f(x_j))^2, \text{ and } J(f) = \int_{x_1}^{x_n} \mid f''(t) \mid^2 dt.$$

Here $J(f)$ acts as a "penalty" term which favors functions with small second derivatives. The effect of J can be increased or diminished by suitably choosing the multiplier ρ.

It is a famous result due to Issac Schoenberg that the function that minimizes $J(f)$ among all functions that interpolate a given sequence of points $(x_1, p_1), \ldots, (x_n, p_n)$ is the *natural global cubic spline* for the points $(x_1, p_1), (x_2, p_2), \ldots, (x_n, p_n)$. This is the function g defined on $[x_1, x_n]$ such that:

(1) $g(x) = g_i(x - x_i)$ for $x \in [x_i, x_{i+1}]$ where g_i is a cubic polynomial function such that $g_i(0) = p_i$ and $g_i(x_{i+1} - x_i) = p_{i+1}$ for $i = 1, \ldots, n$,

(2) The functions g, g', and g'' are continuous on $[x_1, x_n]$.

(3) $g''(x_1) = 0$ and $g''(x_n) = 0$.

The piecewise-cubic polynomial g is determined by the values $x_1, \ldots, x_n, p_1, \ldots, p_n$ and $s_1, \ldots, s_n$ where the slope $s_i = g'(0) = g'_{i-1}(x_i - x_{i-1}) = g'_i(0)$. The abscissa values $x_1, \ldots, x_n$ are called the

knot-values of the spline g. For a natural global cubic spline, the slopes $s_1, \ldots, s_n$ are determined by conditions (2) and (3) in terms of the independent values $x_1, \ldots, x_n, p_1, \ldots, p_n$.

Now suppose g is a function for which $E(g)+\rho J(g)$ is minimal. Let $p_j := g(x_j) \in \mathcal{R}$ for $j = 1, \ldots, n$. Whatever function we have for g, only its values at $x_1, x_2, \ldots, x_n$ affect the value $E(g)$. And whatever the values $p_1, \ldots, p_n$ are, the corresponding function g for which $E(g)+\rho J(g)$ is minimal is a natural global cubic spline that interpolates the points $(x_1, p_1), \ldots, (x_n, p_n)$, since this choice minimizes J while having no effect on E.

Thus we need only determine the scalar values $p_1, \ldots, p_n$ that define a corresponding natural global cubic spline g with the knot-values $x_1, \ldots, x_n$, such that $E(g) + \rho J(g)$ is minimal over all choices of the values $p_1, \ldots, p_n$. The spline g is then our desired smooth approximating function for the points $(x_1, y_1), \ldots, (x_n, y_n)$ with the weights $\lambda_1, \ldots, \lambda_n$. The natural global cubic spline g is called the *optimal smoothing spline* for the points $(x_1, y_1), \ldots, (x_n, y_n)$ with the weights $\lambda_1, \ldots, \lambda_n$ and the smoothing parameter ρ.

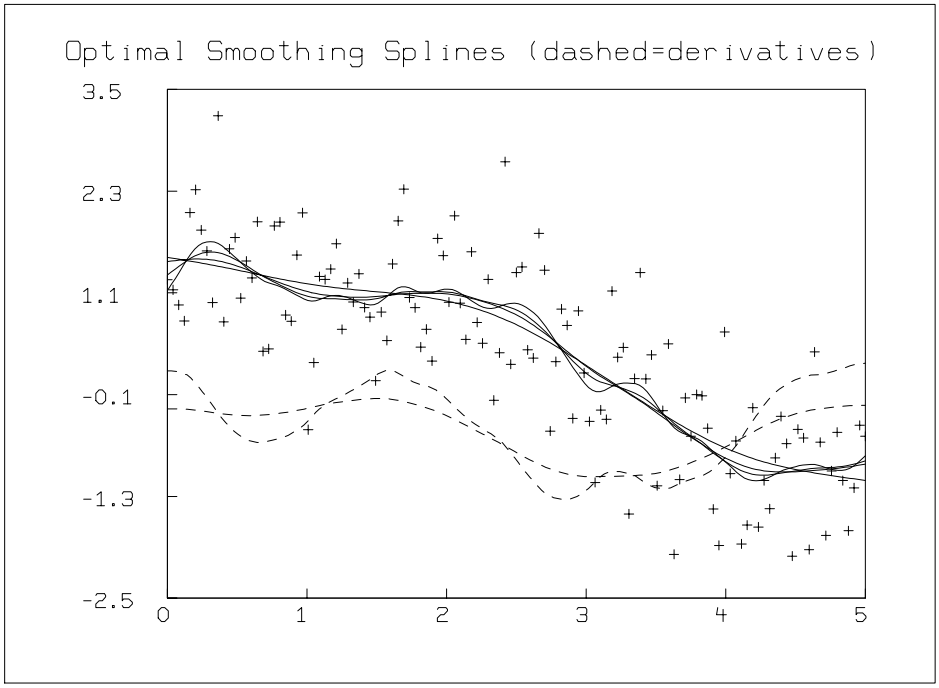
As mentioned above, the positive constant ρ determines the amount of oscillation potentially exhibited in the optimal smoothing spline g. When ρ approaches 0, then p_j approaches y_j, and when ρ approaches ∞, then p_j approaches $b(x_j)$ where the graph of the function b is the best-fitting line for the points (x_1, y_1), ..., (x_n, y_n) in the least-squares sense, with the weights $\lambda_1, \ldots, \lambda_n$. In order to choose a value for ρ, we may minimize $S(\rho) := \sum_{1 \le i \le n} (\tilde{y}_i - g(x_i; \rho))^2$ where $\tilde{y}_1, \ldots, \tilde{y}_n$ are values generated by a weighted moving mean or moving median smoothing process. This method of determining ρ generally results in a value which corresponds to a smoothed form of the points produced by the surrogate smoothing process. If the surrogate smoothing process is locally-focused and produces a relatively large amount of local variation, then ρ will generally be a relatively small value.

An example of the use of SMOOTHSPLINE is given below. First we generate 125 points of a noisy signal, and then we compute and graph various optimal smoothing splines and, for two of them, the corresponding derivative functions are computed and graphed as well.

```
fct f(x)=sin(x)+exp(-x)+.75*normran(0)
m=points(f,0:5!125)
v=0:5!180

s=smoothspline(m,v)
s1=smoothspline(m,v,.02)
s2=smoothspline(m,v,.1)
s3=smoothspline(m,v,1)
s2d=smoothspline(m,v,.1,1^^nrows(m),-1)
s3d=smoothspline(m,v,1,1^^nrows(m),-1)

draw m pt crosspt ptsize .01 lt none
draw s color red
draw s1 color orange
draw s2 color yellow
draw s3 color green
draw s2d lt dashed color yellow
draw s3d lt dashed color green
top title "Optimal Smoothing Splines (dashed=derivatives)"
view
```



24. Functions of Complex Arguments

The functions described in this section perform computations on complex numbers. Since there is no intrinsic complex data type in MLAB, complex numbers are represented in MLAB by matrices. The following convention is used: let z be a complex number such that $z = x + iy$. Then, either a 1-column (1×2) or a 1-row (2×1) matrix Z may be used to represent z. Either Z[1,1] = x, and Z[2,1] = y, or Z[1,1] = x and Z[1,2] = y. In order to use the ON operator with these functions, however, the matrix of arguments must have two columns for each complex argument. Those functions which return a complex result $w = u + iv$ produce a 1×2 matrix W, such that W[1,1] = u and W[1,2] = v.

Many of the real mathematical functions in MLAB may also be evaluated for complex arguments using the above convention. The functions for which this facility is available are listed below.

LOG(Z)	complex natural logarithm
EXP(Z)	complex exponential function
SQRT(Z)	complex square root
SIN(Z)	complex trigonometric sine
COS(Z)	complex trigonometric cosine
TAN(Z)	complex trigonometric tangent
COTAN(Z)	complex trigonometric cotangent
SEC(Z)	complex trigonometric secant
COSEC(Z)	complex trigonometric cosecant
SINH(Z)	complex hyperbolic sine
COSH(Z)	complex hyperbolic cosine
TANH(Z)	complex hyperbolic tangent
COTANH(Z)	complex hyperbolic cotangent
SECH(Z)	complex hyperbolic secant
COSECH(Z)	complex hyperbolic cosecant
ASIN(Z)	complex inverse trigonometric sine
ACOS(Z)	complex inverse trignometric cosine
ATAN(Z)	complex inverse trigonometric tangent
ASINH(Z)	complex inverse hyperbolic sine
ACOSH(Z)	complex inverse hyperbolic cosine
ASECH(Z)	complex inverse hyperbolic secant
ACOSECH(Z)	complex inverse hyperbolic cosecant
ACOTANH(Z)	complex inverse hyperbolic cotangent
GAMMA(Z)	complex gamma function
LOGGAMMA(Z)	complex loggamma function
DIGAMMA(Z)	complex digamma function

For convenience, the functions in this section generalize to apply to vectors of complex numbers $z_j = x_j + iy_j$, for $j = 1, 2, \ldots, n$. These vectors are represented as 2-column matrices. In this case the $n \times 2$ matrix Z represents the list of complex numbers $z_1, z_2, \ldots, z_n$, such that $Z[j, 1] = x_j$, and $Z[j, 2] = y_j$. The function is applied successively to the rows of Z to produce a result which, if it is necessarily real, will be a 1-column matrix representing the list of scalar outcomes for each row. If the result is generally complex, a 2-column matrix will be produced, with the interpretation given above.

For those functions, such as CPROD(Z,W), which take two complex arguments, if NROWS(Z) $\neq$ NROWS(W), the shorter of the two vectors is implicitly cyclicly extended. This is convenient when one of the operands is a single complex number and the other is a vector of complex numbers.

24·1. CPROD(Z,W)^0 complex multiplication

CPROD(Z,W) computes the complex product of one or more complex numbers. Z and W must be matrices representing one or more complex numbers as decscribed above. If either of Z or W have only a single column, it is interpreted as a vector of real numbers, for which the imaginary part is zero.

If Z and W each specify a single complex number, z and w, respectively, the result is a 1-row matrix which is the complex product zw. Otherwise the result is a 2-column matrix V such that V ROW i =CPROD(Z ROW i, W ROW i).

CPROD is useful in developing digital filters using the DFT functions. The CDIV operator is available for complex division. Addition and subtraction of complex vectors in this format (i.e. as 2-column matrices) is directly performed by the "+" and "-" operators.

24·2. CDIV(Z,W)^0 complex division

CDIV(Z,W) computes the complex quotient of one or more complex numbers. Z and W must be matrices representing one or more complex numbers as described above. If either of Z or W have only a single column, it is interpreted as a vector of real numbers, for which the imaginary part is zero.

If Z and W each specify a single complex number, z and w, respectively, the result is a 1-row matrix which is the complex quotient z/w. Otherwise the result is a 2-column matrix V such that V ROW i =CDIV(Z ROW i, W ROW i).

CDIV is useful in developing digital filters using the DFT functions. The CPROD operator is available for complex multiplication. Addition and subtraction of complex vectors in this format is directly performed by the "+" and "-" operators.

24·3. CPR(Z)^0 polar-to-rectangular conversion

CPR(Z) converts one or more complex numbers from polar representation to rectangular representation. Z must be a matrix representing one or more complex numbers in polar form, i.e. Z COL 1 holds the modulus, and Z COL 2 holds the phase angle. If Z specifies a single complex number, z, the result is a 1-row matrix V representing the complex number in the standard form (V[1,1] is the real part and V[1,2] is the imaginary part. Otherwise the result is a 2-column matrix V such that V[i] = CPR(Z ROW i).

Recall the polar and rectangular representations of a complex number. Let $z = x + iy$ be a complex number. z may also be written in the form $re^{i\theta}$, where r is the modulus of z, and θ is the phase angle of z. Then $r = \sqrt{x^2 + y^2}$ and $\theta = \arctan(y/x)$. Inverting these relationships, $x = r\sin(\theta)$, and $y = r\cos(\theta)$. The inverse of CPR is CRP. Another use of CPR is to prepare data in polar form, i.e. as a 2-column matrix of (r, θ)-coordinate pairs, for plotting.

24·4. CRP(Z)^0 rectangular-to-polar conversion

CRP(Z) converts one or more complex numbers from rectangular representation to polar representation. Z must be a matrix representing one or more complex numbers. If Z specifies a single complex number, z, the result is a 1-row matrix V representing the complex number in polar form, i.e. V COL 1 holds the modulus, and V COL 2 holds the phase angle Otherwise the result is a 2-column matrix V such that V[i] =CRP(Z ROW i).

Recall the polar and rectangular representations of a complex number Let $z = x + iy$ be a complex number. z may also be written in the form $re^{i\theta}$, where r is the modulus of z, and θ is the phase angle of z. Then $r = \sqrt{x^2 + y^2}$ and $\theta = \arctan(y/x)$. Inverting these relationships, $x = r\sin(\theta)$, and $y = r\cos(\theta)$. The inverse of CRP is CPR.

24·5. CPOW(Z,W)^0 complex exponentiation

CPOW(Z,W) computes the complex exponentiation of one or more pairs of complex numbers. Z and W must both be matrices representing one or more complex numbers. Z must not be the complex number zero, otherwise zero will be returned and a warning message will be produced. If Z and W each specify a single complex number, z and w, respectively, the result is a 1-row matrix which is the complex power: z^w. Otherwise the result is a 2-column matrix V such that V ROW i = CPOW (Z ROW i, W ROW i).

Recall the definition of $z^w = e^{w\log(z)}$. There is a singularity at $z = 0$ and a branch cut along the positive real axis. The resulting complex value of z^w is always taken to lie on the principal branch of the function.

---

25. Linear Algebra Computations

25·1. DOT(M,N)[4] scalar product of vectors

DOT(M,N) computes a scalar which is the scalar product (dot product) of two vectors, $M \cdot N$. M and N must be matrices, ordinarily one row or one column. If M or N are not one-dimensional, they are implicitly reshaped to be one dimensional, in row-major order. DOT produces a scalar result. Recall the definition:

$$DOT(M,N) = \sum_{i=1}^{n} M[i]N[i], \text{ where } n = \min(\text{NROWS}(M), \text{NROWS}(N)).$$

25·2. DET(M)[4] matrix determinant

DET(M) computes a scalar which is the determinant of a square matrix by using LU decomposition. M must be a square matrix. If M is not square, DET(M) returns zero and produces a warning message.

Example:

Let $M = \begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 0 \\ 7 & 8 & 9 \end{bmatrix}$. Then DET(M) = -24.

25·3. TRACE(M)[4] matrix trace

TRACE(M) computes a scalar which is the trace of a matrix. M must be a matrix.

Recall that the trace of a matrix is the sum of its diagonal elements.

25·4. LINEQ(M,N)[4] solve linear equations

LINEQ(M,N) constructs a matrix which contains the solution to a system of linear equations $MX = N$ by LU decomposition and back-substitution. M must be a square, $n \times n$ matrix. N must be an $n \times p$ matrix (where p is the number of right hand sides to be solved). M and N must have the same number of rows. The result is an $n \times p$ matrix, X, whose columns are p solutions to the p linear systems M (X COL j) = N COL j, for $j = 1, \ldots, p$ $MX = N$. If M is singular, an error will occur. To solve $MX = N$ in the least-squares sense, the statement: `X = M^(-1)*N` may be used. This syntax invokes the Moore-Penrose generalized inverse, and will work for any matrices M and N.

25·5. INVERSE(M)[4] matrix inverse

INVERSE(M) (which may be abbreviated INV(M)) constructs a square matrix which is the inverse of the square matrix M using LU factorization via gaussian elimination. If M is algorithmically singular, an error message will be produced. For well-conditioned square matrices, INVERSE will produce nearly same result as M^-1. The syntax M^-1, however, computes the Moore-Penrose (SVD) pseudo-inverse, and is applicable to any matrix, square or non-square, singular or non-singular.

25·6. EIGEN(M)[4] matrix eigenvalues and eigenvectors

EIGEN(M) constructs a matrix which contains the eigenvalues and eigenvectors of a general real square matrix by the QR method. M must be a matrix. EIGEN produces a matrix result, E. If M

is an $r \times r$ matrix, then E will have $2r + 2$ rows. E ROW 1 is the real part of the set of eigenvalues of M. E ROW 2 is the imaginary part of the set of eigenvalues of M. Each successive pair of rows of E contain the real and imaginary parts, respectively, of the corresponding right eigenvectors of M.

25·7. SVD(M)[4] singular value decomposition

SVD(M) constructs a matrix which contains the singular value decomposition of a general real rectangular matrix. M must be a matrix. SVD produces a matrix result, H, which is composed of three matrices, S, U, and V, such that M = U S V′.

Recall that the singular values of M are the non-negative square-roots of the necessarily non-negative real eigenvalues of the positive semidefinite symmetric matrix M′ M. S is a diagonal matrix, containing the singular values of M in decreasing order, and U and V are matrices with $U'U = I$ and $V'V = I$. If M is $r \times c$, with $r \geq c$, then H is an $(r+c+1) \times c$ matrix with S = DIAG((H ROW 1)′). U = H ROW 2:$(r+1)$, and V = H ROW $(r+2)$:$(r+c+1)$. U is an $r \times c$ matrix, S is a $c \times c$ matrix, and V is a $c \times c$ matrix. If $r < c$, then the computation is performed on M′ , which interchanges U and V.

Example: (The numbers in this example have been rounded to three places.)

Let $M = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$. Then $\mathrm{SVD}(M) = \begin{bmatrix} 9.526 & .514 \\ -.230 & -.883 \\ -.525 & -.241 \\ -.820 & .402 \\ -.620 & .785 \\ -.785 & -.620 \end{bmatrix}$.

Then $S = \begin{bmatrix} 9.526 & 0 \\ 0 & .514 \end{bmatrix}$, $U = \begin{bmatrix} -.230 & -.883 \\ -.525 & -.241 \\ -.820 & .402 \end{bmatrix}$, and $V = \begin{bmatrix} -.620 & .785 \\ -.785 & -.620 \end{bmatrix}$.

25·8. RANK(M)[4] matrix rank

RANK(M) computes a scalar which is the rank, i.e. the dimension of the row space, of the input matrix. M must be a matrix. SVD(M) is computed, and the singular values $\geq 10^{-10}$ are counted.

25·9. LENGTH(M[,X])[4] vector length

LENGTH(M[,X]) constructs a matrix the rows of which are the lengths of the rows of M considered as vectors in an NCOLS(M)-dimensional space. M must be a matrix. The optional argument X must be a positive scalar which defaults to 2. X specifies the metric to be used. X = 2 corresponds to the Euclidean metric. $X \geq 10$ corresponds to the ∞-metric (which is the same as the maximum absolute element norm).

The result is an NROWS(M)-dimensional vector R, the i-th element of which is the length of M ROW i, so

$$R_i = \left[\sum_{j=1}^{n} M_{i,j}^{X} \right]^{1/X} .$$

25·10. MNORM(M[,X])[4] matrix norms

MNORM(M[,X]) computes a scalar which is the value of the specified matrix norm of the input matrix M. M must be a matrix; it need not be square. X must be a scalar.

X is a integer value which specifies the norm to be computed. If X is omitted, it defaults to the value 2. In the discussion below, A denotes an $m \times n$ matrix. In general, the matrix p-norm $\|.\|_p$ with respect to the vector p-norm $|.|_p$ is defined as follows:

$$\|A\|_p = \sup_{x \neq 0}\{|Ax|_p/|x_p|\}.$$

There are four matrix norms provided by MNORM.

If X = 0, the Frobenius matrix norm is computed. Recall the definition of the Frobenius matrix norm of A.

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|A_{ij}|^2}.$$

Let $\{\sigma_1, \sigma_2, \ldots, \sigma_p\}$ be the singular values of A, where $p = \min(m, n)$. The Frobenius matrix norm of A satisfies

$$\|A\|_F = (\sigma_1^2 + \sigma_2^2 + \ldots + \sigma_p^2)^{1/2}.$$

If X = 1, the row matrix norm (matrix 1-norm) is computed. Recall the definition of the row matrix norm of A.

$$\|A\|_1 = \max_j \left[\sum_{i=1}^{m}|A_{ij}|\right].$$

If X = 2, the matrix 2-norm; (SVD matrix norm) is computed. Recall the definition of the matrix 2-norm. Let σ_1 be the largest singular value of A. Then σ_1 is the matrix 2-norm of A. Also $\|A\|_2$ = SQRT(MRHO($A'*A$))

If X = 3, the column matrix norm (matrix infinity-norm) is computed. Recall the definition of the column matrix norm of A

$$\|A\|_\infty = \max_i \left[\sum_{j=1}^{n}|A_{ij}|\right].$$

25·11. MRHO(M)[4] spectral radius of a matrix

MRHO(M) computes a scalar which is the value of the spectral radius of the input matrix M. M must be a square matrix.

Recall the definition of the spectral radius of a matrix. Let $\{\lambda_1, \lambda_2, \ldots, \lambda_p\}$ be the eigenvalues of M, where p = NROWS(M) . The spectral radius of M is $\rho(M) = \max(|\lambda_1|, |\lambda_2|, \ldots, |\lambda_p|)$, where $|\lambda_j|$ denotes the modulus of the (possibly complex) eigenvalue λ_j.

25·12. COND(M)[4] condition number of a matrix

COND(M) computes a scalar which is the matrix 2-norm condition number of the input matrix M. M must be a matrix, but it need not be square.

Given a matrix M, the condition number of M, with respect to a matrix norm $\|\cdot\|$ is $\|M\| \cdot \|M^{-1}\|$, where the Moore-Penrose generalized inverse is denoted by M^{-1}. Note that with this definition, the condition number of a singular matrix is not "infinite", nor necessarily even particularly large. COND computes the condition number with respect to the matrix 2-norm, which is the ratio of the largest singular value of M to the the smallest non-zero singular value of M. Condition numbers with respect to other matrix norms may be obtained by using MNORM.

25·13. QRFAC(M)[4] QR-factorization of a matrix

QRFAC(M) returns a matrix Z which contains the matrices Q and R corresponding to the QR-factorization of the matrix M, such that Z = Q&′ R. M need not be square. Let r = NROWS(M), and c = NCOLS(M). The QR-factorization of M expresses M as the product QR, where Q is an $r \times r$ orthogonal matrix, and R is an $r \times c$ upper triangular matrix.

In the case when $r > c$, Q contains some important information about M. Let Q1 = Q COL $1 : c$, and Q2 = Q ROW $c+1 : r$. If M has full column-rank (rank(M) = c), then colspace(M) = colspace(Q1), and nullspace(M) = orthogonal complement of colspace(M) = colspace(Q2). If $r \leq c$, the relations are similar but weaker.

---

26. Matrix Data Processing Functions

26·1. NROWS(M)[0] number of rows of a matrix

NROWS(M) computes a scalar which is the number of rows of the matrix or string array M. M must be a matrix or a string array. NROWS produces a scalar result.

26·2. NCOLS(M)[0] number of columns of a matrix

NCOLS(M) computes a scalar which is the number of columns of the matrix or string array M. M must be a matrix or a string array. NCOLS produces a scalar result.

26·3. MSIZE(M)[0] size of a matrix

MSIZE(M) constructs a scalar S which is the size, i.e. the number of rows times the number of columns of the input matrix or string array M. MSIZE(M) = NROWS(M)·NCOLS(M).

26·4. MAXV(M)[0] maximum element of a matrix

MAXV(M) computes a scalar which is equal to the largest element of the matrix M. M must be a matrix. MAXV produces a scalar result. To obtain the largest element of a particular set of rows or columns of a matrix, use the appropriate submatrix expression, e.g. MAXV(M ROW 4:63).

26·5. MINV(M)[0] minimum element of a matrix

MINV(M) computes a scalar which is equal to the smallest element of the matrix M. M must be a matrix. MINV produces a scalar result. To obtain the smallest element of a particular set of rows or columns of a matrix, use the appropriate submatrix expression, e.g. MINV(M ROW 4:63).

26·6. MAXROW(M)^0 row index of maximum element in matrix

MAXROW(M) computes a scalar which is equal to the row index of the first occurrence of the largest element of the matrix M. M must be a matrix.

26·7. MAXCOL(M)^0 column index of maximum element in matrix

MAXCOL(M) computes a scalar which is equal to the column index of the first occurrence of the largest element of the matrix M. M must be a matrix.

26·8. MINROW(M)^0 row index of minimum element in matrix

MINROW(M) computes a scalar which is equal to the row index of the first occurrence of the smallest element of the matrix M. M must be a matrix.

26·9. MINCOL(M)^0 column index of minimum element in matrix

MINCOL(M) computes a scalar which is equal to the column index of the first occurrence of the smallest element of the matrix M. M must be a matrix.

26·10. COMPRESS(M[,X[,Y]])^0 censor a matrix

COMPRESS(M[,X[,Y]]) constructs a "censored" version of a matrix with selected rows missing. M must be a matrix. The optional scalar X must be an integer, representing a valid column number of M. If X is omitted, it defaults to 1. Y is an optional scalar which defaults to 0.

COMPRESS produces a matrix V with NCOLS(M) columns. Each successive row of M that does not have the value Y in column X is copied into V, so that V has as many rows as there are non-Y elements in M COL X. It is possible to return an empty matrix with zero rows with COMPRESS. To compress selected columns of a matrix M, use COMPRESS on M$'$, and then transpose the result, i.e. NC = COMPRESS(M$'$,3)$'$.

Example.

Let $M = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 3 & 1 \\ 4 & 0 \\ 5 & 1 \end{bmatrix}$. Then COMPRESS$(M, 2) = \begin{bmatrix} 2 & 1 \\ 3 & 1 \\ 5 & 1 \end{bmatrix}$.

26·11. EXTRACT(M[,X[,Y]])^0 extract rows from a matrix

EXTRACT(M[,X[,Y]]) constructs a copy of a matrix with all but selected rows missing. M must be a matrix. The optional scalar X must be an integer, representing a valid column number of M. If X is omitted, it defaults to 1. Y is an optional scalar which defaults to 0.

EXTRACT produces a matrix V with NCOLS(M) columns. Each successive row of M that has the value Y in column X is copied into V, so that V has as many rows as there are Y-values in M COL X. It is possible to return an empty matrix with zero rows with EXTRACT. To extract selected columns of a matrix M, use EXTRACT on M$'$, and then transpose the result, i.e. NC = EXTRACT(M$'$,3)$'$.

Example.

Let $M = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 3 & 1 \\ 4 & 0 \\ 5 & 1 \end{bmatrix}$. Then EXTRACT$(M, 2) = \begin{bmatrix} 1 & 0 \\ 4 & 0 \end{bmatrix}$.

26·12. SORT(M[,X[,Y]])^0 sort a matrix on a column

SORT(M[,X[,Y]]) constructs a matrix which contains the rows of a matrix sorted either on a specified column or lexicographically. M must be an $r \times c$ matrix. X and Y must be scalars.

The optional argument X must be an integer, which is either ≤ 0 or a valid column number of M. If X is omitted, it defaults to 1. When $X > 0$, the rows of M are to be sorted on column X. $X \leq 0$ means: treat the rows of M as c-dimensional vectors, which are to be sorted in lexicographic order, either ascending or descending, depending upon the sign of Y.

The optional argument Y must be a scalar. If Y is omitted, it defaults to 0. If $Y \geq 0$, an ascending sort is performed. If $Y < 0$, a descending sort will be done.

For $X \leq 0$, SORT returns an $r \times c$ matrix containing the rows of M sorted in lexicographic order, in ascending or descending order, according to Y.

For $X > 0$, SORT returns an $r \times c$ matrix containing the rows of M sorted in ascending or descending order, according to Y, so that column X is in monotone order.

The sorting done by SORT is stable, i.e. duplicate keys retain the same relative order. Thus, SORT can be used repetitively to sort on several columns.

26·13. ROWSUM(M)^0 sum of the rows of a matrix

ROWSUM(M) constructs an $1 \times$ NCOLS(M) matrix S which contains the sum of the rows of the input matrix M. $S[1, j]$ = SUM(i,1,NROWS(M),M[i,j]).

26·14. COLSUM(M)^0 sum of the columns of a matrix

COLSUM(M) constructs a NROWS$(M) \times 1$ matrix s which contains the sum of the columns of the input matrix M. S[i] = SUM(j,1,NCOLS(M),M[i, j]).

26·15. PSUM(M)^0 partial row sums of a matrix

PSUM(M) constructs a matrix R which is the same shape as the input matrix M, such that $R[i, j]$ = SUM$(k, 1, i, M[k, j])$

Example: Let $M = [1, 2, 3, 4, 5]'$. Then PSUM$(M) = [1, 3, 6, 10, 15]'$

26·16. RDUP(M[,F[,C[,T[,S]]]])^6 thresholded duplicate removal

RDUP(M[,F[,C[,T[,S]]]]) returns a matrix which consists of a copy of M with specified rows modified and certain other rows removed. M is a matrix. F is an integer indicating the type of result matrix to be generated. If F is not given, -1 is used by default. T is a scalar representing a threshold value. If T is omitted, 0 is used by default. C is a scalar representing a comparison mode. If C is omitted, 0 is used by default. S is an integer indicating whether a count column is to be included in the result. If S is not given, 0 is used by default.

Let b = NCOLS(M) and let $c = \lfloor C \rfloor$. When $0 < c \leq b$, c specifies a column index, and the distance between M ROW i and M ROW j is defined to be: $|M[i, c] - M[j, c]|$.

When c = 0, the distance between M ROW i and M ROW j is defined to be:

$$\sum_{k=1}^{b} |M[i,k] - M[j,k]|$$

When $c < 0$, the distance between M ROW i and M ROW j is defined to be:

$$\left(\sum_{k=1}^{b} |M[i,k] - M[j,k]|^2 \right)^{1/2}$$

The rows of a copy of M are sequentially scanned and, for each (remaining) row i, every row j with $j > i$ such that distance(M ROW i, M ROW j) $\leq |T|$ is called a "duplicate" row of row i. These duplicate rows are processed and removed according to the result-form code F for each remaining row i.

The result-form code argument F is interpreted as follows:

If $F < 0$, we place the average of all the duplicate rows of row i into row i for each row i.
If $F = 0$, we keep row i, and delete all the following duplicate rows of row i.
If $F > 0$, we place the average of column F of all the duplicate rows of row i into row i for each row i. The other positions in row i being kept in the output are unchanged.

The count-switch argument S is interpreted as follows.

If S =0, RDUP does not produce an additional count column in the output.

If S =1, RDUP appends an additional column to the output matrix whose i-th entry is the number of duplicate rows that appeared in the input corresponding to the output row i.

The resulting matrix, which has been "thinned" by having all following rows sufficiently close to row i removed for $i = 1, 2, \ldots,$ NROWS(M), is returned. The remaining rows may be averages, or exact copies of input rows, depending upon the result code F. Note that if T = 0, RDUP removes only exact "duplicate" rows with respect to the distance metric specified by C.

The process performed by RDUP takes an amount of time proportional to b^2; this can be excessive for large matrices, so an option is provided to declare that the matrix M is sorted so that the time needed can be reduced when exact duplicate rows are to be processed. When $T < 0$, the matrix M is declared to be sorted, and T is taken to be 0.

26·17. TIES(M)[9] multiplicity of a matrix element

TIES constructs a matrix with the same shape as the input matrix M. The (i, j)-th element of the output matrix is the number of times the i-th distinct element of M column j occurs, sorted in increasing order. TIES is useful in correcting for ties in rank-based nonparametric tests.

Example: Let $M = \begin{bmatrix} 1 & 2 \\ 1 & 3 \end{bmatrix}$. Then TIES(M) is the 2x2 matrix, $\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$.

26·18. ELEMENT(M,V)[9] is a value in a matrix

ELEMENT takes a matrix M and a scalar V as arguments. ELEMENT returns 1 if the value V is in the matrix M, otherwise it returns 0.

Example: If $M = \begin{bmatrix} 1 & 2 \\ 2.2 & 3.7 \end{bmatrix}$, then ELEMENT(M,1) = 1 and ELEMENT(M,5) = 0.

---

27. Matrix Construction Functions

27·1. LIST(A,...)^0 construct a matrix from a list

LIST(A,...) constructs a matrix from the arbitrary-length argument list of one or more matrices and/or scalars. Each argument A must be a scalar or a matrix. LIST(A,...) produces a 1-column matrix M. M is constructed as follows. First, a 1-column matrix is implicitly constructed for each supplied argument; for a scalar argument, the corresponding matrix has one row; for a matrix argument A, the corresponding matrix has NROWS(A)·NCOLS(A) rows, and is formed by listing the rows of A in sequence in a single column. These 1-column matrices are then row-concatenated in the order provided to produce the final 1-column result.

Example: Let A = 10, M = (1:3)′ , B = 20, and N = (1:2)′ . Then LIST(A,M,B,N) = [10,1,2,3,20,1,2]′ .

27·2. CROSS(M,N)^0 tensor product of matrices

CROSS(M,N) constructs a matrix which contains the Cartesian set-theoretic cross (tensor) product of two matrices. M and N must be matrices. CROSS(M,N) produces a matrix L, with NROWS(M)·NROWS(N) rows and NCOLS(M)+NCOLS(N) columns. L is constructed as follows. Each row of M is column-concatenated with every row of N. The resulting vectors form the rows of M. The usual case is that M and N are 1-column vectors, in sort order, and the result is the tensor product basis of a rectangular domain.

Example: Let $M = 1:3$ and $N = 1:2$. Then CROSS(M,N) is the 6×2 matrix: $\begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 2 & 1 \\ 2 & 2 \\ 3 & 1 \\ 3 & 2 \end{bmatrix}$.

27·3. SHAPE(X,Y[,A])^4 restructure a matrix

SHAPE(X,Y[,A]) constructs a matrix which is a restructuring of a given matrix. X and Y must be non-negative integer scalars. X is the desired number of rows and Y is the desired number of columns in the result matrix. The optional argument A is a matrix or scalar; it defaults to zero. SHAPE produces a matrix result.

SHAPE(X,Y) produces an $X \times Y$ matrix of zeros. SHAPE(X,Y,A), where A is a scalar, produces an $X \times Y$ matrix, each element of which is equal to A. If A is a matrix, SHAPE(X,Y,A) produces an $X \times Y$ matrix whose successive rows are constructed from the successive elements of LIST(A). If the number of elements of A is less than X·Y, its elements are reused cyclicly. If A has more than X·Y elements, only the initial X·Y elements are used.

Example: Let $M = \text{SHAPE}(2, 2, 1:4)$. Then $M = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$.

27·4. DIAG(M[,A[,X]])^4 build a matrix by diagonals

DIAG(M[,A[,X]]) builds a square matrix P with the columns of the matrix M as the diagonals of P. M must be a matrix. If M is a 1-row matrix, it is treated as a 1-column matrix, by using M′ for M. A is an optional argument which is either a scalar or a 1-column matrix. A defaults to zero. X is an optional scalar, which defaults to zero.

DIAG produces a square matrix P. If A is omitted, $P[i,j] = 0$ if $i \neq j$, and $P[i,i] = M[i,1]$ and NROWS(P) = NCOLS(P) = NROWS(M).

We index the main diagonal of the output matrix P with the integer 0. The successive superdiagonals of P have the increasing positive indices: $1, 2, \ldots$ The successive subdiagonals of P have the decreasing negative indices: $-1, -2, \ldots$

If A is supplied, it is taken as a list of one or more diagonal indices, the elements of which correspond to the successive columns of M. The output matrix P is constructed by copying the i-th column of the matrix M into diagonal number $A[i]$ of the matrix P. Let r = NROWS(M) - MINV(ABS ON A). P is an $r \times r$ matrix. Unspecified elements of P are filled with the value of the scalar X.

Example: Let $M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$. Then $\text{DIAG}(M, -1:1) = \begin{bmatrix} 2 & 3 & 0 \\ 1 & 5 & 6 \\ 0 & 4 & 8 \end{bmatrix}$.

27·5. GETDIAG(M[[,A],X])[4] extract diagonals of a matrix

GETDIAG(M[,A[,X]]) obtains the diagonal(s) of the $n \times m$ matrix M. The optional argument A is 0 by default and is coerced to a 1-column matrix. The optional scalar argument X is 0 by default.

The $n \times m$ matrix M has its diagonals numbered from $-(n-1)$ to $(m-1)$. Diagonal number k is the k-th diagonal above the main diagonal when $k \geq 0$, and diagonal number k is the $(-k)$-th diagonal below the main diagonal when $k \leq 0$.

A is taken as a list of diagonal numbers of length h. The value of GETDIAG(M,A,X) is the $\min(m,n)$ rows by h columns matrix R where R COL j is the $A[j]$-th diagonal of M extended to $\min(m,n)$ elements in length with the value X.

Example: Let $M = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$. Then $\text{GETDIAG}(M, -1:1) = \begin{bmatrix} 2 & 1 & 4 \\ 6 & 5 & 8 \\ 0 & 9 & 0 \end{bmatrix}$.

27·6. BAND(M,[A[,X]])[4] build a diagonally-banded matrix

BAND(M,[A[,X]]) builds a diagonally-banded matrix with the selected diagonals of the $n \times m$ matrix M. The optional argument A is 0 by default and is coerced to a 1-column matrix. The optional scalar argument X is 0 by default.

The $n \times m$ matrix M has its diagonals numbered from $-(n-1)$ to $(m-1)$. Diagonal number k is the k-th diagonal above the main diagonal when $k \geq 0$, and diagonal number k is the $(-k)$-th diagonal below the main diagonal when $k \leq 0$.

A is taken as a list of diagonal numbers of length h. The value of BAND(M,A,X) is the $n \times m$ matrix R where R is identical to M except that the elements in any diagonal of R which is not specified in A are set equal to the value X.

27·7. ROTATE(M,X[,Y])[0] rotate a matrix by rows or columns

ROTATE(M,X[,Y]) returns a matrix in which the rows or columns (depending upon the optional scalar Y) of the input matrix M are rotated by X positions. M must be a matrix. X and Y must be scalars. If Y is omitted, it defaults to zero. If $Y \geq 0$, row-wise rotation is performed, otherwise column-wise rotation is performed. If $X \geq 0$, increasing-direction rotation is performed. If $X < 0$, decreasing-direction rotation is performed. In either case, for $Y \geq 0$, M ROW $j \rightarrow$ M ROW ((j+X) mod NROWS(M)), where *mod* is constrained to return a residue in $\{1, \ldots,$ NROWS(M)$\}$. A similar statement applies for $Y < 0$. The output matrix has the same dimension as the input M.

Example: Let M = [1,2,3,4,5]. Then rotate(M,2,-1) = [3,4,5,1,2].

27·8. MESH(M,N[,X])[0] interleave two matrices by rows or columns

MESH(M,N[,X]) returns a matrix in which the rows or columns (depending upon the optional scalar X) of the input matrices M and N are combined alternately. M and N must be matrices. If the optional scalar X is omitted, it defaults to zero.

If $X \geq 0$, row-wise meshing is performed. If $X < 0$, column-wise meshing is performed. Let $a =$ NROWS(M), $c =$ NROWS(N), $b =$ NCOLS(M), and $d =$ NCOLS(N).

If the two matrices are not of equal length in the mesh direction, implicit cyclic extension will be used on the shorter. The output matrix has the dimensions $2\max(a,c) \times \max(b,d)$ when row-wise meshing is done. The output matrix has the dimensions $\max(a,c) \times 2\max(b,d)$ when column-wise meshing is done.

Example: Let M = [1,2,3] and N = [6,7]. Then MESH(M,N,-1) = [1,6,2,7,3,6]

28. Graphics Computations

28·1. CONTOUR(D[,L])[12] level lines of a surface

CONTOUR(D,L) returns a matrix which contains the piecewise-linear contour curves for a surface corresponding to a function of two variables, $f(x,y)$. D must be an nm-row 3-column matrix. D COL 1:2 must be a tensor product basis for a rectangular region, e.g. produced by the application of the CROSS operator to a pair of increasing sequences of n x-coordinates and m y-coordinates. That is, the first m numbers in D COL 1 are the same and equal to the first x-coordinate, while the corresponding numbers in D COL 2 go through the range of y-coordinates. The second m numbers in D COL 1 must again be the same but equal to the second x-coordinate, while the corresponding numbers in D COL 2 repeat the first m numbers in D COL 2, and so on. D COL 3 contains the values of $f(x,y)$ on D COL 1:2.

L, if it is provided, must be a scalar or a 1-column matrix. If L is a 1-column matrix, it is taken as a list of one or more level values for which contour curves are to be constructed. In this case, L is ordinarily, but not necessarily in increasing sort order. If L is a scalar, then it is taken as *the number* of level values for which contour curves are to be constructed. In this case contour curves are constructed at L equally spaced level values which range from the minimum value of $f(x,y)$ in D COL 3 to the maximum value of $f(x,y)$ in D COL 3. If L is not provided, a scalar value of 12 is provided for L as a default.

The result of CONTOUR is a 2-column matrix, C, which can be drawn using one of the linetypes MARKER (6), VMARKER (7), SMARKER (8), or SVMARKER (9) to display the contours of the function F for the level values in L.

Example: Let $D =$ CROSS$(1:3, -1:1)$. Then $D = \begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 \\ -1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 \end{bmatrix}'$.

Suppose we wish to draw the contours of the function defined by: FCT F(X,Y) = X+Y. Then we may define the surface using the command: Q = POINTS(F,D), and construct the contours using the command: R = CONTOUR(Q,-2:4). The contour plot may now be drawn using the command: DRAW R LINE VMARKER, and viewed using the VIEW command.

28·2. FILL(M[,X[,Y]])[zz] fill lines of a polygon

FILL(M[,X[,Y]]) returns a matrix which contains fill-lines for a polygon. M must be a matrix. The optional arguments X and Y must be scalars. M is a 2-column of matrix of coordinate pairs defining

the polygon. Alternatively, M may contain several polygons each preceded by a marker row in M. A marker row $(M[i,1], M[i,2])$ has $M[i,1] \geq 10^{300}$. The polygon is implicitly closed by a line connecting the initial and final points. X is the desired number of fill lines. If omitted, X defaults to 350; usually this is enough to fill the polygon densely. Y is the angle of the fill lines (in degrees). If omitted, Y defaults to 0. To achieve guaranteed dense filling of polygons, Y should be 0. The result of FILL(M[,X[,Y]]) is a 2-column matrix, which can be drawn to show the fill lines within the polygon (but not the polygon boundary itself) using linetype ALTERNATE (5).

28·3. CHULL(P)[7] convex hull

CHULL(P) returns the vertices of the convex hull of the 2-dimensional points given in the rows of the matrix P. P must be a 2-column matrix. The result is a 2-column matrix whose rows are the points of P which are vertices of the convex hull polygon of all the points in P.

28·4. CURVEM(C)[0] extract data of a curve or axis

CURVEM(C) returns a matrix which contains the data points of a curve. C must be a curve or axis. The result is a 2-column matrix which is the set of points defining the curve, in the original units in which they were defined by the user.

28·5. LABELM(C)[0] extract label numbers of a curve or axis

LABELM(C) returns a matrix which contains the label data matrix of a curve or axis. C must be a curve or axis. The result is a 1 or 2-column matrix which is the most recently specified set of values for the labels of the curve or axis, in the original units (and values) in which they were defined by the user.

28·6. PTSIZEM(C)[0] extract pointsize matrix of a curve or axis

PTSIZEM(C) returns a matrix which contains the pointsize matrix of a curve or axis. C must be a curve or axis. The result is a copy of the pointsize marix associated with C. If the pointsize of C is a scalar, then the corresponding 1×1 matrix is returned.

28·7. WINDOWM(W)[0] extract limit numbers of a window

WINDOWM(W) returns a matrix which contains the limit numbers of a window. W must be a window. The result is a 4 element matrix M. M[1] is the lower limit of the x-axis; M[2] is the upper limit of the x-axis; M[3] is the lower limit of the y-axis; and M[4] is the upper limit of the y-axis.

28·8. FRAMEM(W[,X])[0] extract limit numbers of the frame of a window

FRAMEM(W,X) returns a matrix which contains the limit numbers of the frame associated with the window W in units specified by X. W must be a window. The optional argument X is the unit specifier. The options for X are 1 (FRACT) or 2 (INCHES). If X is omitted, it defaults to 1 (FRACT). The result is a 4 element matrix R. R[1] is the lower horizontal limit of the frame; R[2] is the upper horizontal limit of the frame; R[3] is the lower vertical limit of the frame; and R[4] is the upper vertical limit of the frame. The numbers in R are in the units specified by X.

28·9. IMAGEM(W[,X])[0] extract limit numbers of the image of a window

IMAGEM(W,X) returns a matrix which contains the limit numbers of the image associated with the window W in units specified by X. W must be a window. The optional argument X is the unit

specifier. The options for X are 1 (FRACT), 2 (INCHES), 3 (FFRACT), or 4 (FINCHES). If X is omitted, it defaults to 3 (FFRACT). The result is a 4 element matrix R. R[1] is the lower horizontal limit of the image; R[2] is the upper horizontal limit of the image; R[3] is the lower vertical limit of the image; and R[4] is the upper vertical limit of the image. The numbers in R are in the units specified by X.

28·10. STEPGRAPH(M)^0 construct step function graph matrix

STEPGRAPH(M) computes a matrix suitable for drawing as a step-function graph. M must be a $r \times 2$ matrix, which is interpreted as the points of a tabular function. M COL 1 must be in (increasing) sort order. The result is a $2r \times 2$ matrix H, which contains the coordinates of the line-segments defining the step function specified by M. A step-function graph is one which is piecewise constant. It consists of a horizontal line from the first point, (M_{11}, M_{12}), to the point (M_{21}, M_{12}), a vertical line from there to the second point, (M_{21}, M_{22}), etc. STEPGRAPH is useful for converting data representing piecewise-constant functions, e.g. survival curves, to a form suitable for drawing.

28·11. BARGRAPH(M [,X])^0 construct bar graph matrix

BARGRAPH(M [,X]) computes a matrix suitable for drawing as a bar graph. M must be an $r \times 2$ matrix, column-1 of which is interpreted as the x-coordinates of the bar midpoints and column-2 of which is interpreted as the levels of the bar graph. M COL 1 must be in increasing order. The optional argument X must be a scalar, which defaults to zero.

If $X \neq 0$, points which correspond to the lower end-points of vertical lines which separate the bargraph segments will be generated. If X = 0, no such points will appear and a resulting graph will look like the silhouette of the bar-graph with vertical lines. When X = 0, the result is a $(2r+2) \times 2$ matrix of line segment coordinates defining the bar graph. When $X \neq 0$, the result is a $(3r+1) \times 2$ matrix of line segment coordinates defining the bar graph.

28·12. COLORN(X)^0 assign a color number each number

COLORN(X) computes a valid color number for any scalar X. An element is selected from the list of colors using 1+INT(MOD(ABS(X-1),15) to index into the list. The COLORN function is intended to simplify the task of drawing several successive curves, generated, for example, in a for-loop using the for-loop variable to select the color.

argument	color name
1	WHITE
2	RED
3	GREEN
4	YELLOW
5	BLUE
6	VIOLET
7	PURPLE
8	ORANGE
9	PINK
10	AQUA
11	CHARTREUSE
12	BROWN
13	ROSE
14	TURQUOISE
15	BLACK
16	GREY

Example: COLORN(43) = ROSE

28·13. COLORX(R,G,B)[0] color number for an (R,G,B) triple

COLORX(R,G,B)) computes an integer scalar that specifies the color number closest to the color defined by the (red,green,blue) triple (R,G,B), for the display in use. R, G, and B are intensity numbers that lie between 0 and 1. The triple (0,0,0) refers to the color black; (1,0,0) refers to red and (1,1,1) refers to white.

29. Functional Computations

29·1. INTEGRATE(F1'T,F2'T,...,N)[1] solve ordinary differential equations

INTEGRATE(F1′ T,F2′ T,...,N) computes a matrix which contains the numerical solution of a system of first order ordinary differential equations: F1′ T, F2′ T,... There may be a variable number of differential equations in the system, up to 100. Recall that differential equations are defined in MLAB using function statements of either of the equivalent forms:

```
FCT F1 DIFF T(T) = ...  or FCT F1′ T(T) = ...
```

Each differential equation in the system must have an associated initial condition. Recall that initial conditions are defined in MLAB using INITIAL statements of the form: `INITIAL F1(A) = ...`, where the initial value of the independent variable (A in the above INITIAL statement) must be the same for all the differential equations in the system to be solved simultaneously.

N must be a 1-row or 1-column matrix, the elements of which are the values of the independent variable in the system of differential equations for which the solutions are to be recorded. It is not necessary for the numbers in N to be closely spaced in order to insure an accurate solution, unless differential equations with delay terms are being solved.

The INTEGRATE operator solves an initial value problem for a specified system of ordinary differential equations. INTEGRATE takes a variable number of arguments, $k+1$. The first k arguments are the names of the derivative functions which define the system of differential equations which are to be solved simultaneously. N is a 1-column matrix which is the set of values of the integration variable at which the solution is to be tabulated. The result is a matrix, M, which has $2k+1$ columns and NROWS(N) rows. M COL 1 is a copy of N. The k even columns $(2, 4, \ldots, 2k)$ of M hold the tabulated numerical values of the solution functions to the k differential equations, in the order listed in the integrate statement, for the values of the integration variable in M column 1. The k odd columns $(3, 5, \ldots, 2k+1)$ of M are the tabulated numerical values of the derivative functions corresponding to the solution functions of the k differential equations which are tabulated in the even columns for the values of the integration variable in the rows of N.

Whenever a system of one or more differential equations is solved with INTEGRATE, a string called ODESTR is created which contains the names of the functions in the order corresponding to the columns of the output matrix M.

The operation of the differential equation solver is controlled by a number of scalars which may be set by the user if desired using ordinary assignment statements. If they are not set, each defaults to a value which will generally give satisfactory solutions. These control variables are listed below.

METHOD: METHOD is a scalar which specifies the algorithm used to solve the differential equation system. There are four alternatives for METHOD.

- GEAR: Gear's third order method is used, with a variable step-size controlled by ERRFAC.

- GEAR2: Shrager's order "2.5" form of Gear's method is used, with a variable step-size controlled by ERRFAC.
- ADAMS: The Adams method of order three (effectively of order four) is used, with a variable step-size controlled by ERRFAC.
- MIXED: The Gear2 and Adams methods above are both used, starting with Adams method, and switching to Gear's method whenever the internal step-size becomes too small. Switching between the two methods continues to occur (less and less frequently) in order to maximize the mean step-size archived.

The default value of method is 0 (mixed). The predefined MLAB constants MIXED (equal to 0), ADAMS (equal to 1), and GEAR2 (equal to 2), and GEAR (equal to 3) are available in MLAB. METHOD is given its desired value using an ordinary assignment statement, e.g. METHOD = GEAR2. If METHOD is typed out, its numerical value (0, 1, 2, or 3) will be given, however, rather than the above names.

ERRFAC: ERRFAC is the error tolerance used for stepsize control. When the solution is large, a proportional error criterion is used. When the solution is near zero, a graded absolute/proportional error criterion is used.

The default value for ERRFAC is 10^{-3}. Larger values for ERRFAC may lead to solutions which are increasingly inaccurate. Smaller values for ERRFAC may lead to solutions which are somewhat more accurate. However, the time required to obtain the solutions is increased by using small values of ERRFAC. Values of ERRFAC smaller than 10^{-6} to 10^{-7} may not be able to lead to any further increase in accuracy. However, the value of ERRFAC indirectly determines the step-size used in solving the differential equations, and thus small values of ERRFAC may be used to reduce the step-size.

ODERPT: ODERPT is a scalar which controls intermediate screen output from the differential equation solver. It may take on the following values:

- 0: means no intermediate reporting
- 1: means report the independent variable value, step size, and method at each step
- 2: means report the independent variable value, step size, method and also the derivative values at each step
- 3: means report only the summary information after solving is complete. This includes the number of steps taken and the number of derivative function evaluations and Jacobian evaluations.

This reporting is useful to identify regions of stiffness in a differential equation system. The default value for ODERPT is zero.

MANDSW: MANDSW is a scalar control value used to determine whether interpolation is to be used in determining the solution at each reporting point. If MANDSW is TRUE ($\neq 0$), the step-size is adjusted so that a step ends exactly at each reporting point. If MANDSW = 0 (FALSE), interpolation is permitted, and larger steps will generally be used. Thus, MANDSW = 0 generally leads to faster, but possibly less accurate, solutions. The default value of MANDSW is 0 (FALSE).

JACSW: JACSW is a scalar control value used to determine how the Jacobian matrix is to be computed for Gear's method. It may take on the following values:

- 0: means that numerical partial derivatives are to be used
- 1: means that symbolic partial derivatives are to be used

- 2: means that the partial derivatives are to be computed once only, numerically

The predefined constants: NUMERICAL (0), SYMBOLIC (1), and CONSTJAC (2), are available in MLAB so that JACSW values can conveniently be assigned.

Nearly always, it is both more accurate and faster to use symbolic partial derivatives in computing the Jacobian matrix used with Gear's method. However, for exceedingly large and complicated differential equation systems, the symbolic partial derivatives may exhaust the available space for holding functions. In this event, it may still be possible to proceed using a numerical Jacobian.

DISASTERSW: DISASTERSW is a scalar control value used for handling error tolerance violations. The alternatives for DISASTERSW are

- -2: do not report errors, and increase tolerance indefinitely, as required.
- -1: do not report errors, and increase tolerance as required. If the internal value of ERRFAC becomes greater than 1.2, then terminate the integration.
- 0: report errors and increase tolerance as required. If the internal value of ERRFAC becomes greater than 1.2, then terminate the integration.
- 1: report errors and increase tolerance five times. If it would be necessary to increase the tolerance again, then terminate the integration.
- 2: report errors and increase tolerance five times. If it would be necessary to increase the tolerance again, then continue the solution with fixed step-size.
- 3: report errors and increase tolerance as required. If the internal value of ERRFAC becomes greater than 1.2, then continue the solution with fixed step-size.
- 4: force the solution from the beginning with a large fixed step-size of 1/80 of the total integration interval.
- 5: force the solution for one step. Do not increase the tolerance. Do not report any error.
- 6: force the solution for one step. Do not increase the tolerance. Type out an error report.

The default value of DISASTERSW is 6. Errors of the sort regulated by DISASTERSW are associated with the presence of singularities in the derivative functions or with steep changes in the solution of the differential equation system, i.e. Lipschitz condition violations. The numerical algorithm becomes less accurate near these points, and the requested per-step tolerance may be unobtainable. When derivative functions with singularities are known to be present, as for example as when a step-function appears, (e.g. if $t < t0$ then 0 else 1), then DISASTERSW = -1 is often the best option.

During the calculation of the numerical solution in the INTEGRATE command, the user may toggle on reporting the progress of the current solution by typing "R". A second "R" will toggle reporting off again. This process may be repeated.

During the calculation, the user may also request early termination, i.e. quitting the integration, by typing "Q". The solution up to that point may be available for examination. This feature may be useful when the equations are not being efficiently solved, due to excessive stiffness, for example.

Examples:

1) Two first-order differential equations with consistent initial conditions:

```
FCT X′ T(T) = ATAN2(X,Y); INIT X(0) = 4
FCT Y′ T(T)=SQRT(X*X+Y*Y); INIT Y(0) = 1
```

The solution to this system of differential equations on the set 0:10:.1 is obtained by the statement M = INTEGRATE(X′ T,Y′T,0:10:.1). The matrix M will have 101 rows and 5 columns, in the following order: M = [T X X′ T Y Y′ T].

2) A tabular integral: Suppose we have defined S′ X(X) and S(A) by: FCT S′ X (X) = G(X); INIT S(A) = 0. Now perform the integration by: M = INTEGRATE(S′ X, A:B!100). Then M COL 1:2 contains the desired tabular integral.

29·2. QROMB(F,A,B[,E[,M[,J]]])^0 definite integral

QROMB(F,A,B[,E[,M[,J]]]) computes a scalar equal to the definite integral of a function by adaptive Romberg quadrature. F must be the name of a user-defined scalar function of a single argument; it is the function which is to be integrated. The scalars A and B are the lower and upper limits of integration, respectively. The scalar E is optional; it is the relative accuracy desired. M is an optional scalar, the maximum number of iterations. J is an optional scalar; it is the number of points used in extrapolation. QROMB produces a scalar which is the value of the definite integral. QROMB may not be used in user functions; integral must be used for that purpose.

QROMB is generally more accurate than INTEGRAL for sufficiently smooth functions. The accuracy and time requirements of QROMB are controlled by the optional parameters E, M, and J. E is the relative accuracy. Choosing E smaller means a slower but more accurate computation. If E is omitted, it defaults to 10^{-6}. M is the maximum number of steps. Choosing M larger means a slower but more accurate computation. If M is omitted, it defaults to 20. J is the number of points used in extrapolation. A larger J means a slower but more accurate computation. If J is omitted, it defaults to 5.

29·3. INTEGRAL(X,A,B,E)^0 definite integral

INTEGRAL(X,A,B,E) computes a scalar which is the definite integral of the expression E. X is the name of the formal variable of integration. A and B must be scalars; they are the lower and upper limits of integration respectively. E is the expression which is to be integrated. Usually the symbol X will appear within E.

The definite integral is computed by solving the corresponding differential equation. INTEGRAL can only be used in function bodies; whereas QROMB must be used only at top level.

Tabular integrals, i.e. the points of the function FCT F(Z) = INTEGRAL(X,A,Z,G(X)) at successive values of Z, are easier to construct using an incremental formulaton, or by explicitly solving the differential equation corresponding to the integral function to be tabulated.

Since the expression E in this operator may itself be a function which contains the integral operator, certain classes of multiple integrals may be evaluated in this manner.

Example: Define G by FCT G(X) = .5+INTEGRAL(Z,0,X,GAUSSD(Z,0,1)).
Then G(x) = GAUSSF(x,0,1) for $x > 0$.

29·4. MINIMIZE(F[,A1,A2,...,Ap][,B1,B2,...,Bq][,C])^3 minimize a function
MAXIMIZE(F[,A1,A2,...,Ap][,B1,B2,...,Bq][,C])^3 maximize a function

MINIMIZE computes a local minimum, and MAXIMIZE computes a local maximum, of a function F of zero or more scalar or matrix-valued variables. Both a local minimum (or maximum) value of F and the associated argument and parameter values are obtained.

The input to both MINIMIZE and MAXIMIZE is as follows. The first argument is the name of the function to be minimized or maxmized. After the name F of the function to be optimized, the

next p arguments A1,A2,...,Ap *must* correspond to the $p \geq 0$ arguments of the function F; these are called argument-type parameters. The arguments A1, A2,...,Ap must be scalar or matrix variables. The next $q \geq 0$ arguments B1, B2, ..., Bq are the names of any global matrix or scalar parameters occurring within F that are to be adjusted.

The optional final argument C is the name of a set of constraints consisting of linear equality or inequality constraints involving scalars only. Currently array variables can only be constrained by a special device explained later.

The iterative optimization algorithm uses the input values of the variables as the initial guess. Upon completion, the input variable values are replaced by the best values achieved during the optimization. Array variables are treated as a set of scalar variables.

Example: The function F defined below is maximized and the associated value of the parameter A and values for the arguments X,Y, and Z are computed.

```
FCT F(X,Y,Z) = (X-A)^2 + SUM(I, 1,10,(X-Y[I])^2) + (Z + X - B)^2
X0 = 1; Y0 = 2^^10; Z = 3; A = 2; B = 10;
MAXIMIZE(F, X0, Y0, Z, A)
```

In this example, the arguments X and Z are scalars and the argument Y is an array of 10 arguments. X0, Y0 and Z are variables holding the initial values to be used for X, Y and Z. As for the parameters, we only want to adjust the parameter A and let the parameter B be fixed. Thus, we only put the symbol A in the parameter list. When the optimization is complete, X0, Y0, Z and the parameter A will be changed to hold the values that correspond to the maximum of F found by the MAXIMIZE operator.

Maximization of F is done by *minimizing* -F, so the remaining discussion only considers the minimization case. There are a number of control variables listed below that may be set via assignment statements to control the actions of the optimization process:

BACKTRACKSW: BACKTRACKSW is the backtrack process switch. The minimizer procedure may reach a state where the function value has not decreased much for several iterations. At this point, we may backtrack to the last point which was reached with an adequate decrease and go in the anti-gradient direction. If it is decided that a backtrack process is not desired, then set BACKTRACKSW to -1. Otherwise BACKTRACKSW should be ≥ 0. If BACKTRACKSW = 1, the backtrack process will be invoked according to the default conditions. If BACKTRACKSW is $>$ 1, more stringent conditions than the default conditions are required for backtracking to be invoked. If 0 $\leq$BACKTRACKSW $<$1, less stringent requirements than the default conditions will result in a backtrack process. When BACKTRACKSW = 0, backtracking will always be invoked. As a rough estimate we note that when the backtrack process is invoked but fails to improve the best estimate of the minimum, on the average, the extra cost is about a dozen function and gradient evaluations.

MAXIT: MAXIT is the maximum number of search directions (iterations) that the program is permitted to compute. If MAXIT is ≤ 0, the program will just find a feasible point and compute the function value at that point and return. The default value of MAXIT is 1000.

EPS: EPS is the upper bound required for each element of the gradient vector, q, at the minimum in an unconstrained problem. That is, the convergence criteria requires that abs$(q_i) \leq$ EPS for $i = 1, \ldots, n$. The default value of EPS is .000001.

RHO: RHO = $10^{-\sigma}$, where σ is the number of correct significant digits required in every output parameter value being adjusted. The default value of RHO is .001.

MAXSLOW: MAXSLOW is the number of consecutive searches (iterations) the program will make before declaring "slow convergence" when the estimate of the function minimum is not decreasing by more than TOLERANCE·F(X) (see TOLERANCE, below). The default value of MAXSLOW is 16.

TOLERANCE: TOLERANCE is used to define slow convergence. If the difference between the function values at two consecutive accepted points is less than TOLERANCE·F(X) then the convergence is considered to be slow. If this behavior persists for more than MAXSLOW times in a row, then the surface is considered to be "flat" and the program stops. The default value of TOLERANCE is .00001.

MINSTOP: MINSTOP provides some integrated control over the above three switches. If MINSTOP is set, then the relation between MINSTOP and other switches are: EPS = .001·MINSTOP; TOLERANCE = .01·MINSTOP and RHO = MINSTOP. By setting MINSTOP, the values of EPS, TOLERANCE and RHO will be adjusted accordingly.

KEEPHESS: KEEPHESS may be used to obtain the Hessian matrix H or its inverse H^{-1} at the exit of MINIMIZE. (for maximum likelihood estimation, the Hessian inverse is an estimate of the covariance matrix of the parameters). KEEPHESS = 1 means return the Hessian in the MLAB matrix variable HESSIAN, KEEPHESS = 2 means return the inverse of the Hessian in HESSIAN, and KEEPHESS = 0 means no Hessian or Hessian inverse is needed at the exit.

HESSMSW: HESSMSW determines the method for computing an estimated Hessian matrix at each point where this computation is invoked. A computed Hessian matrix will be inverted and modified by the BFGS update process until it proves unsatisfactory. If HESSMSW = 0, the identity matrix is used as each initial Hessian. This causes the search direction to follow the line of steepest descent from the current point. If HESSMSW = 1 or -1, then forward differencing is used for the Hessian computation; HESSMSW = -1 also means to force the diagonal of the Hessian to be magnified as required to make it positive definite. If HESSMSW = 2 or -2, then centered differencing is used for the Hessian computation, HESSMSW = -2 also means to force the diagonal of the Hessian to be magnified as required to make it positive definite.

NEWHESS: NEWHESS is a switch to control the method of updating the Hessian. When NEWHESS $\geq$ 0, the Hessian will be recomputed after each, at most, NEWHESS+1 iterations. Note that the program automatically recomputes a new Hessian whenever a constraint is added to or removed from the active set or when the BFGS Newton step appears to be too large. This is decided by the comparison of the current BFGS Newton step with those of the previous steps. If NEWHESS=-1 the program will determine heuristically when to recompute the Hessian. Note that NEWHESS = 0 means recompute the Hessian matrix at each iteration.

PRECONDITIONSW: PRECONDITIONSW controls the scaling of the inverse Hessian whenever the inverse Hessian is computed. If PRECONDITIONSW = -1, the inverse Hessian is never preconditioned. If PRECONDITIONSW = 0, the inverse Hessian is always preconditioned. If PRECONDITIONSW = 1 and there is a significant difference in the relative magnitude of the largest and smallest diagonal elements of the inverse Hessian, then the inverse Hessian is preconditioned. "Preconditioning" is done as follows: the off-diagonal elements of the inverse Hessian are set equal to zero and the diagonal elements are scaled in a manner that will avoid erratic searches. If 0<PRECONDITIONSW<1, a less stringent condition for preconditioning is used based on the ratio of diagonal matrix elements of the inverse Hessian; if PRECONDITIONSW > 1, a more stringent condition is used based on this ratio.

GRADSW: GRADSW is a switch for controlling the generation of the search direction. GRADSW = 0 is the default value which means use $-H^{-1} \cdot \text{grad}(F)$ as the search vector, where H is the of the Newton or BFGS Hessian matrix. When GRADSW = 1, the averaged anti-gradient produced by the conjugate gradient method is used as the search vector. When GRADSW = 2, the pure gradient method is used, i.e. the anti-gradient is used as the search direction, and no Hessian is needed.

SYMDSW: SYMDSW may take on the values 1 (for symbolic derivatives) or 0 (for numerical derivatives). In the former case, symbolic partial derivatives are computed in construction of the gradient. In the latter case, numerical partial derivatives are used. For differential-equation-defined functions, numerical derivatives are always used, since symbolic derivatives are not computable. MLAB has the constant scalars SYMBOLIC (1) or NUMERICAL(0), available for assigning an appropriate value to SYMDSW. The default value of SYMDSW is 1 (SYMBOLIC).

Ordinarily, symbolic partial derivatives are preferable for reasons of accuracy. However, when we have a complicated function, the construction of the partial derivatives may occasionally exhaust the space alloted for functions in MLAB. In this unusual circumstance, it is possible to restart MLAB in its initial state, specify SYMDSW as NUMERICAL, and retry the optimization.

STOCHSW: STOCHSW is a switch that controls whether a stochastic function is to be minimized, and what method is used. STOCHSW = 0 is the default value which means that ordinary minimization for a non-stochastic function is used. i.e. the gradient and hessian are computed using symbolic derivatives or centered-differences as controlled by SYMDSW. When STOCHSW $\neq$ 0, we have a stochastic function to be optimized; such a function is computed with error, perhaps because it involves some Monte-Carlo simulation computations.

When STOCHSW$\neq$ 0, we compute the Hessian matrix by fitting a quadratic (parabolic bowl) model to sample points of the objective function F, and we compute the gradient by fitting a hyperplane to sample points instead of using symbolic-derivatives or finite-differences as in the non-stochastic case. In particular, when STOCHSW = 1, the gradient-determining hyperplane is center-fixed, i.e. the fitted hyperplane has to pass through the point at the current parameter vector value. When STOCHSW = 2, the center point is treated like all the other sample points and the fitted plane is more general. The advantage of STOCHSW = 1 is that we don't have to compute a matrix-inverse every time.

SSIZE: SSIZE is only used when we are minimizing a stochastic function. In that case, every time we compute a function value, we actually compute SSIZE function values and return their average.

When we compute the gradient by means of plane fitting, we also estimate the "noise" in the objective function. If this noise is too large, we will internally increase SSIZE. (SSIZE is bounded above by MAXSIZE).

MAXSIZE: MAXSIZE is the upper limit for SSIZE.

NSIZE: NSIZE is only used when we are minimizing a stochastic function. In that case, whenever we wish to compute a gradient by means of plane fitting, we compute a collection of points on the graph of the function in a neighborhood of the place whereat we wish to compute the gradient, and then fit a plane to these sample points. (the gradient is the normal of the fitted plane). The center point of the sample points is the point where we wish to compute the gradient. We compute the center point by computing NSIZE averaged function values, and averaging them; at the same time, we estimate the "noise" in our objective function. This estimated noise value is used to modify the value of SSIZE and also the neighborhood radius used for sampling. The noise level is estimated as the relative standard deviation of 10 function values evaluated at the same point.

REPTSW: REPTSW is a switch controlling the reports to be generated as the program proceeds. The resulting reports depend on the bit settings in REPTSW: (bit 1 is the rightmost, low order bit.) For example, REPTSW=515 selects the options controlled by bits 1, 2, and 10 defined below.

REPTSW (bit) controls

- 1(1) prints "backtracking to start of iteration #" whenever the backtrack procedure is invoked. (This may be useful or observing how the program responds to a given setting of BACKTRACKSW.)
- 2(2) prints the components of the unit vector directed along the initial search step at each iteration and also the length of the initial search step.
- 3(4) prints the parameter vector x,the associated function value F(x), and the associated gradient vector grad(F)(x) at the start of each iteration.
- 4(8) generates a message whenever a new Hessian is evaluated. prints "computing Hessian" after a new finite difference Hessian is computed, or "computing Hessian = I" when the Hessian is set equal to the identity matrix.

- 5(16) generates a message whenever a new inverse Hessian is evaluated.
- 6(32) prints the inverse Hessian's scale index, *sc*, whenever it is determined or changed. (This may be useful for observing how the program responds to a given setting of PRECONDITIONSW.)
- 7(64) generates a message whenever an existing inverse Hessian is modified, updated, or conditioned. MLAB prints "BFGS-updating inverse Hessian" before the inverse Hessian is updated using the BFGS formula, prints "conditioning inverse Hessian" when the scale parameter indicates that the inverse Hessian requires scaling, and prints "modifying diagonal of inverse Hessian" when a small positive number is added to the diagonal elements of the inverse Hessian because convergence is slow or the search direction is not descending.
- 8(128) MLAB prints "add constraint" or "delete constraint" and a constraint number each time a constraint is added to or deleted from the active set.
- 9(256) MLAB prints all step lengths used within each iteration to probe the function F. If there is more than one step length for a given iteration, then a line search has occurred in that iteration.
- 10(512) MLAB prints the final result, number of function evaluations, number of gradient evaluations, and number of Newton steps before exit.

29·5. LINPROG(F,C[,P])[3] linear programming by the simplex method

LINPROG(F,C[,P]) determines a vector X that minimizes the linear function F(X) subject to a set of linear equality and inequality constraints on the components of X that are specified in the constraint set C. The optional scalar argument P, if provided specifies whether the components of X are to be regarded as non-negative (P=0) or not. If omitted, P defaults to 0.

Example:

```
FCT F(X1,X2,X3) = X1+X2+X3
CONSTRAINTS Q = {X1<1, X2<1, X3<1, X1+X2<1.5, X1+X3<1.5, X2+X3<1.5, X1+X2+X3>1}
LINPROG(F,Q)
```

produces the result $x_1 = 0.75$, $x_2 = 0.75$, $x_3 = 0.75$, and the minimum value of the objective function $= 2.25$.

29·6. SIMPLEX(F,C,R,B[,P])[3] linear programming by the simplex method

SIMPLEX(MF,MC,MR,MB[,P]) determines a vector X[1:n] that minimizes the linear function $F = \sum_{i=1}^{n} F_i x_i$ subject to the constraints specified by C, R, and B, where C is an $r \times n$ matrix, $R[1:n]$ is a vector whose elements take on the values -1, 0, or 1, corresponding respectively to $<$, $=$, and $>$, and $B[1:n]$ is a vector of constants. C, R, and B thus represent the constraints $[C\ X\ R\ B]$. The optional scalar argument P, if provided specifies whether the components of X are to be regarded as non-negative (P=0) or not. If omitted, P defaults to 0.

Example:

```
MF = LIST(1,1,1); MR = LIST(-1,-1,-1,-1,-1,-1,1)
MB = LIST(1,1,1,1.5,1.5,1.5,1)
MC = SHAPE(7,3,LIST(1,0,0,0,1,0,0,0,1,1,1,0,1,0,1,0,1,1,1,1,1))
SIMPLEX(MF,MC,MR,MB)
```

produces the result $x_1 = 0.75$, $x_2 = 0.75$, $x_3 = 0.75$, and the minimum value of $F = 2.25$. This example is the same one as was used in the documentation of the function LINPROG.

29·7. EVAL(X,A,E)^0 evaluate by substitution

EVAL(X,A,E) computes a scalar which is the value of an expression E obtained by computing E after substituting the value of the expression A, in place of each occurence of the formal variable X in the expression E. X is the name of the scalar variable for which substitution is to take place. A must be a scalar, the value of which is to be substituted for X. E is an expression in which substitution is to take place. EVAL can only be used in function bodies. It is useful in defining functions compactly.

29·8. SUM(I,A,B,E)^0 sum of a sequence

SUM(I,A,B,E) computes a scalar which is the sum of the terms generated by the expression e. The symbol I is the name of the formal summation index; it is treated as a scalar. A and B must be scalars; they are the lower and upper limits of the summation. E is a scalar expression, which ordinarily involves the symbol I. SUM produces a scalar, which is the value of the summation. SUM currently can only be used in function bodies.

Example: Let `FUNCTION S(N) = SUM(I,1,N,A*J)`, and A = 2. Then, S(10) = 110.

29·9. PRODUCT(I,A,B,E)^0 product of a sequence

PRODUCT(I,A,B,E) computes a scalar which is the product of the factors generated by the expression E. The symbol I is the name of the formal product index; it is treated as a scalar. A and B must be scalars; they are the lower and upper limits of the product. E is a scalar expression, which ordinarily involves the symbol I. PRODUCT produces a scalar which is the value of the product. PRODUCT currently can only be used in function bodies.

Example: FUNCTION P(N) = PRODUCT(I,1,N,I)/FACT(N). Then, P(5) = 1.

29·10. ROOT(X,A,B,E)^0 zero of a function of one argument

ROOT(X,A,B,E) computes a scalar which is a real-valued numerical solution of a single equation in a single variable. X is the name of the formal scalar variable in the equation E(X) = 0 which is to be solved. A and B must be scalars; they are the lower and upper limits for X within which the search for a solution to E(X) = 0 is to be made. E is a scalar expression. The symbol X should appear in E. ROOT produces a scalar value, v, such that $E(v) = 0$. If no root is found, a warning message is produced and the result is MAXPOS. ROOT can currently only be used in function bodies.

Example: Let FCT F(Z) = ROOT(X,Z,Z+PI,BESSJ(X,0)). Then F(Z) will be the value of a zero of the Bessel function J_o in the interval [Z,Z+π].

29·11. PROOT(M[,N])^0 zeros of a polynomial (real or complex)

PROOT(M[,N]) computes a 2-column matrix whose rows are the complex roots of the polynomial whose coefficients are specified in the matrix M and the optional matrix N, which defaults to (NROWS(M)-1):0:-1. M may have one or two columns. When M has one column, $M[i]$ is the coefficient of $x^{N[i]}$ in the polynomial

$$P(X) = M[1]x^{N[1]} + M[2]x^{N[2]} + \ldots + M[\text{NROWS}(M)]$$

which is assumed to have real coefficients. When M has two columns, $M[i,1]$ is the real part and $M[i,2]$ is the imaginary part of the coefficient of the term in $x^{N[i]}$ in the polynomial which

has complex coefficients. PROOT uses Newton's method with successive deflation. However, the Newton iterations are always done on the entire polynomial, so that high accuracy is obtained.

29·12. ITERATE(F1'T, F2'T,...,X)[1] solve first order difference equations

ITERATE(F1′ T, F2′ T,...,X) constructs a matrix which is the solution to a set of first order difference equations of the form:

$x_{n+1} = f_x(x_n, y_n, \ldots)$, with initial value $x_0 = x0$
$y_{n+1} = f_y(x_n, y_n, \ldots)$, with initial value $y_0 = y0$
...

In MLAB, difference equations are represented using derivative functions (with the ′ notation) although there is no connection to differential equations. Initial conditions for the difference equations are provided using the initial statement. Thus the MLAB statement of the above system of difference equations is

```
FCT X′ T(T) =FX(X, Y, ...,T)
INIT X(0) = X0
FCT Y′ T(T) = FY(X, Y, ...,T)
INIT Y(T0) = Y0
...
```

The derivative notation is commandeered so that $x't(t)$ is interpreted as $x(t+1)$. Only first order difference equations may defined directly. However, just as in the case of higher order differential equations, higher order order difference equations may be defined by the use of a sequence of first order difference equations. Hence, to define the difference equation: $x(n+2) = x(n+1)/x(n)$, we would introduce auxiliary functions, leading to the system of two first order differential equations: $y(n+1) = y(n)$ and $z(n+1) = y(n)/z(n)$.

ITERATE takes a variable number of arguments. The first $k-1$ arguments of k given arguments must be the names of derivative functions defined by FUNCTION statements, each with a corresponding initial condition, defined by INITIAL statements, usually, although not necessarily, with an initial argument of 0. The final argument, X is a scalar, the number of iterations to be made, counting the initial conditions as iteration #1.

The argument for initial conditions in iteration functions is ignored, but, since it must be specified, zero is convenient. The result is an $r \times (n+1)$ matrix, H; H ROW 1 contains the initial conditions; row $i+1$ contain the solutions for iteration i. Column 1 of H contains the integers 1:x, and column j contains the solution for the i-th function in the set.

29·13. POINTS(F1[,F2...],M)[0] evaluate function(s) on a list of argument vectors

POINTS(F1[,F2...],M) constructs a matrix which contains the values of a list of one or more functions evaluated on a list of argument vectors. F1, F2, ... are the names of k scalar-valued user-defined direct functions, differential-equation-defined functions or builtin functions, which each have c formal arguments. M must be an $r \times c$ matrix. POINTS(F1[,F2...],M) produces an $r \times (c+k)$ matrix result V. In computing V, each row of M, M ROW i, is taken as an actual argument list for each of F1, F2, ..., and is copied into the first c columns of V in row i. Then, each function F_j is evaluated on the argument list, with missing optional arguments supplied as pre-determined default values, and the result placed in V COL $(c+j)$, for $j = 1, 2, \ldots$.

If some function F is a differential-equation-defined function, all the other functions must be also. Moreover, M must have 1 column, the rows of which are successive values of the independent variable at which the differential-equation-defined function(s) are to be evaluated by numerical integration. All of the differential equations which are members of the system of ODE's which must be solved

to obtain the values of the specified functions are identified automatically; the entire system not be specified explicitly if only some of the members are of interest. The current values of the differential equation solver control variables (METHOD, ERRFAC, ODERPT, JACSW) apply when differential equations are solved by use of the POINTS operator; however MANDSW is effectively taken as 1, although its value is not changed.

29·14. MAPPLY(F,M)^0 apply a function to matrix indices

MAPPLY(F,M) constructs a matrix with the same shape as the input matrix M. F must be a function with 1, 2, or 3 arguments. M must be a matrix. There are three cases.

If F has one argument, then the $[i, j]$-th element of the result matrix is equal to $F(M[i, j])$.

If F has two arguments, then the $[i, j]$-th element of the result matrix is equal to $F(i, j)$.

If F has three arguments, then the $[i, j]$-th element of the result matrix is equal to $F(i, j, M[i, j])$.

29·15. LINLOG([F,]M)^0 evaluate a function or data for linlog drawing

LINLOG([F,]M) constructs an NROWS(M) × 2 matrix R suitable for semilog drawing, where the x-coordinate is linear and the y-coordinate is base-10 logarithmic. There are two cases.

• M is a 1-column matrix, and F is a function of a single argument. The result R is a 2-column matrix containing the points of F ON M, transformed such that R COL 1 = M, and R COL 2 = LOG10 ON (F ON M).

• M is a 2-column matrix, and F is omitted. The result R is a 2-column matrix containing the points of M transformed such that: R COL 1 = M COL 1, and R COL 2 = LOG10 ON (M COL 2).

Example:

Let `F(X) = A*EXP(-B*X)`, and suppose we want to draw a semilog plot of log(F(X)) vs. X on 0:10!100 . Then the statement DRAW LINLOG(F,0:10!100) will draw the desired graph, which is a straight line that intersects the y-axis at A and has slope -B. The x-limits of the curve will be 0 and 10.

29·16. LOGLIN([F,]M)^0 evaluate a function or data for loglin drawing

LOGLIN([F,]M) constructs an NROWS(M) × 2 matrix R suitable for semilog drawing, where the x-coordinate is base-10 logarithmic and the y-coordinate is linear. There are two cases.

• M is a 1-column matrix, and F is a function of a single argument. The result R is a 2-column matrix containing the points of F ON M transformed such that R COL 1= LOG10 ON M, and R COL 2 = F ON M.

• M is a 2-column matrix, and F is omitted. The result is a two column matrix R containing the points of M transformed such that: R COL 1 = LOG10 ON (M COL 1) and R COL 2 = M COL 2.

Example: Let F(X) = (A-D)/(1+(X/C)^B), and suppose we want to draw a log-linear plot of the values of F(X) vs. log(X) on the data matrix M = 10^(-3:3!100). Then the statement DRAW LOGLIN(F,M) will draw the desired graph,which is a semilog plot of a descending s-shaped curve with upper asymptote A, lower asymptote D, midpoint C, and midpoint slope B, over the logarithmic range -3 to 3.

29·17. LOGLOG([F,]M)^0 evaluate a function or data for log-log drawing

LOGLOG([F,]M]) constructs an NROWS(M) × 2 matrix R suitable for log-log drawing. There are two cases.

• M is a 1-column matrix, and F is a function of a single argument. The result R is a 2-column matrix containing the points of F ON M transformed such that: R COL 1 = LOG10 ON M, and R COL 2 = LOG10 ON (F ON M).

• M is a 2-column matrix, and F is omitted. The result R is a 2-column matrix containing the points of M transformed such that: R COL 1 = LOG10 ON (M COL 1) and R COL 2 = LOG10 ON (M COL 2).

Example: Let F(X) = (B+(X-A)^(D/C)). Then the statement DRAW LOGLOG(F,1:3!10) will draw a log-log graph of the equation (Y-B)^C = (X-A)^D for X = 1:3!10, which is a straight line with slope D/C passing through the point (A,B).

29·18. RPOLAR(F,M)^0 evaluate radial polar function for cartesian drawing

RPOLAR(F,M) constructs an NROWS(M) × 2 matrix S containing the points of the radial polar coordinate function F ON M transformed for cartesian drawing. F must be a function with 1 argument which returns the azimuthal coordinate output value θ corresponding to a given radial coordinate input value R. M must be a 1-column matrix .

The function F is evaluated at the radial coordinate values specified in M. The resulting matrix of (r, θ)-coordinate pairs is converted to a corresponding list of (x, y)-coordinate pairs using the polar-cartesian conversion formulas: $x = r\cos(\theta)$, $y = r\sin(\theta)$.

Example:

Suppose we want to draw a polar graph of the equation $4\sin(3\theta) = r^2(r-1)$ for $r = 0 : 1!100$. Define the MLAB function F(R) = (1/3)ASIN(R*R*(R-1)/4). Then typing DRAW RPOLAR(F,0:1!100) will draw the desired graph.

29·19. TPOLAR(F,M)^0 evaluate azimuthal polar function for cartesian drawing

TPOLAR(F,M) constructs an NROWS(M) × 2 matrix S containing the points of F ON M transformed for polar drawing. F must be a function with 1 argument which returns the radial coordinate output value r corresponding to a given azimuthal coordinate input value θ. M must be a 1-column matrix .

The function F is evaluated at the azimuthal coordinates specified in M. The resulting matrix of (r, θ)-coordinate pairs is converted to a corresponding list of (x, y)-coordinate pairs using the polar-cartesian conversion formulas: $x = r\cos(\theta)$, $y = r\sin(\theta)$.

Example: Suppose we want to draw a polar graph of the equation $r = [\sin(\theta/3)]^3$ for $\theta = 0 : (6\pi)!100$. Define the MLAB function F(T) = SIN(T/3)^3. Then typing DRAW TPOLAR(F, 0:(6*PI)!100) will draw the desired graph.

29·20. PPOLAR(F,G,M)^0 parameterized polar functions for cartesian drawing

PPOLAR(F,G,M) constructs an NROWS(M) × 2 matrix S containing the points of the radial polar function F and the azimuthal polar function G ON M, transformed for cartesian drawing. F must be a function with 1 argument which returns the radial polar coordinate R corresponding to a given parameter value. G must be a function with 1 argument which returns the azimuthal polar coordinate θ corresponding to a given input parameter value. M must be a 1-column matrix of parameter values.

The functions F and G are evaluated at the parameter values specified in M to produce the result matrix of (r, θ)-coordinate pairs which is then converted to a corresponding list of (x, y)-coordinate pairs using the polar-cartesian conversion formulas: $x = r\cos(\theta)$, $y = r\sin(\theta)$.

The function CPR(M), where M is a 2-column matrix of (r, θ)-coordinate pairs, may also be used to prepare polar coordinate data for drawing.

Example: Let F(T) = SIN(T^3), and G(T) = 1/SQRT(1-T^2). Then typing DRAW TPOLAR(F,G, 0:1!100) will draw a graph of the curve defined by the parametric equations $r(t) = \sin(t^3)$; $\theta(t) = 1/(1-t^2)^{1/2}$ for $t = 0:1!100$.

---

30. Input/Output Functions

30·1. READ(F[,X[,Y]])^0 read a matrix from a file

READ(F[,X[,Y]]) constructs a matrix with numbers read from a file. F specifies a file name. The optional arguments X and Y must be scalars. X is an integer value which specifies the maximum number of rows for the result matrix. If X is omitted, it defaults to 1000. Y is an integer value which specifies the number of columns for the result matrix. If Y is omitted, it defaults to one.

The filename argument F may be any of the following: an explicit name (which is not the name of a string variable), a string variable name, or a quoted string.

If F is an explicit non-string variable name, e.g. FOO, then it is assumed that the file extension is ".dat". If F is a non-string variable name which contains a dot , e.g. FOO. or FOO.OLD, the given extension will be used; specifically the null extension is used if no characters follow the dot in the name.

If F is a quoted string, or if F is the name of a string variable, the exact contents of that string will be used. In this case a directory path may be prefixed in the given string. Remember, for a DOS or Windows PC, a directory name is terminated with a backslash, for a Macintosh, a folder name is terminated with a colon, and in Unix, a directory name is terminated with a slash. Thus, for a PC, the argument F might be: "C:mlab\work\dataf.in". For a Macintosh, F might be "data folder:mlab data:dataf.in", and for Unix, F might be "/usr/local/data/dataf.in".

In any case, the filename specified by F is automatically prefaced by the FILEDIR string to form the full filename. Generally, the FILEDIR control variable holds the text specifying the desired directory path. If no file of that name is found, an error will result.

The filename specified by F can be either an absolute or relative file-name. If you specify a relative path, it will start from your current working directory. In Windows, the notion of *current working directory* is obscure, and on a Macintosh, it is non-existent, so beware of the vageries of your operating system.

The special filename CON is used in MLAB to denote the keyboard, treated as a file. Input by this mechanism must be terminated by a *CTRL-Z* or the Escape key. However, reading from the keyboard is often done more easily using the function KREAD.

If both X and Y are omitted, the result is a $k \times 1$ matrix, where k is the the minimum of 1000 and the number of numbers in the file. If both X and Y are specified, the result matrix will have X or fewer rows and Y columns. If there are more than $X \cdot Y$ numbers in the file F, only the first $X \cdot Y$ of them will be read-in. If there are fewer than $X \cdot Y$ numbers in the file, the last row, if incompletely filled, will be completed with zeros.

The numbers in the file F are used to fill the matrix M being read in row-wise order, i.e. the first number goes in M[1,1], the second number goes in M[1,2], ..., the Y-th number goes in M[1,NCOLS(M)], the Y+1-st goes into M[2,1], etc, where Y is the number of columns in M.

In the input file, strings of characters enclosed in double quotes (") are ignored. Double quotes embedded in a such a comment string must be indicated by two adjacent double quotes.

Example: Suppose that F is a string variable with F = "F.D". Then the statement M = READ(F) will read data from the file named F.D.

Suppose that the file F.D consists of the following lines.

```
"November 16, 1990–Rat liver weights"
 10124   14.829
 10178   17.828   10879   16.989   "this weight looks fishy"
 10214   23.829   10827   15.392
```

Then the statement M = READ("F.D",15,2) will cause the matrix below to be created.

$$M = \begin{bmatrix} 10124 & 14.829 \\ 10178 & 17.828 \\ 10879 & 16.989 \\ 10214 & 23.829 \\ 10827 & 15.392 \end{bmatrix}$$

30·2. KREAD([S[,X[,Y]]] | [X[,Y]])^0 read from the keyboard one or more numbers

KREAD([S[,X[,Y]]] | [X[,Y]]) constructs a scalar or matrix by reading one or more numbers from the keyboard. There are six options for the arguments: no arguments; a single string argument S followed by zero, one or two scalar arguments, X and Y; or one or two scalar arguments, X and Y.

The first optional argument S is a string that will be typed-out as a prompting message.

The optional argument X specifies the number of rows and the optional argument Y specifies the number of columns of the matrix result, so that $X \cdot Y$ is the number of numbers to be read. If X and/or Y is not specified, 1 is provided by default.

If there are no arguments, a single number is read and returned as a scalar value. If, when a single number is requested, the user enters no number but simply presses the presses the *ENTER* key, then the value MAXPOS is returned. Testing for MAXPOS is a convenient way to determine that some default input value should be used.

If the first *and only* argument is a string, the given string will echo to the screen, and a single scalar will be read and echoed on the same line. This is convenient for dialog-directed input. Otherwise, this case is identical to the no argument case.

If X is given and Y is omitted, then X numbers are read, and an $X \times 1$ matrix of these values is returned. If both X and Y are given, then $X \cdot Y$ numbers are read, and an $X \times Y$ matrix is returned.

The numbers may be typed in free format, separated by spaces, tabs, or commas. Input lines are terminated by pressing the *ENTER* key. If the Escape and Return (Enter) keys are struck before the specified number of numbers has been given, the current row of the input matrix is completed with zeros, if necessary, and input is terminated. Excess numbers are ignored.

30·3. WREAD[X[,Y[,Z]]]^0 enter pairs of coordinates via the mouse

WREAD([X[,Y[,Z]]]) constructs a 2 column matrix by reading one or more pairs of coordinates in an MLAB graphics window that are specified with a pointing device (mouse). The optional argument X specifies the number of coordinate pairs to be read; it is 10 if not specified. The optional argument Y specifies the name of the MLAB window from which coordinates are to be read; it is the default window, W, if not specified. The optional argument Z specifies additional options; it is taken as 0 by default.

The second argument, Y, if supplied, must be the name of an existing window which was created by an MLAB graphics command.

The third argument, Z, specifies features of the mouse-cursor-position-reading process. If Z is 0, then WREAD stores the world coordinates of the mouse cursor position in MLAB window Y each time the mouse button is released, until X coordinate pairs have been stored. WREAD changes the pixel at each stored mouse cursor position to the foreground color. The user may terminate WREAD before X coordinate pairs have been stored by striking the Escape key. If Z is 1, WREAD will dynamically display the world coordinates of the mouse cursor position at the bottom left corner of the image rectangle of the specified window. If Z is 2, WREAD starts to read and store world coordinates of the mouse cursor position when the mouse button is pressed, and continue to read and store world coordinates of the cursor position as the mouse is dragged. WREAD stops reading and storing the world coordinates of the mouse cursor postion when the mouse button is released or when X coordinates have been stored. If Z is 3, both of the features for Z equal 1 and Z equal 2 are enabled.

In order to use this function on DOS and Windows systems, the mouse-driver software must be loaded prior to running MLAB. If the mouse-driver software is not detected by WREAD, an error message will be printed.

Example: Starting from a new MLAB session, type

```
TITLE T = "TITLE"
M = WREAD(1)
TITLE T AT (M[1,1],M[1,2]) WORLD
VIEW
```

This will display the default window with "title" printed at the bottom left corner of the MLAB graphics window and then reposition the title where ever the mouse is positioned when the mouse button is released.

30·4. TYPEOUT(X)^0 type-out a value from within a function

TYPEOUT(X) is a function which types-out on the display the value of its argument as a side-effect. X must be a scalar. TYPEOUT returns the value of its argument. TYPEOUT is useful in debugging functions.

Example: Define the MLAB function F(X) = 1+TYPEOUT(3+X). Then A = F(4) types 7.0 and sets A to 8.

30·5. ETIME()^0 elapsed time

ETIME() returns a scalar which is the elapsed time in seconds since the previous call to ETIME (or since the start of the current MLAB session if this is the first call). This function is useful for measuring the duration of MLAB calculations.

30·6. JDATE(M,D,Y)^0 Julian date

JDATE computes the Julian day number, which begins at noon on the calendar day specified by the month M, the day D, and the year Y, which are doubles that are truncated to integers.

Positive year value signifies AD; negative year value signifies BC. Note there is no year 0. The year after 1 BC is 1 AD. Julian day 2440000 starts on calendar day May 23, 1968 at noon.

Example: JDATE(5,23,1968) = 2440000.

30·7. DTYPE(R)^0 data type of an object

DTYPE(R) returns a scalar which is the data type of the object R. The MLAB datatypes and the corresponding numbers appear in the table below.

data type	datatype code
unknown	1
constant	2
scalar	4
function	5
matrix	6
initial condition	7
string	9
window	11
curve/axis	12
title	13
constraint set	14
string matrix	16

Examples: Let S be a scalar, M be a matrix, W be a window, I be an initial and U be undefined. Then, DTYPE(S) = 4, DTYPE(M) = 6, DTYPE(W) = 11, DTYPE(I) = 7, and DTYPE(U) = 1.

30·8. CORE()^0 available workspace

CORE() returns a scalar which is the size of the free workspace segment (in bytes) highest in memory. CORE is only defined in the DOS Overlay version of MLAB. Dynamically-allocated workspace is used for user-defined objects of type matrix, window, curve, title, and constraint set of MLAB.

Matrices are the main users of space in MLAB. Each number in a matrix uses 8 bytes. Thus, a 100×100 matrix uses 80,000 bytes. When the workspace is insufficient for a requested operation, a message will appear on the screen. The user should save the current MLAB state (with an SSAVE statement), and exit the program. A new session of MLAB may be started, and the saved state may be restored (with a reuse statement). At this point, unneeded objects, especially matrices and windows, should be discarded (with the DELETE statement). Frequently, enough space can be obtained in this manner to permit further operations.

30·9. KSREAD()^0 read a string from the keyboard

KSREAD() returns a string which is read from the keyboard. The string to be entered is terminated by the first white-space character (space, tab, or LF (*ENTER* key)) which appears in the input. Entry must be terminated by pressing the *ENTER* key.

This function provides a convenient means of entering a file name for use in a do-file.

30·10. READON(F[,X[,Y]])^0 read a matrix from a file without closing

READON(F[,X[,Y]]) constructs a matrix with numbers read from a file. It is identical to READ, except that it leaves the file open after a portion of it has been read. A subsequent READON applied to the same file continues where the previous READON operation left off. In this way, the numbers in a file may be processed piecemeal, even when there are too many to use simultaneously in MLAB due to memory limitations. If fewer than $X \cdot Y$ numbers remain in the file, then only a $\lfloor k/Y \rfloor \times Y$, matrix is returned, where k is the number of values remaining in the file; the last row is padded with 0-values as necessary.

If $X < 0$, the file F is immediately closed, and a 0-row null matrix is returned.

Example: Suppose the file F contains the numbers $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Then, the two MLAB statements: M1 = READON(F,5); M2 = READON(F,5) construct two matrices: M1 = (1,2,3,4,5)$'$, and M2 = (6,7,8,9,10)$'$.

30·11. GETSTRINGS(T,S[,H[,X[,Y[,C[,L]]]]])^0 displays a form and returns user entries

GETSTRINGS(T,S[,H[,X[,Y[,L]]]]) displays a tabular form with an optional title and a list of prompt-field and text-entry-field pairs on the screen. Generally, GETSTRINGS will be used within a DO-file.

The user selects each text-entry field to be edited using the cursor movement keys or the *TAB* key. The initial values for all the text-entry fields is recovered by pressing the *ESCAPE* key. The user quits the form fill-in process by pressing the *ENTER* key. On Macintosh and Windows systems, the form fill-in process may also be terminated by clicking the mouse on the *CONTINUE* button. Upon quitting, the form is removed from the display and a string array containing the contents of the updated text entry fields is returned.

The argument T is a string defining the title. If the string T is an empty string(""), neither the title nor its associated box-border are displayed. A box-border is formed with certain text-characters, as are all the other components of the tabular form. Each such character has a "foreground" color and a "background color".

The argument S is a 1-dimensional string array that contains the prompts (descriptive text) associated with each text-entry field. These prompt strings must be short enough so that the length of the prompt combined with the length of the text entry field is less than or equal to 78 characters.

The optional argument H is a 1-dimensional string array that contains the initial text values of the text-entry fields. If H is omitted, then the initial values of the text entry fields default to empty strings. If H has fewer elements than S, the initial values of the text entry fields with no corresponding H element default to empty strings. If H has more elements than S, then only the first NROWS(S) elements of H are used.

The optional argument X is an scalar integer that determines the particular text-entry field that is active when the form is initially displayed. If X is omitted, the initial active text-entry field defaults to 1.

The optional argument Y is a scalar or a 1-dimensional array that specifies the width (in characters) of the text-entry fields. If Y is omitted, the text-entry field width for all the text-entry fields defaults to a width of 10 characters. If Y is a scalar, the text-entry field width for all the text-entry fields defaults to the value of the scalar.

If Y is a 1-dimensional array, the width of each text-entry field is determined by the value of the corresponding element of Y. In all cases, if an initial value for a text-entry field is defined by H *and* the width of the initial string is more than the width defined by Y, then the width will be expanded to accomodate the initial string defined in H.

For DOS systems, the optional argument C is a scalar or a 1- or 2-dimensional array that specifies the color attributes for each of the components of the text-entry form in the following order: the title text, the prompt text, the text-entry field text, the title-box border and the prompt/entry field-box border. If C is omitted altogether, all background colors are black and all text and box-border foreground colors are white. If C is a scalar, all background colors are the color defined by the value of that scalar. If C is a 1-column array, the background colors for each of the components are defined by the corresponding values of the array and the text and/or box-border foreground colors are white. If C is a 2-column array, the first column defines the background colors and the second column defines the text and box-border foreground colors.

For Macintosh and Windows systems, the argument C is ignored. All background colors are white and all text is black.

The optional argument L is a scalar that specifies the layout style of the tabular form. L determines the columnar layout of the prompts and entry fields. The width of the tabular form is determined by the overall maximum of the width of the title string and the maximum of the combined widths of the prompt and entry fields. We have the following options.

L = 1 (This is the default if the argument L is not given)

When L=1, the prompt strings are all left-justified in the form. The widths of all the prompts are the same, and this width is the width of the entire form less the width of the text-entry fields. The widths of each text-entry field is determined by the associated text-entry width value as determined by the entry width argument processing. The start of the text-entry field immediately follows the prompt field on the line. The visual result is that the prompts are all in a rectangular column, and the text-entry fields are all left justified against the prompt column while the right edge of the entry area will be ragged if the entry fields have different widths or uniform if all entry fields have the same width, as in the case of a constant or scalar entry width argument.

L = 2

When L=2, the prompt strings are left-justified in the form. The length of each prompt field is determined by the length of the prompt string. If all prompt strings have the same length, the prompt area is rectangular, otherwise the right edge of the prompt area is ragged. The text-entry fields start immediately after the associated prompt fields, and the text-entry field widths are determined by the entry width argument, so in general both the left and right sides of the entry area are ragged.

L = 3

When L=3, the prompt strings are left-justified in the form. The width of each prompt field is determined by the length of the prompt string. If all prompt strings have the same length, the prompt area is rectangular, otherwise the right edge of the prompt area is ragged. The text-entry fields start immediately after the associated prompt fields, and the entry widths are extended to the right edge of the window, so that the right edge of the entry area is uniform.

L = 4

When L=4, we obtain a layout identical to the layout specified by L = 1 except that the prompt strings are right justified in the prompt area.

30·12. MENUCHOICE(T[,S[,X[,C]]])^0 display a menu and return user choice

MENUCHOICE(T[,S[,X[,C]]) displays a menu with a title and a list of text strings specifying a set of k choices to be made. For Macintosh and Windows systems, a "radio-button" appears to the left of each text string. Generally MENUCHOICE will be used within a DO-file.

By use of the cursor keys and the *TAB* key, the user may select one member of this list. When the *ENTER* key is pressed, an integer value is returned that is the sequence number in $\{1, 2, \ldots, k\}$ of the selected string.

The argument T is a string title or string array for the menu. If T is a 1-column string array, each element of the string array appears on a different line.

The optional argument S is a 1-column string array containing the text strings that will be displayed as the menu choices. If S is not provided, only the title will be displayed until the user strikes the RETURN/ENTER key; this provides a means of presenting instructions when running MLAB do-files.

The optional scalar argument X specifies one of the strings in the menu as the default selection. If omitted, the default value of X is 1, corresponding to the first string. In DOS, the currently-selected string is displayed in reverse-video; in Macintosh and Windows, the radio-button corresponding to the currently-selected string is filled in. If X $<$ 1, the first string in S is initially selected.

For DOS systems, the optional scalar or array C specifies the color choice as an integer for each of the elements of the menu in this order: the title text, the title box-border and the title background, the menu text,the menu box-border, and the menu background. If C is a 2-column matrix, column 1 contains the background color while column 2 contains the foreground color. If C is a 1-column matrix, it contains the background color and the text color defaults to white.

Row 1 of the array C pertains to the title; row 2 pertains to the menu items; row 3 pertains to the title box, and row 4 pertains to the menu box. If rows 2-4 are omitted, the information from row 1 is used for all items. If rows 3 or 4 are omitted, the information from rows 1 and 2 is used respectively.

If C is a scalar, it defines the background color, while all text and box-borders are white.

If C is omitted, the menu text, title text, menu box-border, and title box-border all default to white, and the backgrounds all default to black.

For Macintosh and Windows systems, the argument C is ignored. All backgrounds are white and all text is black.

30·13. RUN(S[,M])^0 Run an independent program returning a matrix

In Windows, the MLAB RUN function can be used to execute other Windows programs with the exchange of an argument array of floating point numbers via the Clipboard. RUN takes 2 arguments. The first argument, S, is mandatory; S is a string containing the name of a Windows program. The second argument, M, is an array (a matrix) of numbers having nr rows and nc columns. If M is not given, a null matrix is assumed.

If the array M is provided, RUN places the nr*nc floating point numbers in the Windows Clipboard as data of type CF_TEXT. An additional number is placed in the Clipboard. In particular, the first number is the value nr*nc specifying the number of subsequent numbers, and the remaining numbers are M[1,1], ..., M[1,nc], M[2,1], ..., M[2,nc], ..., M[nr,nc]. RUN then calls the MS Windows API function WinExec() with S as its first argument and SW_SHOWNORMAL as its second argument to execute the program named in S.

The specified program is expected to access the CF_TEXT data in the Clipboard, and remove it from the Clipboard. When the specified program has finished its computation and is ready to return control to MLAB, it must supply its results to MLAB via a Clipboard message of type CF_TEXT. The supplied results must be a sequence of floating point values. These values are to be prefaced by their number when placed in the Clipboard.

When the MLAB console window becomes the active window again (i.e. when MLAB receives control again), RUN checks the contents of the Clipboard. If the Clipboard data is of type CF_TEXT, the first 8 bytes are interpreted as the number of subsequent floating point numbers stored in the remainder of the Clipboard, and RUN returns a 1 column array with those numbers.

If the Cliboard data is not of type CF_TEXT (or if the number of supplied result values is 0), a null array is returned.

If the program being run with the RUN function does not remove the supplied CF_TEXT message from MLAB that encodes the array M, then that message will be present when MLAB continues and it will be taken as results from the program that was run. Example: RUN("NOTEPAD",1:10) followed by Alt F-X in the NOTEPAD program returns the 1 column array [1,2,3,4,5,6,7,8,9,10]'.

The RUN function is programmed to pause until the MLAB text console window becomes "active". This assumes that the program S being run establishes its own window which becomes the "active" window, thereby causing MLAB to pause. If the program S does not appropriate the "active" status for one of its windows, then MLAB will continue to run, multiplexed with the program invoked via RUN. This is the case for example if RUN("COMMAND.COM") is executed. Clearly this can lead to some complex behavior.

In DOS, the RUN function is implemented as a "system" invocation. That is the text S is interpreted as a DOS command and a temporary new copy of the DOS COMMAND.COM program is loaded in order to execute it. For example, in DOS, typing RUN("PRINT F") will spool the file F for printing. In DOS, any second argument supplied to RUN is ignored and a null matrix is unconditionally generated as the result.

30·14. SRUN(S[,M])^0 Run an independent program returning a string array

In Windows, the MLAB SRUN function can be used to execute other Windows programs with the supplying of a floating point argument array via the Clipboard. An array of strings may be returned via the Clipboard. SRUN takes 2 arguments. The first argument, S, is mandatory; S is a string containing the name of a Windows program. The second argument, M, is an array (a matrix) of numbers having nr rows and nc columns. If M is not given, a null matrix is assumed.

If the array M is provided, then SRUN places the nr*nc floating point numbers in the Windows Clipboard as data of type CF_TEXT. An additional number is also placed in the Clipboard. In particular, the first number is the value nr*nc specifying the number of subsequent numbers, and the remaining numbers are M[1,1], ..., M[1,nc], M[2,1], ..., M[2,nc], ..., M[nr,nc]. SRUN then calls the MS Windows API function WinExec() with S as its first argument and SW_SHOWNORMAL as its second argument to execute the program named in S.

The specified program is expected to access the CF_TEXT data in the Clipboard, and remove it from the Clipboard. When the specified program has finished its computation and is ready to return control to MLAB, it must supply its results to MLAB via a Clipboard message of type CF_TEXT. The supplied results must be a sequence of character strings. These strings are to be prefaced by their number when placed in the Clipboard. In particular, the first two bytes are interpreted as a 16-bit integer equal to the number of 0x00-terminated strings stored in the remainder of the Clipboard data area.

When the MLAB console window becomes the active window again (i.e. when MLAB receives control again), SRUN checks the contents of the Clipboard. If the Clipboard data is of type CF_TEXT, the first 2 bytes are interpreted as the number of subsequent character strings stored in the remainder of the Clipboard, and SRUN returns a 1 column string-array with those strings as elements.

If the Cliboard data is not of type CF_TEXT (or if the number of supplied result values is 0), a null string-array is returned.

If the program being run with the SRUN function does not remove the supplied CF_TEXT message from MLAB that encodes the array M, then that message will be present when MLAB continues and it will be taken as character string results from the program that was run. Clearly, this can lead to a foolish string array arising as the result of SRUN.

The SRUN function is programmed to pause until the MLAB text console window becomes "active". This assumes that the program S being run establishes its own window which becomes the "active" window, thereby causing MLAB to pause. If the program S does not appropriate the "active" status for one of its windows, then MLAB will continue to run, multiplexed with the program invoked via SRUN. This is the case for example if SRUN("COMMAND.COM") is executed. Clearly this can lead to some complex behavior.

In DOS, the SRUN function is implemented as a "system" invocation. That is the text S is interpreted as a DOS command and a temporary new copy of the DOS COMMAND.COM program is loaded in order to execute it. For example, in DOS, typing SRUN("PRINT F") will spool the file F for printing. In DOS, any second argument supplied to SRUN is ignored and a null string array is unconditionally generated as the result.

---

31. String-valued Functions

31·1. STRVAL(Q)^0 build a string from a defined object

STRVAL(Q) returns a string which is the same as the string which would be produced as screen output if q was typed using the TYPE statement. q must be a defined object of one of the types: scalar, matrix, function, initial, or conset. STRVAL is useful in creating titles with which to label a graph.

Examples: Let FCT F(X) = SIN(X) and A = 5. Then,
STRVAL(F) returns the string "FUNCTION F(X) = SIN(X)".
STRVAL(A) returns the string "A = 5".

31·2. STRTIME()^0 build a string with the current time

STRTIME() returns a string containing the current time in the format hh:mm:ss using a 24 hour clock. This function is useful for labeling graphs, print files, and the log file.

Example: Let A = STRTIME(). Then A = "13:08:49"

31·3. STRDATE()^0 build a string with the current date

STRDATE() returns a string containing the current date in the format mmm-dd-yyyy. This function is useful for labeling graphs, print files, and the log file.

Example: Let A = STRDATE(). Then A = "AUG-23-1990"

31·4. TITLESTR(T)^0 get the string of a title

TITLESTR(T) returns a string containing the text of a title.

Example: Suppose we had made a title with the statement: TITLE T43 = "GOOD". Then the statement: A = TITLESTR(T43) would set the string variable A to the string "GOOD".

31·5. SUBSTR(T,A)^0 get a substring of a string

SUBSTR(T,A) returns a string which is a substring of the string T. A may be a scalar (which is treated as a 1-row matrix) or a matrix. The elements of A are taken as indices of the characters in T in order to construct the output string. If any (or all) of the elements of A address non-existent elements of T, then an empty string will be produced.

Example: Let T = "abcdefghij", and A = [1,1,9,9,2,3,4]′ . Then SUBSTR(T,A) = "aaiibcd". Let B = 4:7. Then SUBSTR(T,B) = "efgh".

31·6. STRLEN(S)^0 length of a string

STRLEN takes a string as an argument and returns a integer that is the number of characters that make-up the string argument.

Example: If S = ”a string”, then STRLEN(S) = 8.

31·7. STRREV(S)^0 reverse a string

STRREV takes a string as an argument and returns a string that is the reverse of the argument.

Example: If S = ”string”, then STRREV(S) = ”gnirts”.

31·8. STRDBLANK(S)[0] deblank a string

STRDBLANK takes a string as an argument and returns a string with leading blanks removed from the string.

Example: If S = " string", then STRDBLANK(S) = "string".

32. Cluster Analysis Functions

32·1. ASCALE(M[,X])[7] translation to zero mean and scaling to unit variance

ASCALE(M[,X]) computes a matrix N which is a rescaling of the input so that each column has zero mean and unit variance. M must be a matrix. X must be a scalar. If X is omitted, it defaults to one. The result has the same shape as M. If the range of any column of M is zero (i.e. if all the values in a column are identical), then the values in that column of the output will be unchanged.

In the output matrix N, N[i, j] is related to the input M as follows:

N[i, j] = (M[i, j]-MEAN(M COL j))/STDDEV(M COL j).

where MEAN is the arithmetic mean, and STDDEV is the biased standard deviation with denominator NROWS(M) when X = 0, and STDDEV is the unbiased standard deviation with denominator NROWS(M)-1 when X $\neq$ 0.

32·2. RSCALE(M)[7] minimum/maximum scaling

RSCALE(M) computes a matrix N which is a linear rescaling of the elements of the columns of the input matrix M, so that each column of the result matrix has range from 0 to 1. M must be a matrix. If M has a single row, or if the range of any column of X is zero (i.e. if all the values in a column are identical), then all the values in that column of the output will be set to zero.

In the output matrix N, N[i, j] is related to the input M as follows:

N[i, j] = (M[i, j]-MINV(M COL j))/(MAXV(M COL j)-MINV(M COL j)).

32·3. DISTS(M[,P])[7] Minkowsky p-metric, Euclidean distance

DISTS(M[,P]) computes a matrix D containing the distances (measured in the P-norm) between each pair of rows of M. M must be a matrix and P must be a scalar. Let n = NROWS(M) and m = NCOLS(M). The rows of M consist of n points in m-dimensional space. P must be positive. If P is omitted, it defaults to 2, so that the 2-norm (euclidean distance) is used. If P $>$ 10, then the infinity norm (also known as the maximum norm) is used. If P $\leq$ 0, an error message is produced. The formula for the distance, d_{ij} ,between row i and j of a matrix M in the P-norm is

$$d_{ij} = \sum_{k=1}^{n} (|M_{ik} - M_{jk}|^P)^{1/P}, \text{ for } 0 < P \leq 10, \text{ and } d_{ij} = \max_k |M_{ik} - M_{jk}| \text{ for } P > 10.$$

This measure is sometimes called the Minkowsky P-metric, and also the L^P norm. Let $k = n(n-1)/2$. The result is a $k \times 1$ matrix D: $D[(i-1)(i-2)/2+j]$ = the P-norm distance between M ROWi and M ROW j, for $1 \leq j < i$.

32·4. FISHERRK(M,N)[7] Fisher's rank-correlation

FISHERRK(M,N) computes a matrix R which is Fisher's rank correlation for the columns of M. M and N must be matrices. The result is an NCOLS(M) $\times$ 2 matrix R, where $R[j,1]$ is the column number of M (corresponding to feature j given in MCOLj) and $R[j,2]$ is the Fisher rank-correlation weight associated with the j-th feature (column). The rows of R are ordered according to their associated weights, in R COL 2, with the highest appearing first.

M is an $m \times n$ data matrix, each column of which corresponds to a feature. N is an $n \times 1$ single column vector, each row of which contains the category number for the corresponding column of M. The category numbers in N must be positive integers. Let c = MAXV(N). The formula for Fisher's rank for category j is

$$R[j,2] = \sum_{k=1}^{c} \sum_{i=1}^{k-1} \frac{(\bar{M}[i,j] - \bar{M}[K,j])^2}{\frac{S[i,j]^2}{P[i]} + \frac{S[K,j]^2}{P[K]}}$$

where $P[i]$ is the number of samples in M COL j belonging to category i, $\bar{M}[i,j]$ is the mean of the samples in category i in M COL j, and $S[i,j]^2$ is the variance of the samples in category i in M COL j. The larger the Fisher rank correlation weight is, the better the categories are separated in that column.

32·5. PRCOMP(M)[7] principal components of a covariance matrix

PRCOMP(M) computes a matrix N containing the eigenvalues and principal component vectors corresponding to the input correlation or covariance matrix M. M must be an $n \times n$ square matrix. Let n = NROWS(M). The result is an $n \times n$ matrix N containing the (necessarily non-negative real) eigenvalues in column 1 and the corresponding right-eigenvectors of M in columns $2 : (n+1)$, where N col j is the right-eigenvector associated with the eigenvalue N[j-1,1].

M must be a positive semidefinite symmetric matrix of covariances or correlations, such as is produced, for example, by the COV and CORR functions. N COL 1 contains the eigenvalues of M, sorted in decreasing order. The principal component eigenvector corresponding to the eigenvalue of M in $N[i,1]$ is in N COL (i+1). Denote the sum of the eigenvalues by S. $N[i,1]/S$ is the fraction of the variance about the principal axis explained by the variation of component i.

32·6. NLM(M[,X[,Y[,N]]])[7] dimensionality reduction using nonlinear mapping

NLM(M[,X[,Y[,N]]]) computes a matrix which is the result of a heuristic stress-minimizing nonlinear point mapping of some original data, whose interpoint distances are given in M, into a (usually lower-dimensional) space of dimension Y. This is a form of Kruskal's multi-dimensional scaling. Stress is minimized when the distances matrix M is preserved as closely as possible. M must be a matrix. X and Y must be scalars. N must be a matrix. The result is an NROWS(M) $\times$ Y matrix holding the generated images of the points in the Y-dimensional space.

M is an $m(m-1)/2$ vector of distances, such as is produced by the DISTS function applied to a data matrix P containing m points in k-dimensional space. X is the number of iterations to be used in generating the mapping. If X is omitted, it defaults to five, and five iterations will be taken. Y is the dimensionality of the target space. If Y is omitted, it defaults to 2, and a 2-dimensional space will be used.

If N is given, it must be an $m \times Y$ matrix, whose rows will be the initially-guessed image points for the mapping. If N is omitted, a randomly-chosen initial mapping will be used. (To be able to repeat that initial guess, the random number generator should be initialized with some seed value (using a statement like DUMMY = RAN(SEED)) before executing NLM, and then re-initialized with the same seed value when repetition is desired). It should be noted that repeated use of NLM using different random starts will generate different results. Another useful choice for N is the first Y principal component vectors of the (unspecified) data P that produced the distances matrix M.

The iterations are performed so as to minimize

$$S = \sum_{j=1}^{m} \sum_{i=1}^{j} \frac{(d^*_{i,j} - d_{i,j})^2}{d^*_{i,j}}$$

where $d_{i,j}$ is the euclidean distance between row i and row j of P in the original (k-dimensional) space, and $d^*_{i,j}$ is the euclidean distance between row i and j of P in the image (Y-dimensional) space. The $d_{i,j}$ values are given in M. The $d^*_{i,j}$ values will be computed as follows:

$$d^*_{i,j} = Y \frac{d^2_{i,j} - d^2_{\min}}{d^2_{\max} - d^2_{\min}}$$

where $d_{\min}$ is the minimum of the interpoint distances in k-space, and $d_{\max}$ is the maximum of the interpoint distances in k-space. Thus $0 \le d^*_{i,j} \le Y$.

32·7. KMEANS(M,X[,Y[,N]])[7] K-means clustering

KMEANS(M,X[,Y[,N]]) computes a matrix which represents a partitioning of the data defined by the rows of M into clusters. M and N must be matrices. X and Y must be scalars with $1 \le X \le Y$. The result is a matrix R with NROWS(M) rows and Y-X+1 columns. Each successive column of R contains the cluster assignments for the rows of M with X clusters, X+1 clusters, ..., and, finally, Y clusters.

M is the data matrix, the rows of which are the coordinates of the points to be clustered. Let z = NCOLS(M) and w = NROWS(M). These points lie in a z-dimensional space. X is the smallest number of clusters to be considered. X must be a positive integer. Y is the maximum number of desired clusters. Y must be a positive integer greater than or equal to X. If omitted, the value of Y is taken to be the same as X. In any case, both X and Y must be smaller than w. The optional argument N is a $w \times 1$ matrix defining an initial clustering of the points of M into X clusters; N[i] is the initial cluster number to which M ROW i is taken to belong. The cluster numbers must be integers $1, \ldots, X$. If N is omitted, a random initial clustering will be computed.

If X or Y is not a positive integer between 1 and w, or if $X > Y$, an error message will be produced.

32·8. CLUSTINFO(M[,N])[7] cluster summary table

CLUSTINFO(M[,N]) computes a matrix which summarizes the characteristics of a clustering resulting from the application of one of the cluster generating functions. M and N must be matrices. M is the input data matrix, consisting of y rows of z-dimensional data. N is a $y \times 1$ vector of cluster numbers to which each of the y points of M belongs. If N is omitted, all the points are assumed to belong to cluster 1.

Let z = NCOLS(M). The result R is a $c \times (2z+1)$ matrix, where c is the number of distinct clusters, numbered sequentially from 1 to c. R ROW i contains the following information for cluster i:

R[$i, 1$] is the number of points in cluster i.
R[$i, 2 : (z+1)$] are the means for each of the z components of the points in cluster i.
R[$i(z+2) : (2z+1)$] are the standard deviations for each of the z components of the points in cluster i.

32·9. CLUSTERR(M[,N])[7] clustering error

CLUSTERR(M[,N]) computes a matrix which contains the squared error for each cluster resulting from a specified cluster assignment. M and N must be matrices. The result R is an $k \times 1$ vector,

where k = NCOLS(N), and R[j] contains the total clustering error for the data in M with the cluster assignments specified by N COL j, as defined below.

M is a data matrix, consisting of y rows of z-dimensional data points. N is a $y \times k$ matrix, each column of which is a list of cluster numbers corresponding to the y points of M. Each column of N contains positive integers between 1 and, at most, MAXV(N). If N is omitted, it defaults to 1, so all the points are assumed to belong to cluster 1.

Let P be a cluster of m points. The error for this cluster is defined to be $e(P) = \sum_{i=1}^{m} D(i,\beta)^2$, where the z-vector β is the mean of the m points, i.e. the center of the cluster P of m points, and $D(i,\beta)$ is the euclidean distance from the i-th point of the cluster to the cluster center β.

The total clustering error for a group of h clusters $P_1, P_2, \ldots, P_h$ is $\sum_{j=1}^{h} e(P_j)$.

32·10. DELAUN(M)[7] Delaunay triangulation

M is an $n \times 2$ matrix whose rows contain the (x, y) coordinates of n points. DELAUN returns an $m \times 6$ matrix defining the Delaunay triangulation of the coordinates, where m is the number of triangles generated.

A triangulation of a set of points is the maximal connection of the points with straight lines where no lines intersect except at their endpoints. The result is a set of triangles whose vertices are the original points. A Delaunay triangulation has the property that the circumcircle of any triangle contains no points of the set in its interior; it can be used to find Voronoi regions (see below).

Columns 1:3 of row i of the result contain the indices of the three triangles adjacent to triangle i, and columns 4:6 contain the indices (in $1, \ldots, n$) of the points which are the vertices of triangle i.

The matrix returned can be used with the DELCURVE operator and a DRAW statement to draw the triangulation, or with the VORCURVE operator and a DRAW statement to compute and draw the Voronoi regions for the points.

32·11. DELCURVE(M[,D])[7] construct matrix for drawing Delaunay triangulation

M is an $n \times 2$ matrix whose rows contain the (x, y) coordinates of n points and D is an $m \times 6$ matrix containing the information for drawing the m triangles for the Delaunay triangulation of the points. If D is not given then it is computed internally as DELAUN(M). DELCURVE creates a curve matrix to be used with linetype MARKER or VMARKER for drawing the triangulation.

32·12. DCIRCLES(M[,D])[7] matrix for circumcircles of Delaunay triangulation

DCIRCLES(M[,D]) produces a 2-column matrix suitable for use in drawing using linetypes MARKER or VMARKER. M is an $n \times 2$ matrix whose rows contain the (x, y) coordinates of n points and D is an $m \times 6$ matrix containing the information for drawing the m triangles for the Delaunay triangulation of the points. If D is not given then it is computed internally as DELAUN(M). Each group of rows in the result matrix beginning with a marker row represents 36 points on the circumference of a circle which circumscribes a Delaunay triangle of the point set in M.

32·13. VORREGIONS(M[,X[,Y[,N]]])[7] compute Voronoi diagram

VORREGIONS(M[,X,Y[,N]]) computes a 2-column matrix Q holding the midpoints of the line-segments that form the Voronoi regions of the points in M. The result matrix R contains n marker rows and is suitable for use in drawing using linetypes MARKER or VMARKER. M is an $n \times 2$ matrix whose rows contain the (x, y) coordinates of n points and N is an $m \times 6$ matrix containing the information for drawing the m triangles for the Delaunay triangulation of the points given in M.

If N is not given then it is computed internally as DELAUN(M). The Voronoi regions are clipped to within the rectangle R which is X units wide by Y units high centered over the points in M. If $X < 0$, then horizontal clipping is not done. If $Y < 0$, then vertical clipping is not done. If Q ROW j is a marker row, then Q$[j, 2]$ is the row index of the corresponding point in M.

32·14. VORCURVE(M[,X[,Y[,N]]])[7] construct matrix for drawing Voronoi diagram

VORCURVE(M[,X,Y[,N]]) computes the information defining the Voronoi regions for the points in M. M is an $n \times 2$ matrix whose rows contain the (x, y) coordinates of n points. N is an $m \times 6$ matrix containing triangulation information. If N is not given then it is computed internally as DELAUN(M). The Voronoi regions are clipped to within the rectangle R which is X units wide by Y units high centered over the points in M. If $X < 0$, then horizontal clipping is not done. If $Y < 0$, then vertical clipping is not done. VORCURVE returns a 2-column matrix that can be used with linetype MARKER or VMARKER for drawing the Voronoi diagram. Unlike VORREGIONS, each group of rows represents a section of a Voronoi region boundary, and not necessarily the entire boundary. VORCURVE will work for any triangulated planar graph, even though only the triangulation DELAUN(M) yields the dual Voronoi diagram.

32·15. VORSTAT(M[X[,Y[,N]]])[7] edge-count, area and perimeter of Voronoi regions

M is an $n \times 2$ matrix whose rows contain the (x, y) coordinates of n points. N is an $m \times 6$ matrix containing triangulation information. If N is not given, then it is computed internally as DELAUN(M). VORSTAT returns an $n \times 4$ matrix with each row containing four values (number of edges, area, perimeter, open/closed code) for each of the n Voronoi regions. The Voronoi regions are clipped to within the rectangle R which is X units wide by Y units high centered over the points in M. If $X < 0$, then horizontal clipping is not done. If $Y < 0$, then vertical clipping is not done. If any of the edges of a Voronoi region is outside the boundary of the rectangle R, then all four values are zero for that region.

---

33. Tree Cluster Analysis Functions

33·1. MST(D)[7] minimal spanning tree

MST(D) generates a 3-column output matrix Q defining the node connections and edges of a minimal spanning tree of the points numbered $1, 2, \ldots, m$ whose inter-point distances are in D. The argument D must be a 1-row or 1-column matrix of $m(m-1)/2$ distances derived originally from m points in a space of 2 or more dimensions. Frequently this distance vector will be produced by the MLAB DISTS function. The output Q is an $(m-1) \times 3$ matrix. Q COL 1:2 contain integer node number pairs, defining the $m-1$ edges of the minimal spanning tree; Q COL 3 contains the corresponding edge lengths.

Example: The 3 points, (1,1),(2,2) and (3,3), generate a 3-element distance vector of [1.4,2.8,1.4]. Given this distance vector, MST will produce a 2 row matrix. The first row of this matrix is [2,1,1.4], which indicates node 2 is connected to node 1 with an edge length of 1.4. The second row is [3,2,1.4] showing that node 3 is connected to node 2 with an edge length also of 1.4.

33·2. TREECURVE(M[,X[,N]])[7] tree/graph matrix for drawing

TREECURVE(M[,X[,N]]) computes an output matrix L with 2, 3, or 4 columns. L can be used with the DRAW statement to display the tree (or graph) corresponding to the edge table given in M.

The input argument M is a 2- or 3- column matrix of node connections, defining a graph (usually a tree). In M, each row defines one edge. For a tree, each node must be connected to at least one other node, there must be a path between any two nodes and no cycles are permitted. Thus, if there is a total of n nodes, then, for a tree, M must have $n-1$ rows.

The nodes are numbered with integers from 1 to n. M may have 2 or 3 columns, with the first two columns always being node connection pairs. If present, the third column of M contains the numeric label (usually the length) of the edge between the 2 nodes in columns 1 and 2. For example, if M ROW 1 = [1 2 3.3], this means that node 1 is connected to node 2 with edge length 3.3.

The optional input scalar X is the number of the node that is to be used as the root of the tree. If X is omitted or is not an integer between 1 and n, the root of the tree will be a (computed) centroid node.

The optional input matrix N has n rows and $c =$ (1, 2, or 3) columns. N ROW i contains information for node i. If N is omitted, it is assumed to be the 0×0 null matrix. If N has only 1 column, this column is assumed to contain numeric node labels. For example, if N[1] = 12, node 1 is to be labelled with the number 12. If N has 2 columns, then N ROW i is assumed to be the (x, y) coordinates of node i. If N has 3 columns, N COL 1 is assumed to contain numeric node labels and N COL 2:3 is assumed to be (x, y) coordinates.

If (x, y) coordinates are given in N, then it is not necessary for M to contain exactly $n-1$ rows, i.e. a general graph may be given when explicit (x,y) coordinates are given for its vertices.

The first 2 columns of the output matrix L can be used with the draw statement using the linetype ALTERNATE to display the underlying tree (or graph) on the screen.

If neither the edges nor the nodes are to be labelled (because M has only 2 columns and N has 0 or 2 columns), then L has only 2 columns.

If node labels, but not edge labels, are desired (because N has 1 or 3 columns, but M has only 2 columns), then L has 3 columns. In this case, L COL 3 is the list of node labels given in N, and L COL 3 is to be used as a label matrix with the LABEL-clause, i.e. we may use the statement

DRAW L COL 1:2 LINETYPE ALTERNATE LABEL L COL 3

to draw the tree (or graph) and label its nodes.

If the matrix M has 3 columns, the output matrix L will be such that edges can be labelled with the numbers contained in M col 3. This is done by "drawing" each edge with two line segments in L.

When edge labels, but not node labels, are desired (because M has 3 columns and N has 0 or 2 columns), then L has 4 columns. The last 2 columns of L can be used as the label matrix. L COL 4 specifies the point which is to be labeled with the corresponding numeric label appearing in L COL 3. A zero value in L COL 4 indicates that the corresponding label is not to be used. The DRAW statement to draw a tree (or graph) with only edge labels is:

DRAW L COL 1:2 LINETYPE ALTERNATE LABEL L COL 3:4

If both edge and node labels are desired (because M has 3 columns and N has 1 or 3 columns), then L has 3 columns and twice the usual number of rows. This doubling in the number of rows is necessary because, instead of going from one node to a connecting node, a line segment must go from a node to a midpoint and then another extra line segment must go from the midpoint to the other node. The edge labels are placed at the midpoints.

Both the node labels and the edge labels appear intermixed in L COL 3. Thus, the statement

DRAW L COL 1:2 LINETYPE ALTERNATE LABEL L COL 3

can be used to display the tree (or graph) with both node and edge labels.

Note when we are drawing both node labels and edge labels, then, in order to place point symbols, e.g. circles, at the node points along with numeric labels at nodes, but not at edge midpoints, we

must use a PTSIZE matrix clause to explicitly select the points to be labeled. For example, to create such a matrix, P, the following MLAB command is used: P = (LIST(.1,0,0,.1)^^(NROWS(L)-1).

Then to display the tree, we can type:

DRAW L COL 1:2 LINE ALTERNATE PT CIRCLE PTSIZE P LABEL L COL 3

A *dendrogram* is a binary tree where the m leaves are the m individual data points being clustered. There are exactly $m-1$ internal nodes in the tree. Each internal node x represents a cluster consisting of the leafnodes of the subtree rooted at x. A dendrogram as a whole specifies a hierarchical clustering of the data.

A dendrogram tree is represented by an array $S[1:m-1, 1:3]$ defining a binary tree with the pairs of points linked by the chosen linkage algorithm in the first two columns, and the distance between the points(clusters) in the third column. To be more specific, the first row of S contains the indices of the first two clusters(points) to be merged, and the newly-formed cluster is assigned the integer index $m+1$. (m is the total number of initial leaves). The second row of S contains the indices of the second two clusters to be merged and the newly-formed cluster is assigned the integer index $m+2$ etc. For example:

$$S \text{ COL } 1:2 = \begin{bmatrix} 2 & 3 \\ 1 & 5 \\ 4 & 6 \end{bmatrix}$$

This means that cluster 2 and cluster 3 are merged first. Then new cluster 5 merges with cluster 1 to form the new cluster 6. Finally, cluster 4 and 6 are merged to form new cluster 7.

The third column of S are the distances between the two merged clusters in column 1 and 2, except optionally in the case of Ward's linkage. For Ward's linkage, the cluster distances tend to become very large when the number of points in the cluster becomes large, (since the distances in this case are the sums of variances), hence the max/min ratio is very large and the dendrogram sometimes looks awkward. Thus, another option is provided to take the square root of the result twice; this makes the distances appear to be distributed more evenly.

In the single linkage and complete cases, the binary tree defined by S is actually a minimal spanning tree.

The function DENCURVE() takes a dendrogram tree matrix as input and generates a matrix whose rows are groups of points in the plane defining line segments which can be drawn to show the dendrogram tree on the screen.

33·3. SLINK(D)[7] single linkage dendrogram

SLINK(D) D is an $m(m-1)/2$ vector of distances for some $m \times n$ matrix whose rows are m n-dimensional points. D is generally calculated using the DISTS operator. An $(m-1) \times 3$ matrix S defining the single-linkage sequential clustering tree (or dendrogram) is returned. This matrix S can be used to produce a dendrogram graph with the aid of the DENCURVE operator. The matrix S can also be used with the COPHEN operator to compute the cophenetic correlation between the data and the clusters.

Column 1 and 2 of S contain cluster indices linked in pairs to form a binary tree. The leaf nodes are numbered from 1 to m. They are the singleton clusters from which all higher clusters are built. Each newly-formed cluster, corresponding to S ROW i, is assigned the index $m+i$, where m is the total number of initial leaves. S ROW i COL 1:2 contains the indices of the two component clusters which form cluster $m+i$. There are $m-1$ higher clusters which correspond to the interior nodes of the output clustering tree. $S[i,3]$ contains the corresponding single linkage distances between the two clusters which are merged in S ROW i.

The single-linkage distance between two clusters $A = \{a_1, a_2, \ldots, a_k\}$ and $B = \{b_1, b_2, \ldots, b_h\}$ is $\min(\text{dist}(a_i, b_j))$ for a_i in A and b_j in B, where $\text{dist}(x, y)$ is the euclidean distance between x and y.

33·4. ALINK(D)[7] average linkage dendrogram

ALINK(D) D is an $m(m-1)/2$ vector of distances for some $m \times n$ matrix whose rows are m n-dimensional points. D is generally calculated using the DISTANCES operator. An $(m-1) \times 3$ matrix S defining the average-linkage sequential clustering tree or dendrogram is returned. This matrix can be used to produce a dendrogram graph using the DENCURVE operator. The matrix S can also be used with the COPHEN operator.

Column 1 and 2 of S contain cluster indices linked in pairs to form a binary tree. The leaf nodes are numbered from 1 to m. They are the singleton clusters from which all higher clusters are built. Each newly-formed cluster, corresponding to S ROW i, is assigned the index $m+i$, where m is the total number of initial leaves. S ROW i COL 1:2 contains the indices of the component clusters which form cluster $m+i$. There are $m-1$ higher clusters which correspond to the interior nodes of the output clustering tree. $S[i,3]$ contains the corresponding average linkage distances between the two clusters which are merged in S ROW i.

The average-linkage distance between two clusters $A = \{a_1, a_2, \ldots, a_m\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ is defined as sum(dist(a_i, b_j) for a_i in A and b_j in B) / (mn). i.e. the sum of distance between every pair of points, one in each cluster (A and B), divided by the total number of such pairs, which is equal to mn, where dist(x, y) is the euclidean distance between x and y.

33·5. CLINK(D)[7] complete linkage dendrogram

CLINK(D) D is an $m(m-1)/2$ vector of distances for some $m \times n$ matrix whose rows are m n-dimensional points. D is generally calculated using the DISTS operator. An $(m-1) \times 3$ matrix S defining the complete-linkage sequential clustering binary tree or dendrogram is returned. This matrix can be used to produce a dendrogram graph using the DENCURVE operator. The matrix S can also be used with the COPHEN operator.

Column 1 and 2 of S contain cluster indices linked in pairs to form a binary tree. The leaf nodes are numbered from 1 to m. They are the singleton clusters from which all higher clusters are built. Each newly-formed cluster, corresponding to S ROW i, is assigned the index $m+i$, where m is the total number of initial leaves. S ROW i COL 1:2 contains the indices of the component clusters which form cluster $m+i$. There are $m-1$ higher clusters which correspond to the interior nodes of the output clustering tree. $S[i,3]$ contains the corresponding complete linkage distances between the two clusters which are merged in S ROW i.

The complete-linkage distance between two clusters $A = \{a_1, a_2, \ldots, a_m\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ is defined as max(dist(a_i, b_j)) for a_i in A and b_j in B where dist(x, y) is the euclidean distance between x and y.

33·6. CENTROID(D)[7] centroid linkage dendrogram

CENTROID(D) D is an $m(m-1)/2$ vector of distances for some $m \times n$ matrix whose rows are m n-dimensional points. D is generally calculated using the DISTS operator. An $(m-1) \times 3$ matrix S defining the centroid-linkage sequential clustering binarytree or dendrogram is returned. This matrix can be used to produce a dendrogram graph using the DENCURVE operator. The matrix S can also be used with the COPHEN operator.

Column 1 and 2 of S contain cluster indices linked in pairs to form a binary tree. The leaf nodes are numbered from 1 to m. They are the singleton clusters from which all higher clusters are built. Each newly-formed cluster, corresponding to S ROW i, is assigned the index $m+i$, where m is the total number of initial leaves. S ROW i COL 1:2 contains the indices of the component clusters which form cluster $m+i$. There are $m-1$ higher clusters which correspond to the interior nodes

of the output clustering tree. $S[i,3]$ contains the corresponding centroid linkage distances between the two clusters which are merged in S ROW i.

The centroid-linkage distance between two clusters $A = \{a_1, a_2, \ldots, a_m\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ is defined as $\text{dist}(\bar{a}, \bar{b})$, where $\text{dist}(x, y)$ is the euclidean distance between x and y, and $\bar{a}$ and $\bar{b}$ are the centroids of A and B respectively, i.e.

$$\underline{a} = \sum_{i=1}^{m} \frac{a_i}{m}, \text{ and } \underline{b} = \sum_{i=1}^{n} \frac{b_i}{n}$$

33·7. WARD(M [,X])[7] Ward's linkage dendrogram

WARD(M [,X]) M is an $m(m-1)/2$ vector of distances for some $m \times n$ matrix whose rows are m n-dimensional points. M is generally calculated using the DISTS operator. An $(m-1) \times 3$ matrix S defining the Ward's-linkage sequential clustering tree or dendrogram is returned. This matrix can be used to produce a dendrogram graph using the DENCURVE operator. The matrix S can also be used with the COPHEN operator.

Column 1 and 2 of S contain cluster indices linked in pairs to form a binary tree. The leaf nodes are numbered from 1 to m. They are the singleton clusters from which all higher clusters are built. Each newly-formed cluster, corresponding to S ROW i, is assigned the index $m+i$, where m is the total initial leaves. S ROW i COL 1:2 contains the indices of the component clusters which form cluster $m+i$. There are $m-1$ higher clusters which correspond to the interior nodes of the output clustering tree. $S[i,3]$ contains the corresponding Ward's linkage distances between the two clusters which are merged in S ROW i.

The Ward-linkage distance between two clusters $A = \{a_1, a_2, \ldots, a_m\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ is defined as the overall variance of A and B. i.e. sum of the squared distances of each point of A and B to the centroid of $A \cup B$, where $A \cup B$ is all the points in A and all the points in B. More precisely,

$$\text{dist}_{\text{Ward}}(A, B) = \left(\sum_{i=1}^{m} (a_i - \underline{c})^2 + \sum_{i=1}^{n} (b_i - \underline{c})^2\right)^{1/2}$$

where $\underline{c} = \frac{1}{m+n}\left(\sum_{i=1}^{m} a_i + \sum_{i=1}^{n} b_i\right)$ is the centroid of $A \cup B$.

The optional input X is the distance switch. When X is not zero, the 4th root of all the Ward distances between clusters will be returned in S COL 3. The reason you may wish to do this is because the Ward linkage distance between two clusters is the square root of the sum of the variances. The max/min ratio can be very large when the number of points is large and the dendrogram sometimes looks awkward. Taking 4th roots makes the distances appear to be distributed more evenly.

33·8. DENCURVE(M[,X])[7] dendrogram tree matrix for drawing

DENCURVE(M[,X]) M is a 3-column matrix defining a dendrogram, generally computed by the SLINK, CLINK, ALINK, CENTROID, or WARD operators. DENCURVE returns a 4-column matrix Q, which, when drawn with line-type MARKER in the DRAW statement, as: DRAW Q COL 1:2, LINE MARKER, LABEL Q COL 3:4, draws a dendrogram. Q COL 1:2 is a marker-row-matrix with groups of points to be drawn with connecting line segments. Q COL 3:4 contains labels for the data points occurring along the bottom of the tree.

X is an optional switch. If omitted or equal to zero, it means do not generate labels for internal nodes. If X is not equal to zero, it means generate labels for internal nodes.

33·9. COPHEN(M,N)[7] cophenetic correlation of a dendrogram

COPHEN(M,N) M is an $(m-1)\times 3$ matrix defining a dendrogram linkage, as produced by the MST, SLINK, ALINK, CLINK, CENTROID, or WARD operators. N is an $m(m-1)/2$ vector as computed by the DISTS operator. The (cophenetic) correlation between the distances in column 3 of the dendrogram linkage matrix M and the distances in N is returned. The cophenetic correlation between M COL 3 and N is defined as:

$$\frac{\sum_{i<j}(d_{i,j}-\bar{d})(q_{i,j}-\bar{q})}{[\sum_{i<j}(d_{i,j}-\bar{d})^2\sum_{i<j}(q_{i,j}-\bar{q})^2]^{1/2}},$$

where $d_{i,j}$ is the distance between objects i and j given in the matrix N and $q_{i,j}$ is the distance between objects i and j in M COL 3. $\bar{d}$ and $\bar{q}$ are the average of $d_{i,j}$ and $q_{i,j}$.

33·10. INCON(M,X)[7] inconsistency coefficients of a dendrogram

INCON(M,X) M is an $(m-1)\times 3$ matrix defining a binary tree. X is an integer specifying tree depth. The output Q is an $(m-1)\times 4$ matrix with $Q[i,1]$ equal to the mean of the edge-lengths adjacent to edge i in tree T, to a depth of X, $Q[i,2]$ is equal to the standard deviation of the adjacent edge lengths, $Q[i,3]$ is equal to the number of edges adjacent to edge i to a depth of X and $Q[i,4]$ is the *inconsistency coefficient* of the i-th edge length and the average edge length, i.e. $Q[i,4]=(M[i,3]-Q[i,1])/Q[i,2]$.

Example: The following dendrogram tree has its edge-lengths marked. The depth-2 average edge-length of node A is (13+15+17+ 11+9+14+19)/7 = 98/7 = 14, given in Q COL 1.

The standard deviation of the adjacent depth-2 edge lengths of node A is
$\sqrt{1^2+1^2+3^2+3^2+5^2+0^2+5^2}=\sqrt{70}=8.36$, given in Q COL 2

The number of adjacent depth-2 edges is 7.

The inconsistency coefficient of node A is (13-14)/8.36 = 0.1196.

33·11. TREEGROUP(M, N, X)[7] regroups clusters from a dendrogram tree

TREEGROUP(M, N, X) M is an $(m-1)\times 3$ matrix defining a hierachical clustering tree. The output P is an $m\times 1$ matrix with integer elements, where $P[i]$ is equal to the index number of the cluster to which point i belongs.

The second input, N, is an $(m-1)\times 4$ matrix, which is the output of the INCON operator. It gives information on the inconsistancy of cluster edges. $N[i,1]$ is equal to the average of adjacent edge lengths of edge i to the depth D which was used by INCON. $N[i,2]$ is equal to the standard deviation of the adjacent edge lengths of edge i to the depth D. $N[i,3]$ is the number of such adjacent edges and $N[i,4]$ is the inconsistent coefficient of edge i with respect to the adjacent edges. Only N COL 4 is used by TREEGROUP.

The other input, X, is the inconsistency cutoff value. It determines at what level of inconsistency the edge will be broken for inconsistency.

---

34. Survival Analysis Functions

34·1. KMSURV(M[,X])[9] Kaplan-Meier survival curve estimation

A survival function $S(t)$ is a function of time, such that $S(t)$ represents the probability that a person or thing from a specified population will "survive" to time t (or more). It is assumed that the initial

probability of survival at time $t = 0$ is 1 ($S(0) = 1$) and that, eventually, everyone dies or everything fails ($S(t) \to 0$ as $t \to \infty$). Note: $1 - S(t) = \Pr[\text{lifetime} < t]$.

KMSURV(M[,X]) computes a matrix which is the Kaplan-Meier estimate of the survival probability curve which generated the survival data in M. M is a $k \times 2$ matrix M. Each row of M records an event, with the time of the j-th event in $M[j,1]$, and the type of the j-th event in $M[j,2]$. There are two types of event: failure (death) or censored (lost to follow-up). A censoring event occurs when a subject is lost to observation. failed is denoted by 1, and censored is denoted by 0. It is assumed the first column of M is sorted in increasing order.

The optional argument X is the probability level for the confidence intervals which are generated. If X is omitted, it defaults to zero and confidence intervals are not computed.

Let $t_i = M[i,1]$. Thus, t_i is the time at which the i-th event occurs. The time values in M COL 1 are measured in any consistent unit, such as number of days. The first time-value t_1 need not be 0, but if it is not 0, then we assume that no events have occurred in the interval $[0, t_1]$.

Let d_i be the number of failure events occurring at time t_i and let n_i be the number still living before time t_i. Then the Kaplan-Meier estimate of the survival probability at t_j is given by

$$s(t_j) = \Pr[\text{the individual survives to a time} > t_j] = \prod_{i=1}^{j}(1 - \frac{d_i}{n_i}).$$

This formula is interpreted within MLAB to allow for ties between the times of censored events and/or failure events. In the event that censored events are tied with failure events, these tied events are assumed to occur in the order they appear in the data. For example, if all the failure observations for a given time appear in M before the censored observations for the same time, then they are considered to occur just before the censored observations.

The estimate of the variance of the estimated survival probability at t_j is given by Greenwood's formula:

$$\text{var}(s(t_j)) = s(t_j)^2 \sum_{i=1}^{j} \frac{d_i}{n_i(n_i - d_i)}.$$

KMSURV(M) returns a matrix S with 3 or 5 columns, depending on whether the optional argument X was provided with a non-zero value. The increasing time-values in S COL 1 correspond to the time-values of failure events. If $M[1,1] \neq 0$, an initial time of 0 will be included in S COL 1. Moreover, a final time-value $M[k,1]$ will be included in S COL 1 if the event at the time $M[k,1]$ was a censoring. The estimates of the survival probabilities at these times are in S COL 2, and the estimates of the variances of the estimates of the survival probabilities at these times are in S COL 3.

If the argument X was given as a value in (0,1], then the 100X-percentile approximate confidence intervals for the estimated survival probabilities in S COL 2 are given in S COL 4:5. In this case, the probability X that the true survival probability $S(t)$, estimated by $S[i,2]$, is in the interval $[S[i,4], S[i,5]]$.

34·2. ACTSURV(M[,X])[9] actuarial survival curve estimate

A survival function $S(t)$ is a function of time, such that $S(t)$ represents the probability that a person or thing from a specified population will "survive" to time t (or more). It is assumed that the initial probability of survival at time $t = 0$ is 1 ($S(0) = 1$) and that, eventually, everyone dies or everything fails ($S(t) \to 0$ as $t \to \infty$). Note: $1 - S(t) = \Pr[\text{lifetime} < t]$.

ACTSURV(M[,X]) computes a matrix which is the actuarial estimate of the survival curve which generated the data in M. M must have 4 columns, and k rows, corresponding to the k left-endpoints which determine k adjacent intervals of observation which partition the total observation period $[M[1,1], \infty]$.

The optional argument X is the probability level for the confidence intervals which are generated. If X is omitted, it defaults to zero and confidence intervals are not computed.

$M[j,1]$ contains the starting time of interval j.
$M[j,2]$ contains the number of subjects alive at the start of interval j.
$M[j,3]$ contains the number of subjects that died (failed) during interval j.
$M[j,4]$ holds the number of subjects withdrawn or lost to follow-up (censored) during interval j.

It is assumed that M COL 1 is sorted in increasing order.

Note that, except for the entry in M[1,2], all the numbers in M COL 2 may be deduced from the data in M COL 3:4.

Let $t_i = M[i,1]$. The range of time from t_1 to ∞ is partitioned into k half-closed intervals $(t_1, t_2], \ldots, (t_k - 1, t_k], (t_k, \infty]$ where t_i is the starting time for interval i. The time values in M COL 1 are measured in any consistent unit, such as number of months. The starting time, t_1, of the first interval need not be 0, but if $t_1 \neq 0$, then the interval $[0, t_1]$ is implicitly appended to the data and no failure or censoring events are assumed to have occurred in the interval $[0, t_1]$.

Let n_i be the number of subjects alive at the start of the i-th interval, so $n_i = M[i,2]$. Let d_i be the number of subjects that die in the i-th interval, so $d_i = M[i,3]$. Let w_i be the number of subjects lost to follow-up or withdrawn in the ith interval, so $w_i = M[i,4]$.

The actuarial estimate of the survival probability at the start of the j-th time interval is given by $s(t_j) = \Pr(\text{an individual survives to a time} \geq t_j)$

$$= \prod_{i=1}^{j} (1 - \frac{d_i}{n_i - .5w_i}).$$

Note that $(1 - d_i/n_i)$ would be the probability of surviving the i-th interval if one survived to time t_i and there were no censored events between t_i and $t_i + 1$. Those who were censored are assumed to have been, on average, at risk for half the interval.

The variance of $s(t_j)$ is estimated by Greenwood's formula:

$$var(s(t_j)) = s(t_j)^2 \sum_{i=1}^{j} \frac{d_i}{(n_i - .5w_i)(n_i - .5w_i - d_i)}$$

ACTSURV(M) returns a matrix S with either 3 or 5 columns, depending on whether the optional argument X was provided.

S COL 1 contains the starting times of each interval given in M COL 1. If $M[1,1] \neq 0$, then time 0 is included in S COL 1.
S COL 2 contains the estimated survival probabilities $s(t)$ for each interval.
S COL 3 contains the estimated variances of the estimated survival probabilities in S COL 2.

If the argument X was given as a value in (0,1], then the 100X-percentile approximate confidence intervals for the estimated survival probabilities in S COL 2 are given in S COL 4:5. In this case, the probability that the true survival probability $S(t)$, estimated by $S[i,2]$, is in the interval $[S[i,4], S[i,5]]$ is X.

34·3. MHT(M[,X[,Y]])[9] Mantel-Haenszel-Armitage-Cochran test

MHT(M,[,X,[Y]]) performs either the basic Mantel-Haenszel-Armitage-Cochran test, or a weighted-variant of it such as the Tarone-Ware version or the Gehan version, on a set of k $2 \times r$ contingency tables. M is a $[1:2k, 1:r]$ matrix containing k $2 \times r$ contingency tables in its k successssive pairs of rows.

The optional input scalar argument X is the weight exponent. In particular, if X = 0, then the Mantel-Haenszel-Armitage-Cochran test is performed; if X= .5, then the Tarone-Ware form of the test is performed; and if X = 1, then the Gehan form of the test is performed. If X is not given, it defaults to 0.

When M has two columns, i.e. when $r = 2$, the optional input argument, Y, specifies whether a continuity correction is to be done. If $Y \neq 0$, a continuity correction is included in computing the test statistic value. If Y is omitted, it defaults to 0, and no continuity correction is done.

The k contingency tables are stored in successive pairs of rows in the input array $M[1:2k, 1:r]$. Each table i has the following form:

treatment:	1	2	...	r	
outcome A	$a_{i,1}$	$a_{i,2}$	...	$a_{i,r}$	$m1_i$
outcome B	$b_{i,1}$	$b_{i,2}$	...	$b_{i,r}$	$m2_i$
	$n1_i$	$n2_i$	...	nr_i	N_i

Table i may be interpreted as representing $n1_i$ subjects treated with treatment 1, $n2_i$ subjects treated with treatment 2, ..., and nr_i subjects treated with treatment r. Of the $N_i = n1_i + n2_i + \ldots + nr_i$ subjects, $m1_i$ result in outcome A and $m2_i$ result in outcome B.

The contingency tables $1, 2, \ldots, k$ may distinguish studies made under k different conditions called *strata*, such as study location or age of patients.

Let P_{ij} denote the probability of outcome A for subjects in the i-th stratum undergoing treatment j.

The null hypothesis, H_0, is that, in each stratum, there is no difference between the r treatments so that $P_{ij} = P_{it}$ for $j = 1, 2, \ldots, r$, $t = 1, 2, \ldots, r$ and $i = 1, \ldots, k$. The alternate hypothesis, H_1, is that H_0 is false; i.e. that there is a difference between the two treatments in at least one stratum (i.e. $P_{ij} \neq P_{it}$ for at least one stratum, i).

The test statistic sample value c is computed for the k input $2 \times r$ contingency tables as $c = (A - E)'V^{-1}(A - E)$. c is a sample of an approximately chi-square distributed random variable S with $r - 1$ degrees of freedom, given the null hypothesis H_0 that all the treatments have the same effect. A is a $(r-1)$-vector with $A_i = \sum_{j=1}^{k} w_i a_{j,i}$, for $i = 1, \ldots, r-1$. It is the observed number of subjects under treatment i with outcome A in stratum j, weighted with the weight w_i, and summed over all strata. E is a $(r-1)$-vector with $E_i = \sum_{j=1}^{k} w_i m1_j n1_j / N_i$, for $i = 1, \ldots, r$. It is the expected value, assuming H_0, for the number of subjects under treatment i with outcome A in stratum j, weighted with the weight w_i, and summed over all strata.

V^{-1} is a $(r-1) \times (r-1)$ matrix which is the inverse of the covariance matrix V. The (s,t) element of the covariance matrix V is given by

$$V_{s,t} = \sum_{j=1}^{k} \frac{\delta_{s,t} ns_j m1_j m2_j}{(N_i - 1)N_i} - \frac{w_i^2 m1_j m2_j ns_j nt_j}{(N_i - 1)N_i^2}$$

where $\delta_{s,t}$ is 1 if $s = t$, and 0 otherwise. $V_{s,t}$ is the sample covariance of the treatment s and t pairs $(a_{i,s}, a_{i,t})$ with outcome A, assuming H_0.

Note that A and E have $(r-1)$ components, not r components, and that V is a $(r-1) \times (r-1)$ dimensional array, not a $r \times r$ dimensional array. One of the r treatments (the last one) is eliminated from the calculation to avoid singularities when solving for the value of the test statistic.

When $r = 2$ and the continuity correction switch argument Y is non-zero, then the test statistic sample value c is $[(\text{abs}[A[1] - E[1]) - .5)/sqrt(V[1,1])]2$.

In all of the preceding definitions, the weights $w_1, w_2, \ldots, w_k$ are determined by the choice of the weight exponent argument X. If X = 0, then $w_i = 1$ and the Mantel-Haenszel-Armitage-Cochran test results; if X = .5, then $w_i = N_i^{1/2}$ and the Tarone-Ware test results; and if X = 1, then $w_i = N_i$ and the Gehan test results. In general the weight $w_i = N_i^X$ is used.

Under the null hypothesis, the test statistic for the k $2 \times r$ contingency tables, of which c is a sample, is approximately distributed as a random variable S that is chi-square distributed with $r - 1$ degrees of freedom. We compute $\Pr[S > c] = p$. If p is small, then the sample value c is unlikely to arise when H_0 is true. This is a 1-tailed test. A 3-row 1-column array P is returned. With $P[1] = c$, $P[2] = r - 1$, $P[3] = p$.

34·4. THOMAST(M)[9] Thomas' test

THOMAST(M) performs Thomas' exact test on a set of k 2×2 contingency tables. M is a $[1 : k, 1 : 4]$ matrix containing k 2×2 contingency tables in successsive rows. The i-th row of m holds count data from the i-th contingency table as follows:

	Success	Failure	
group 1	$a_i = M_{i,1}$	$b_i = M_{i,2}$	$n1_i$
group 2	$c_i = M_{i,3}$	$d_j = M_{i,4}$	$n2_i$
	$m1_i$	$m2_i$	N_i

where $i = 1, \ldots, k$. The data values a_i, b_i, c_i and d_i are taken as samples of corresponding random variables A_i, B_i, C_i, and D_i, respectively.

The odds ratio for the i-th table is $\psi_i = (p1_i q2_i)/(q1_i p2_i)$, where $p1_i$ is the probability of success in group 1 and $q1_i = 1 - p1_i$ is the probability of failure in group 1 for the i-th table. $p2_i$ is the probability of success in group 2 and $q2_i = 1 - p2_i$ is the probability of failure in group 2 for the i-th table.

Also note that the unconditional maximum likelihood estimator of the odds ratio for each individual table is $(A_i D_i)/(B_i C_i)$.

Thomas' exact test is a generalization of Fisher's exact test for testing that the success probabilities in 2 independent groups are equal ($H_0 : p_1 = p_2$) in 1 table (stratum) to doing the test in k independent tables (k strata).

Thomas' test is an exact test for testing the null hypothesis H_0: that in every stratum, the success rate for each of the two groups are the same. In other words, the null hypothesis is that the individual odds ratios $\psi_1, \ldots, y_k$ have the common value 1.

The alternative hypothesis is that one or more of the ψ_i-values are different from 1. Often the 2 groups are referred to as treatments, since medical settings are a common genesis of this type of data.

Given H_0, $E(A_i) = n1_i m1_i / N_i$. Also, given H_0, the distribution of the random variable A_i, conditional on the 4 marginal row sums and column sums for table i, is hypergeometric. Since $A_1, \ldots, A_k$ are independent, the joint density function of $(A_1, \ldots, A_k)$ is the product of the density functions of $A_1, \ldots, A_k$. Call this joint density function $f(z_1, \ldots, z_k)$. Then,

$$f(z_1, \ldots, z_k) = \prod_{i=1}^{k} \Pr[A_i = z_i | \psi_i = 1; n1_i; m1_i; m2_i]$$

where $\Pr[A_i = z_i | \psi_i = 1; n1_i; m1_i; n2_i; m2_i] = \sum_v \binom{n1_i}{v} \binom{n2_i}{m1_i - v} / \binom{n1_i + n2_i}{m1_i}$.

In table i, with its fixed marginals, the upper left corner entry a_i is constrained to obey $\max(n1_i - m2_i, 0) \le a_i \le \min(n1_i, m1_i)$ (*) due to the non-negativity of the table entries.

Let $s = a_1 + \ldots + a_k$. We compute the two probabilities p_1 and p_2. We define $p_1 = \Pr((A_1, \ldots, A_k) \le (a_1, \ldots, a_k)|H_0) = \sum f(z_1, \ldots, z_k)$. The sum is taken over all possible vectors $(z_1, \ldots, z_k)$ obeying (*) and such that $z_1 + \ldots + z_k \le s$. The value p_2 is similarly defined as $p_2 = \Pr((A_1, \ldots, A_k) \ge (a_1, \ldots, a_k)|H_0) = \sum f(z_1, \ldots, z_k)$. The sum is taken over all possible vectors $(z_1, \ldots, z_k)$ obeying (*) and such that $z_1 + \ldots + z_k \ge s$.

Note $p_1 + p_2 - 1 = \Pr((A_1, \ldots, A_k) = (a_1, \ldots, a_k)|H_0)$ and $\Pr((C_1, \ldots, C_k) \le (a_1, \ldots, a_k)|H_0) = p_2$. This is because $A_i + C_i$ is always a constant in all the configurations in the sum which computes this probability.

Small values for p_1 or p_2 indicate that it is unlikely that the null hypothesis is correct, given the input k 2×2 tables. Tables as extreme as the input tables will occur with probability $2\min(p_1, p_2)$, given H_0.

THOMAST returns the values p_1 and p_2 computed as above as the first and second rows of a 2 row result vector.

References:

Armitage, P. Statistical Methods in Medical Research. John Wiley &Sons, New York, 1971, pp. 135-138.

Fisher, R.A. Statistical Methods for Research Workers. 14th edition. Hafner Publishing Company, Darien, Connecticut, 1970, pp. 96-97.

Thomas, D.: "Exact and Asymptotic Methods for the Combination of 2 X 2 Tables", Computers and Biomedical Research, 8, 423-446,1975

34·5. SURV2T(M1,M2[,X[,Y]])[9] Mantel-Haenszel test on 2 group survival data

SURV2T(M1,M2[,X[,Y]]) performs either the basic Mantel-Haenszel-Armitage-Cochran test, or a weighted variant of it such as the Tarone-Ware version, or the Gehan version, based on two arrays of survival data stored in $M1[1:m, 1:2]$ and $M2[1:n, 1:2]$.

The two input arrays M1 and M2 contain survival data stored so that, in both M1 and M2, the first column consists of event times and the second column consists of the result code corresponding to the event time in column 1. The result codes in column 2 are 1 for FAILED and 0 for CENSORED. M1 corresponds to a group of subjects treated with treatment 1; M2 corresponds to a group of subjects treated with treatment 2. SURV2T tests if treatment 1 is significantly different from treatment 2, i.e. if the survival curve of the tratment 1 group is significantly different from the survival curve for treatment 2 group.

The optional input scalar argument X is the weight exponent. In particular, if X = 0, then the Mantel-Haenszel-Armitage-Cochran test is performed; if X= .5, then the Tarone-Ware form of the test is performed; and if X = 1, then the Gehan form of the test is performed. If X is not given, it defaults to 0.

The second optional argument Y controls the inclusion of a continuity correction in the test statistic calculation. If $Y \neq 0$, the continuity correction is performed. If Y = 0, the continuity correction is not performed. If Y is not given, it defaults to 0.

SURV2T transforms the two arrays M1 and M2 for treatments 1 and 2 into a sequence of k 2×2 contingency tables, where k is the number of event times at which at least one FAILED event occurs in either M1 or M2. For each such event time, a 2×2 contingency table is formed. The i-th 2×2 contingency table can be diagramed as follows:

treatment:	1	2
FAILED	$a_{i,1}$	$a_{i,2}$
NOT FAILED	$b_{i,1}$	$b_{i,2}$

Such a table indicates that at the i-th event time t_i, the $a_{i,j} + b_{i,j}$ subjects treated with treatment j whose associated event times are $\geq t_i$, result in $a_{i,j}$ FAILED events and $b_{i,j}$ not-FAILED events.

This constructed sequence of 2×2 contingency tables is used as input to the Mantel-Haenszel test (See MHT). The form of the Mantel-Haenszel test specified by the test selection switch X is then performed. SURV2T returns a 3-row array Q with the chi-squared test statistic sample value v in row 1, the associated degrees of freedom (= 1) in row 2, and the probability that, given the null hypothesis, the test statistic takes a value greater than v in row 3.

Reference: Miller, R.G. Survival Analysis. John Wiley & Sons, New York, 1981

34·6. SURVT(M[,X])[9] Mantel-Haenszel test on multiple group survival data

SURVT(M[,X]) performs either the basic Mantel-Haenszel-Armitage-Cochran test, or a weighted variant of it, such as the Tarone-Ware version, or the Gehan version, based on the survival data stored in $M[1 : m, 1 : 3]$.

The input array $M[1 : m, 1 : 3]$ contains survival data from r different groups. The first column of M consists of event times, the second column of M consists of the FAILED-CENSORED codes for the corresponding event times in column 1. Each code value is either 0 (for CENSORED) or not 0 (for FAILED). The third column of M consists of group numbers that tells for which group the corresponding specified event in columns 1 and 2 happens. Let the group numbers in column 3 be $1, 2, \ldots, r$; corresponding to r groups.

The optional input scalar argument X is the weight exponent. In particular, if X = 0, then the Mantel-Haenszel-Armitage-Cochran test is performed; if X = .5, then the Tarone-Ware form of the test is performed; and if X = 1, then the Gehan form of the test is performed. If X is not given, it defaults to 0.

SURVT transforms M into a sequence of k $2 \times r$ contingency tables, where r is the number of groups in the given survival data, and k is the number of event times at which at least one FAILED event occurs. For each such event time, a $2 \times r$ contingency table is formed. The i-th $2 \times r$ contingency table can be diagrammed as follows:

treatment:	1	2	...	r
FAILED	$a_{i,1}$	$a_{i,2}$	...	$a_{i,r}$
NOT FAILED	$b_{i,1}$	$b_{i,2}$	...	$b_{i,r}$

Such a table indicates that at the i-th event time t_i, the $a_{i,j} + b_{i,j}$ subjects treated with treatment j whose associated event times are $\neq t_i$,result in $a_{i,j}$ FAILED events and $b_{i,j}$ not-FAILED events.

The form of the Mantel-Haenszel test (See MHT) specified by the test selection switch X is then performed on the sequence of $2 \times r$ contingency tables based on the input M. SURVT returns an array with 3 rows which includes the sample value of the chi-squared test statistic v in row 1, the associated degrees of freedom ($= r - 1$) in row 2, and the probability that the test statistic takes a value greater than v in row 3.

Reference: Miller, R.G. Survival Analysis. John Wiley & Sons, New York, 1981

34·7. HBPOWER(M1,M2[,P[,Q[,R[,S[,T[,U]]]]]])[9] Mantel-Haenszel test power simulation

HBPOWER() computes a scalar which is the power of a specified version of the Mantel-Haenszel test using the simulation method described by Halpern and Brown[1]. This entails generating data for a model clinical study which is designed to distinguish two types of treatment response at a prespecified significance level.

M1 and M2 are arrays holding points of the two postulated survival curves for treatments 1 and 2. Column 1 of M1 and M2 are positive scalars representing times and column 2 of M1 and M2 are probabilities in [0,1] for a patient to survive to at least the corresponding time in column 1.

The remaining arguments for HBPOWER are optional. If any optional argument is specified, all preceding optional arguments must be supplied.

P is a scalar in [0,1] specifying the significance level for the test. The default value for P is .05.

Q is a positive scalar taking on integer values specifying the accrual rate, which is the number of patients admitted to the study at even intervals during one time unit. The default value for Q is 30.

R is a positive scalar specifying the accrual time, which means the number of time units during which patients are admitted. The default value for R is 1.

S is a positive scalar specifying the follow-up time, which means the number of years of follow-up observation. The default value of S is 1.

T is an positive scalar specifying the number of simulation-based trials, which is the number of times the Monte Carlo simulation will be performed. The default value of T is 20.

U is a scalar specifying the exponent weight for the test. U = 0 corresponds to the Mantel-Haenszel-Armitage test. U = 0.5 corresponds to the Taron-Ware test. U = 1 corresponds to the Gehan test. However, other values for the weight exponents are also admissible. The default value for U is 0.

HBPOWER returns as its output value a scalar which is the estimated power of the selected test. The power of the test is the probability that the selected test with the specified significance level correctly detects that the two sample-based survival curves are different. This test is of use when designing clinical studies because one can observe the dependence of power on significance level, sample size, expected survival curve difference and several other clinical trial parameters.

---

35. Statistical Power and Sample Size Calculations

35·1. STUTFA(B,N,T,R)[9] rejection error probability of the t-test for equal means

STUTFA(B,N,T,R) computes a scalar A which is the rejection error (type I error) probability A for the two sample Student's t-test implemented by TST. The arguments B, N, T, and R must all be scalars. The computed value A corresponds to the arguments B, N, T, and R. The value N is the control group size and $N \cdot T$ is the alternate treatment group size. N must be a positive value (ordinarily an integer). T is the ratio of the size of the control group to the size of the treated group; it must be positive. R is the ratio of the difference of the two group means to the common standard deviation postulated under the alternate hypothesis. R must be positive. B is the acceptance error (type II error) probability. B must lie in [0,.5].

35·2. STUTFB(A,N,T,R)[9] acceptance error probability of t-test for equal means

STUTFB(A,N,T,R) computes a scalar B which is the acceptance error (type II error) probability B for the two sample Student's t-test implemented by TST. The arguements A, N, T, and R must all be scalars. The computed value B corresponds to the arguments A, N, T, and R. The value N is the control group size, and $N \cdot T$ is the alternate treatment group size. N must be a positive value (ordinarily an integer). T is the ratio of the size of the control group to the size of the treated group;

it must be positive. R is the ratio of the difference of the two group means to the common standard deviation postulated under the alternate hypothesis. R must be positive. A is the rejection error (type I error) probability. A must lie in [0,.5].

Note: 1-STUTFB(A,N,T,R) is the power of the two-sample, equal variance t-test specified by the values A, N, T, and R.

35·3. STUTFN(A,B,T,R)[9] sample size of the t-test for equal means

STUTFN(A,B,T,R) computes a scalar N which is the control group sample size N for the two sample Student's t-test implemented by TST. The arguments A, B, T, and R must all be scalars. The computed value N corresponds to the arguments A, B, T, and R. A is the rejection error (type I error) probability. A must lie in [0,1]. B is the acceptance error (type II error) probability. A and B must lie in [0,.5]. T is the ratio of the size of the control group to the size of the alternate treatment group. T must be positive. R is the ratio of the difference of the two group means to the common standard deviation postulated under the alternate hypothesis. R must be positive.

35·4. STUTFT(A,B,N,R)[9] sample size ratio of the t-test for equal means

STUTFT(A,B,N,R) computes a scalar T which is the ratio of the size of the control group to the size of the alternate treatment group for the two sample Student's t-test implemented by TST. The arguments A, B, N, and R must all be scalars. The computed value T corresponds to the arguments A, B, N, and R. A is the rejection (type I error) probability. A and B must lie in $[0, .5]$. B is the acceptance error (type II error) probability. N is the size of the control group sample. N must be a positive value (ordinarily an integer). R is the ratio of the difference of the two group means to the common standard deviation postulated under the alternate hypothesis. R must be positive.

35·5. QFA(B,N,T,E)[9] rejection error probability of the variance-ratio F-test

QFA(B,N,T,E) computes a scalar A which is the rejection error (type I error) probability A for the variance-ratio F-test implemented by QFT. The computed value A corresponds to the scalar arguments B, N, T, and E. N is the numerator group sample size. N must be a non-negative integer. $T \cdot N$ is the denominator group sample size. T must be positive. B is the acceptance error (type II error) probability. B must lie in [0,.5]. E is the variance ratio value postulated under the alternate hypothesis (in opposition to the null hypothesis variance ratio value 1). E must be positive.

35·6. QFB(A,N,T,E)[9] variance-ratio F-test acceptance error probability

QFB(A,N,T,E) computes a scalar B which is the acceptance error (type II error) probability B for the variance-ratio F-test implemented by QFT. The computed value B corresponds to the scalar arguments A, N, T, and E. N is the size of the numerator group sample. N must be a non-negative integer. $T \cdot N$ is the size of the denominator group sample. T must be positive. A is the rejection error (type I error) probability. A must lie in [0,1]. E is the variance ratio value postulated under the alternate hypothesis (in opposition to the null hypothesis variance ratio value 1).

Note: 1-QFB(A,N,T,E) is the power of the variance ratio F-test specified by the values A, N, T, and E.

35·7. QFN(A,B,T,E)[9] variance-ratio F-test sample size

QFN(A,B,T,E) computes a scalar N which is the numerator group sample size N for the variance-ratio F-test implemented by QFT. The computed value N corresponds to the scalar arguments A, B, T, and E. A is the rejection error (type I error) probability. A must lie in [0,1]. B is the acceptance

error (type II error probability). B must lie in [0,.5]. T is the ratio of the numerator group size to the denominator group sample size, (i.e. the denominator group sample size is $T \cdot N$). T must be positive. E is the variance ratio value postulated under the alternate hypothesis (in opposition to the null hypothesis variance ratio value 1). E must be positive.

35·8. QFE(A,B,N,T)[9] variance-ratio F-test alternate hypothesis variance ratio

QFE(A,B,N,T) computes a scalar which is the alternate hypothesis variance ratio value > 1 that can be detected with rejection error probability A and acceptance error probability B by using the variance-ratio F-test implemented by QFT. The computed value E corresponds the scalar arguments A, B, N, and T. A is the rejection error (type I error) probability. A must lie in [0,1]. B is the acceptance error (type II error) probability. B must lie in [0,.5]. N is the numerator group sample size. N must be a positive integer. $T \cdot N$ is the denominator group sample size. T must be positive.

35·9. CHISQFA(B,N,D)[9] chi square test rejection error probability

The input scalar B is the probability of an acceptance (type II) error. The scalar N is the degrees of freedom of the chi-squared value. The scalar D is the RMS-average bucket deviation, corresponding to the non-centrality parameter value ND^2.

In general, a chi-squared test involves the construction of a chi-squared statistic value constructed as the sum of N squared approximately-normal mean-0, variance-1 sample values. N is the number of "buckets" used to form the sample-based N approximately-normal values; N corresponds to the degrees of freedom associated with the chi-squared value. The total sample size (the number of individual samples) involved in the chi-squared test should be at least 5N, with at least 5 sample-values falling in each bucket.

Under the null hypothesis H_0, each of the N buckets gives rise to an approximately-Normal mean-0,variance-1 sample value. Under the alternate hypothesis H_1, we assume that each of the N buckets gives rise to an approximately-Normal mean-D, variance-1 sample value, or more correctly, that the non-central chi-squared sample-value that arises under the alternate hypothesis has the expected-value $N+ND^2$ rather than N. Thus D is the RMS-average deviation amount from the approximately-Normal mean-0, variance-1 sample value that would arise in each bucket under the null hypothesis when the alternate hypothesis holds instead.

In the Poisson case, where the i-th normal sample term in our chi-squared test-statistic is of the form $(kp_i - y_i)2]/(kp_i)$, the alternate hypothesis H_1 says that y_i differs from its null-hypothesis expected value kp_i by approximately the amount $D(kp_i)^{1/2}$.

The statistic that arises under the null hypothesis H_0 is a central chi-squared random variable with N degrees of freedom. The statistic that arises under the alternate hypothesis H_1 is a non-central chi-squared random variable with N degrees of freedom and the non-centrality parameter ND^2.

CHISQFA(B,N,D) computes the rejection (type I) error probability A for a chi-squared test with N degrees of freedom (N buckets), acceptance (type II) error probability B, and the alternate hypothesis RMS-average bucket deviation D.

We have B = CHISQF(CHISQI(1-A,N,0),N,$N \cdot D^2$), so CHISQI(B,N,$N \cdot D^2$) = CHISQI(1-A,N,0) and so CHISQF(CHISQI(B,N,$N \cdot D^2$),N,0) = 1-A.

35·10. CHISQFB(A, N, D)[9] chi square test acceptance error probability

The scalar A is the probability of a rejection (type I) error. The scalar N is the degrees of freedom of the chi-squared value. The scalar D is the RMS-average bucket deviation, corresponding to the non-centrality parameter value ND^2.

CHISQFB(A,N,D) computes the probability of a rejection (type II) error, B, for a chi-squared test with N buckets, acceptance error probability A and alternate-hypothesis RMS-average bucket deviation D. The quantity 1-B is called the power of the chi-squared test associated with A,N, and D.

The central relationship between A,B,N,and D is:

B = CHISQF(CHISQI(1-A,N,0),N,$N \cdot D^2$).

35·11. CHISQFN(A,B,D)[9] chi square test degrees of freedom

The scalar B is the probability of an acceptance (type II) error. The scalar A is the probability of a rejection (type I) error. The scalar D is the RMS-average bucket deviation, corresponding to the non-centrality parameter value ND^2.

CHISQFN(A,B,D) computes the number of buckets, (i.e. the degrees of freedom) N, for a chi-squared test with rejection (type I) error probability A, acceptance (type II) error probability B, and the alternate hypothesis RMS-average bucket deviation D.

The value N corresponds to the number of "buckets" used in forming the chi-squared test statistic value. The number of actual individuals to be counted in each bucket should be at least 5 as a rule of thumb, so the total sample size should be at least 5N.

35·12. CHISQFD(A, B, N)[9] chi square test rms average bucket deviation

The scalar B is the probability of an acceptance (type II) error. The scalar A is the probability of a rejection (type I) error. The scalar N is the degrees of freedom of the chi-squared value.

CHISQFD(A,B,N) computes the alternate-hypothesis RMS-average bucket deviation, D, which corresponds to the non-centrality parameter value ND^2, for a chi-squared test with rejection (type I) error probability A, acceptance (type II) error probability B, and N degrees of freedom (N buckets).

11. Expressions

EXAMPLES:

```
A[I+1,B[3]]
(SORT(M,2) COL 3:6) [1,1]
M ROW (1,1,5,3) COL (N')
13.4-(M + F(X/Y,0))
```

Scalar or matrix values appear throughout various MLAB statements. Such values may be given by simple scalar or matrix names, or by expressions which evaluate to the required values.

Expressions in MLAB are much the same as in Fortran. Parentheses may be used freely to control the order in which operations are performed.

All arithmetic in MLAB is done with double precision (8-byte) floating point values. This allows exact integers to be handled up to magnitude $2^{52} - 1$. When underflows occur, zero is returned. When overflows or zero-divides occur, the value of the greatest magnitude with the correct sign (MAXPOS or MAXNEG) is returned, when this is possible. When the MLAB control variable ROUNDSW is zero, exception handling is suppressed, and "round towards zero" is used, so that "infinities" are still not generated.

The precedence between pairs of operators is defined and used when parentheses are absent. These precedence relationships are somewhat elaborate. The pre-cedence relation is generally non-commutative so that, for example, B&′ F ON M = B&′ (F ON M), while F ON M&′ B = F ON (M&′ B). The intent is to have the "expected and reasonable thing" happen most of the time. Parentheses cost nothing, however, and should be used freely to avoid mishaps.

Almost all operators can be used together in an expression. In particular, expressions such as (M′ &′ 1:NCOLS(M)) ROW 2:N:2 are perfectly legal. Most operators in MLAB are left-associative. Of course some operators, such as ON, cannot meaningfully occur adjacent to each other because of domain and range conflicts.

The operator precedence table appears below. The table is to be interpreted as follows. In the expression Aop_1Bop_2C, the table entry for (op_1, op_2) is a 1 if Aop_1Bop_2C means $(Aop_1B)op_2C$ and is a 2 if Aop_1Bop_2C means $Aop_1(Bop_2C)$. If the table entry is missing, it means that the corresponding pair of operators may not be juxtaposed in an expression in a meaningful way.

	^	unary -	*/ *’ /’	+ -	≤ ¡ = ¿ ≥	^^’ ^^	&’	’	ROW COL	AND	OR	NOT	:	!	#	_
(caret)^	1	2	1	1	1	1	1	2	2	1	1	1	1	1	1	1
unary -	2	1	1	1	1	1	1	2	2	1	1	1	1	1	1	1
/ *’ /’	2	2	1	1	1	1	1	2	2	1	1	1	1	1	1	2
+ -	2	2		1	1	1	1	2	2	1	1	1	1	1	1	2
<≤=≥	2	2	2	2	2	2	2	2	2	1	1	2	2	2		2
^^ ^^’	2	2	1	1	1	1	1	1	2	1	1	2			1	1
& &’	2	2	2	2	1	1	1	1	2	2	2	2	2		1	2
(prime) ’	1	1	1	1	1	1	1	1	1	1	1				1	
ROW COL	1	1	1	1	1	1	1	1	1	1	1	1	2		1	1
AND	2	2	2	2	2	2	1	1	2	2	1	2	1	1	2	2
OR	2	2	2	2	2	2	1	1	2	2	1	2	2	2	1	2
NOT	2	2	2	2	2	2	1	1	2	1	1	1	1		1	2
(colon) :	1	1	1	1	2	1	2	1	1	1	1	1	2		1	2
(exclamation)!	1	1	1	1	2	1	2	1	1	1	1	1	2	2	1	2
#	2	2	1	1	1	1	1	2	2	1	1	1	1	1	1	2
(assignment)_	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
ON	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

Sometimes scalar values are required in MLAB statements and other times matrices are needed to make sense. This is an issue of the correct type of value being specified. In some cases MLAB will automatically convert a matrix value to a scalar value or conversely in order to satisfy type requirements. The possibility of such type conversions is indicated in the descriptions of the various operations and statements.

A particular kind of expression, which may occur as a component of a larger expression, is a function value. This is the result of evaluating a function at certain argument values. Notationally, a function value is designated as F(X1,X2,...) where F is the function name and X1, X2,... are actual argument expressions of the appropriate type. A function may have an empty argument list, denoted by parentheses enclosing nothing ().

When a function value is computed, the actual argument values are used in place of the corresponding formal arguments used in defining the function; and the function body is evaluated following the usual rules for expression evaluation, except for formal arguments being assigned a value in an embedded assignment. The number of actual arguments must be the same as the number of formal arguments. Some builtin functions have both mandatory arguments and optional arguments. In this case, the number of actual arguments must be at least as many the number of mandatory arguments and no more than the total number of mandatory plus optional arguments.

Free variables in function bodies are globally free. That is all references to a free variable X in any function body, no matter how the evaluation process is nested when it is referenced, access the same global value. Of course the notion of a free variable is local to each function body. Thus, the value of X may be the value of the free variable X in one case, while it is the value of some corresponding actual argument in another case. The number and types of actual arguments and free variables in builtin functions are checked when they are used. User functions are checked for the correct number of arguments, but not for the correct type.

12. Statements

One uses MLAB by typing statements (commands). When the MLAB program is started, it indicates that it is ready to accept a statement by placing an asterisk (∗), which is the MLAB prompt symbol, on the display screen at the beginning of the next blank line. The prompt symbol indicates that MLAB is ready to accept a statement. When the statement (or statements, separated from one another by a semicolon (;)) is terminated (by pressing the *ENTER* key or, on some keyboards, the *return* key), MLAB will begin processing the statements at once. Most statements are processed completely without additional information from the user; however, some statements may prompt the user for further information. When the processing of the input has been completed, MLAB places another asterisk (∗) prompt character on the display screen to indicate that it is ready to accept a new statement.

The user may leave MLAB and return to the operating system by typing EXIT. Under DOS on a PC, the user may interrupt the execution of a statement by typing *CTRL-BREAK* (i.e. type the *CONTROL* key and the *BREAK* key simultaneously. Following an interrupted statement, MLAB returns to top level, produces a prompt character, and is ready to receive another command. This action may leave MLAB in an error state, however, and, to be safe, you should type EXIT and then restart MLAB. It may be possible to save some existing user objects before exiting, using the SAVE statement.

Some statements may be longer than the width of the display screen. To continue a statement when the end of the screen is reached, just continue typing and the statement will wrap around to the the next line on the screen. If desired, however, a line may be broken by typing a back-slash character at any point in the line. The statement may then be continued on the next line. In this manner, any number of lines may be used for a single statement. However, when the *delete* key is used to edit a command in progress, it will not back up over the back-slash character. Some compound statements, for example the FOR statement, may be placed on several lines without use of the back-slash by use of the *ENTER* key to break to a new line at the end of an individual component statement. However, most statements are effectively terminated by this process.

Several statements may be placed on the same line by typing a semicolon (;) after each statement except the last, which does not require a semicolon. Each completed line of command input must be terminated by pressing the *ENTER* key.

You may enter statements in any mixture of upper and lower-case. On DOS or Windows PC's, MLAB will interpret lower-case characters as upper case, and all text written to the display will be in upper case. The only exception to this is that strings of text enclosed in quotes will be accepted without any case-conversion imposed. On Unix and Macintosh machines, either all text entered will be converted to upper-case (when the control variable *casesw* is 0), or alternately, certain common words and names will be treated as though they were given in upper case, but all other text entered will be taken as given (when *casesw* is 1.) In all cases, quoted strings are taken as they are given; no case conversion is performed.

One special statement is a quoted string, e.g. "THIS IS A COMMENT". Such a string statement is ignored by MLAB and another ∗ prompt occurs. Such strings can be typed as comments. Comments are particularly useful in documenting a do-file, and also the MLAB log file, which is named MLAB.LOG. After an MLAB session which is terminated by an exit statement, the file MLAB.LOG will be found in the directory from which MLAB was run. Any previously existing file named MLAB.LOG will be named to MLABn.LOG, where n is 0, 1, or 2. Any initially-existing file named MLAB2.LOG will be deleted and supplanted.

On DOS or WINDOWS PC's, you may scroll back in the MLAB dialog by typing the up-arrow (↑), down-arrow (↓), page-up (*PG-UP*), or page-down (*PG-DN*) keys. MLAB will cause the specified text to be placed on the screen by obtaining the required lines from the MLAB log-file. Hitting the *ENTER* key or scrolling back to the current command line will cause MLAB to leave scrolling mode and to re-enter command entry mode. In a Unix-based windowing system, the terminal emulator in

use provides scrolling facilities integrated with the window system in use. On Macintosh computers, you may scroll back as much as 32,000 characters in the MLAB dialog by moving the elevator box in the vertical scroll bar of the MLAB window with the mouse. Hitting any key will resume command input at the end of the MLAB dialog.

When a syntactically or semantically incorrect statement is encountered, an error message will be produced. On DOS PC's the MLAB line editor will be entered with the offending statement as the text to be edited. Use of the line editor is described in section 35, the EDIT statement. If the incorrect statement is in a do-file, however the editor will not be entered.

The MLAB statements are listed below; each statement is described in the indicated section. The descriptions specify the legal form (syntax) of the statements and discuss their meaning (semantics).

Various symbols are used within the syntactic descriptions of MLAB statements to represent the entities possible in MLAB. The following symbols are generally used throughout. E,A,B,X,Y are scalar-valued expressions. M and N are matrix-valued expressions. F is a function name. W is a plane-window-image name. C is a curve or axis name. However, there are occasional exceptions, which should not cause confusion.

The following rules apply to the syntax descriptions used in subsequent sections.

1. Single letters stand for names or expressions as indicated above.
2. Words (with two or more letters) should be typed exactly as shown.
3. Phrases enclosed in square brackets, [ ], are optional.
4. When phrases are enclosed in angle brackets, as in $< A|B >$, the user must select one of the phrases separated by |.
5. The construction T,... represents an arbitrary sequence of one or more instances of the construct T, each separated a comma. Thus, for example, Z1,... denotes all strings of the form "Z1,Z1,...,Z1".

The following words are MLAB lexical elements. These words may not be used as names of data objects.

ADJUST	FIT	LINE	ROW
AND	FILE	LINETYPE	SAVE
ANGLE	FINCHES	LIST	SCREEN
AT	FONT	LT	SHIFT
AXIS	FOR		SIZE
BLANK	FORMAT	NOT	SSAVE
BREAK	FRACT	OFFSET	THEN
COL	FRAME	ON	TITLE
COLOR	FRAMEBOX	OR	TO
CONSTRAINTS	FUNCTION	PAUSE	TYPE
DEL	HELP	PLACE	UNBLANK
DELETE	IF	PLOT	UNVIEW
DIFF	IFRACT	POINTTYPE	USE
DIR	IMAGE	PTFONT	VIEW
DO	IMAGEBOX	PTSIZE	WEIGHT
DONE	IN	PRINT	WINDOW
DRAW	INCHES	PRIORITY	WITH
EDIT	INIT	PT	WORLD
ELSE	INITIAL	RESET	WT
EXIT	LABEL	RESAVE	XAXIS
FCT	LABELFONT	RESSAVE	YAXIS
FFRACT	LABELSIZE	REUSE	

Note that the names of builtin functions are also reserved words. These names are listed in the builtin functions section.

MLAB has a collection of pre-defined scalars and strings, used for various purposes. Some of these have immutable values; others may be changed by the user in order to internally control some aspect the operation of MLAB. These are listed below in separate tables. The first table presents the fixed constants which may not be changed or deleted. The second table presents the control variables which may be changed; but if they are given inappropriate values, unpredictable results may be obtained.

The machine-dependent numeric constants MAXPOS, MAXNEG, and MACHEPS are provided. MAXPOS is the largest positive floating point value. MAXNEG is the largest negative floating point value. MACHEPS is the smallest positive floating point value such that $1 + \text{MACHEPS} > 1$. For machines using an Intel 80x87, MAXPOS = 1.7976..X10^{308}, MAXNEG = -1.7976... X 10^{308}, and MACHEPS = 1.1113... X 10^{-16}.

The colors listed below show the values used for VGA displays; other values are used for other display devices.

MLAB pre-defined scalar constants

ADAMS	1
ALTERNATE	5
AQUA	machine dependent value
ARROW	14
ARROWTIP	31
AXES	14
BLACK	machine dependent value
BLUE	machine dependent value
BOTTOM	4
BROWN	machine dependent value
CENTER	0
CHARTREUSE	machine dependent value
CIRCLE	13
CLEAR	-2
CONSETS	15
CONSTANTS	2
CONSTJAC	2
CROSSPT	2
CURVES	12
DASHED	2
DBAND	26
DDBAND	30
DLBAND	28
DUBAND	29
DRBAND	27
DIAMOND	16
DOTPT	5
DOTTED	8
DPS	16
DTICK	12
EGAC	"EGAC"
EGAM	"EGAM"
FALSE	0
FUNCTIONS	5
GEAR2	2
GEAR	3
GREEN	machine dependent value
GREY	machine dependent value
HBAR	6
INITIALS	7
LBAND	24
LEFT	2
LJ2L	"LJ2L"
LTICK	9
MACHEPS	machine dependent value
MARKER	6
MATRICES	6
MAXNEG	machine dependent value
MAXPOS	machine dependent value
MIXED	0
NBAR	21

NONE	0
NULLM	0 X 0 matrix
NUMERICAL	1
OCTAGON	7
ODERPT	0
ORANGE	machine dependent value
PI	3.14159265358
PINK	machine dependent value
PSCL	"PSCL"
PSL	"PSL"
PURPLE	machine dependent value
RBAND	23
RPS	17
RED	49
RIGHT	3
ROSE	machine dependent value
RTICK	10
SCALARS	4
SOLID	1
STRINGS	9
SQUARE	4
SYMBOLIC	0
SYMBOLS	0
TBAR	22
TITLES	13
TOP	1
TRIANGLE	3
TRUE	1
TURQUOISE	machine dependent value
UBAND	25
UTICK	11
VBAR	1
VGAC	"VGAC"
VGAM	"VGAM"
VIOLET	machine dependent value
VMARKER	7
WEXPAND	2
WFIX	0
WFLOAT	3
WINDOWS	11
WHITE	machine dependent value
WNICE	11
WSHRINK	1
WSLACK	8
XDEV	15
XERRBAR	17
XLOGTICK	19
XPT	8
YELLOW	machine dependent value
YERRBAR	18
YLOGTICK	20

MLAB pre-defined control variables

APPENDSW	1
CMDERRSW	0
CUTOFF	0
DGRADSW	0
DISASTERSW	3
DISPLAY	"VGAC" or installed val
DOFILEDIR	""
DOSTEP	0
ECHODO	1
ERRFAC	.001
FILEDIR	"" (empty string)
FITNORM	2
HESSMSW	0
INCOLOR	1
JACSW	1
LINEARSW	1
LINELEN	78
LSQRPT	8
MANDSW	0
MAXITER	20
METHOD	2
MFORMAT	"9 1 3 0 0 1 3 0"
NFORMAT	"-3 9 0 0 2 0"
MAXITER	20
NAMESW	0
ODESTR	""
OUTCOLOR	13
PLOTDEV	"PSL"
PSLINEW	2
REPORTSW	0
RESIDSW	0
ROUNDSW	3
SCREENCOLOR	0
STATPSW	1
STOCHSW	0
SYMDSW	0
TOLSOS	0.001
VIEWX0	0.2
VIEWX1	1
VIEWY0	0.2
VIEWY1	1
WARNSW	1

13. Assignment Statement

SYNTAX: V = U

Examples:

```
A = 1.5
B[i,j+1] = 2*EXP(1)
Q COL (3,1) = (M ROW 1:2)'+.5
W1= W
M1 = HISTO(POINTS(F,READ(FILE)))
```

V is a scalar name, a matrix name, an undefined name, a function name, a matrix element designator or a submatrix designator. U is a matrix expression or scalar expression, a window name, a curve or axis name, a title name, or a function name.

The assignment statement is used to assign or reassign values to a matrix, scalar, or string, and to change the name of a function or window.

If V is a function name and U is a scalar value, then the *previous value* of the function V is set to U. The previous value of a function is the value which the function has if it is referred to without arguments in a function body. This has utility in various circumstances, such as schemes of recursive functions.

If V is an undefined name and U is a scalar expression, V will be made a scalar with value equal to the value of U. If V is an undefined name and U is a matrix expression, V will be made a matrix equal to the value of U. If V is a scalar name or matrix name and U has the same type as V, V will be set equal to U. If V is a scalar name and U is a matrix, the value of V will be set to U[1,1]. If V is a matrix name and U is a scalar, each element of V will be set to the value of U. The above information is summarized in the table below for the assignment V=U.

initial V	U	result V
undefined	scalar	scalar
undefined	matrix	matrix
scalar	scalar	scalar
matrix	matrix	matrix
scalar	matrix	value of U[1,1]
matrix	scalar	V[i, j]=U for all i, j

If V is a matrix element designator, it has the form M[A,B] or M[A], where A and B are scalar expressions and M is a matrix name or undefined name.

M[A,B] denotes the matrix element in the A-th row and B-th column of the matrix M. M[A] is an abbreviation for M[A,1].

The specified element of M, M[A,B], will be set to the value of U if U is a scalar or to U[1,1] if U is a matrix. If M does not have at least A rows and B columns, it will be enlarged as necessary, by adding zero-valued elements. If M was previously undefined, it will be made a matrix with A rows and B columns. In this case, all elements except the [A,B] element will be set to 0.

If U or V is a submatrix designator, it must have one of the forms

M ROW N
M COL N
M ROW N COL Q
M COL N ROW Q

where M is a matrix or undefined name and N is a scalar or matrix and Q is a scalar or matrix. If N is a scalar or Q is a scalar, it is treated as a 1×1 matrix with its [1,1] element equal to the value of N or Q, respectively. U may also be a matrix or scalar; again a scalar is treated as a 1×1 matrix with its [1,1] element equal to the value of U.

The appropriate elements of the row or column specified by the [1,1] element of N are replaced by (or defined to be) the elements of the first row or column of U. If necessary, M will be expanded by adding rows or columns of zeros. If the first row or column of U has P elements and the designated row or column of M specified by N[1,1] has $R > P$ elements, then the last R-P elements of the designated row or column of M will not be changed.

Next, the row or column of M specified by N[2,1], if N[2,1] exists, has its initial elements replaced by the elements of the second row or column of U; and this process continues until the end of N is reached, scanning N row by row to obtain successive row or column indices to M. If the rows or columns of U are exhausted, the assignment process recycles, starting again with the first row or column of U.

If V is an undefined name and U is a window, then the window U will be *renamed* to V. All the properties of the window U, and all the members (curves, axes, and titles) of U will be placed in V. The names in U which depend upon the window name are: U.LEFT, U.RIGHT, U.TOP, and U.BOTTOM for title names and U.XAXIS and U.YAXIS for axis names. These objects will be renamed, so that U.TOP becomes V.TOP, etc. The symbol U will then be deleted.

Special Assignment Statements

The special assignment statement:

FILEDIR = S

redefines the string variable FILEDIR, which is prefixed to user file names specified to be read or written by MLAB. S must be a string constant, i.e. a quoted string, or the name of a string variable. The main use of filedir is to specify the path, i.e. the drive and directory, for user-requested files which are read or written by MLAB. Of course, illegally-long path names must not be produced. A typical DOS or WINDOWS path specification is "C:\USER\PROJ1\". A typical Unix path specification is "/u/uname/". A typical Macintosh path specification is "disk:folder".

Files which are read into MLAB (by the DO, READ, USE, REUSE statements), or written out from MLAB (by the PRINT, PLOT, SAVE, RESAVE statements) will be automatically prefixed with the FILEDIR string. If FILEDIR is not set, it defaults to the null string, and files are read and written using the current directory, e.g. C:\USER for DOS. Upon setting the FILEDIR string to a different path, e.g. C:\USER\PROJ1\ for DOS, subsequent files will be read and written using the directory specified by the new path. When FILEDIR represents a path, then for DOS it should end in a back-slash character (\), for Unix it should end in a slash character (/), and for Macintosh it should end in a colon (:). If this final delimiting character is not present, it will be added.

The special assignment statement:

DISPLAY = T

is used to select a display device for use in graphics mode, i.e. with the VIEW statement. T must be the name of one of the available display types . The available display adapters for a PC are:

EGAM	640 x 350 monochrome
EGAC	640 x 350 with 16 colors
VGAM	640 x 480 monochrome
VGAC	640 x 480 with 16 colors

For NeXT-based systems, the Display Postscript device DPS is available.

These names are available as pre-defined MLAB string variables.

On a PC, graphics will work properly only if the display device which is selected by a display assignment is the same as (or is compatible with) the display device actually present. The consideration

here is the display adapter, i.e. the video card, and not the display monitor. For example, TTL EGA monochrome displays work perfectly well with the display assigned as EGAC. Different colors appear on the screen as somewhat distinct grey levels.

The special assignment statements:

INCOLOR=C
OUTCOLOR=C.

INCOLOR is a control variable that determines the screen color of the text of all commands entered by the user in DOS systems. It is set by default to WHITE. The user can change this color to any of the predefined colors used in MLAB. For example, the user can type INCOLOR = GREEN.

OUTCOLOR is a control variable that determines the screen color of the text of MLAB's response to all user commands in DOS systems. It is set by default to YELLOW. The user can change this color to any of the predefined colors used in MLAB. For example, the user can type OUTCOLOR = RED.

Assignment statements are also used to change the values of various other reserved symbols such as the control parameters for curve fitting and for numerically solving systems of differential equations. These are ordinary assignment statements, but they are discussed in the sections on the TYPE statement, the FIT statement, the PLOT statement and the INTEGRATE operator.

14. TYPE Statement

SYNTAX:

TYPE U,...
TYPE <SCALARS | MATRICES | FUNCTIONS | INITIALS | CONSTANTS |
WINDOWS | CURVES | AXES | TITLES | CONSETS | SYMBOLS>
TYPE FILE S
TYPE DIR S

EXAMPLES:

```
TYPE SIN ON 1:5:.1
TYPE A,B,W
TYPE F,F'X, F(3.2),F(X)
TYPE AXES
TYPE "ENTER X-VALUE"
TYPE FILE myfile.do
TYPE DIR "*.do"
```

U may be a matrix expression, scalar expression, function name, curve name, window name, title name, axis name, initial condition name, or the name of a set of constraints.

The statement TYPE U causes the value or definition of U, depending upon its data-type, to be displayed. As a convenience, the word TYPE may be omitted when a single object or data class list is to be typed-out.

TYPE SCALARS will cause a list of all currently defined symbols of type *scalar* to be displayed.

TYPE MATRICES will cause a list of all currently defined symbols of type *matrix* to be displayed.

TYPE INITIALS will cause a list of all currently defined symbols of type *initial condition* to be displayed.

TYPE FUNCTIONS will cause a list of all currently defined symbols of type *function* to be displayed. Both ordinary functions and derivative functions will be listed.

TYPE CONSTANTS will cause a list of all currently defined symbols of type *constant* to be displayed.

TYPE CONSETS will cause a list of all currently defined symbols of type *constraint set* to be displayed.

TYPE WINDOWS will cause a list of all currently defined symbols of type *window* to be displayed.

TYPE TITLES will cause a list of all currently defined symbols of type *title* to be displayed.

TYPE CURVES will cause a list of all currently defined symbols of type *curve* to be displayed. Both axis-type curves and ordinary curves are included in this list.

TYPE AXES will cause a list of all currently defined symbols of type *curve* which are *axes* to be displayed.

TYPE SYMBOLS will cause a list of all currently defined symbols of all types to be displayed.

TYPE F, where F is a user-defined function, results in typing out the name of the function, its argument list enclosed in parentheses (), an equal sign (=), and the text of the function definition. If F is a function and Z is a scalar symbol, the statement TYPE F DIFF Z (or TYPE F′ Z) will cause the symbolic differentiator to calculate the derivative of F with respect to Z and establish it as a function in the symbol table, unless the symbol F DIFF Z already exists. Then the function F DIFF Z is displayed.

TYPE F, where F is an initial condition, i.e. the initial condition for a differential-equation-defined function, results in typing out the name of the initial condition function, its argument list, an equal sign (=) and the symbolic expression of the initial definition.

TYPE T, where T is the name of a string, results in typing out the name of the string, an equal sign (=), and then the contents of the string without the surrounding quotes.

TYPE X, where X is an explicit quoted string, results in typing out the contents of the string without the surrounding quotes. Typing a string is often useful in a do-file to give directives.

TYPE C, where C is a curve, axis, or title, results in typing out the name of the curve, axis, or title, followed by a detailed listing of the properties of the curve, axis, or title.

TYPE W, where W is a window, results in typing out the name of the window, followed by a detailed listing of the properties of the window, including the names of all the curves, titles, and axes which are in the window.

TYPE S, where S is the name of a scalar (or constant), results in typing out the name of the scalar, an equal sign (=) and the value of the scalar.

TYPE E, where E is a scalar expression, results in typing out an equal sign (=) followed by the value of the expression.

Options for controlling number printing are discussed below in connection with the NFORMAT control string.

TYPE U, where the name U is undefined, results in typing out the name of the unknown object, an equal sign (=) and the word *unknown*.

TYPE M, where M is the name of a matrix, results in typing out the name of the matrix, the number of rows and columns in M, and the successive rows of the matrix on successive lines, preceded by the row number.

TYPE E, where E is a matrix-valued expression, results in typing out the number of rows and columns in the matrix-valued expression, and the successive rows of the matrix expression on successive lines, preceded by the row number. If the matrix has more columns than will fit on the screen, each line will wrap around showing as many complete columns as possible on each line.

Options for controlling matrix printing are discussed below in connection with the MFORMAT control string.

TYPE Q, where Q is a constraint set, results in typing out the name of the constraint set, an equal sign (=), followed by the list of constraints, as a string.

TYPE FILE S, where S is a name or an explicit quoted string, results in typing out the text of the file named S or named by the text in the string. The file name is preceded by the FILEDIR string in order to obtain the fully-qualified name of the file.

TYPE DIR S, where S is a string variable or an explicit quoted string, results in typing out the names of some or all of the files in the specified directory. The directory name is implicitly preceded by the FILEDIR string, which may contain path information in order to obtain the fully-qualified name. For the PC, DOS wild-card conventions are supported.

TYPE A,... , where A,... denotes a list of one or more objects of the above types, results in a typeout as described above for each of the objects in the list.

The MLAB control variable NAMESW determines whether the name of a variable to be typed out is prefixed to its value. In the case of matrices, NAMESW also controls whether the matrix size and row index values are typed out. If NAMESW is unequal to 0 (i.e. set to TRUE), the name (and size and row index values for matrices) will be typed out. This is the default. If NAMESW is equal to 0 (i.e. set to FALSE), the name (and size, and row index values for matrices) will not be typed out.

There are two permanent string-valued control variables in MLAB which may be used to control the printing of scalars and matrices. These are NFORMAT and MFORMAT.

NFORMAT

A format for numbers to be typed out by TYPE or PRINT statements is set using the string variable NFORMAT. NFORMAT may be assigned a string value S, where S may be either:

- a C-language format string (%le, %lf or %lg, with associated options), or
- a string of the form "H K A B R E" (the separating blanks must be present) where

  - H is the number of integer part digits or precision code when $H < 0$.
  - K is the number of fraction part digits or the total number of digits (precision) when $H < 0$.
  - A is the format code for integer part.
  - B is the format code for fraction part.
  - R is the required number of significant digits for numbers with negative exponents.
  - E is the optional/mandatory exponent code (0 for no exponent).

If $H \geq 0$, it and K together represent the maximum number of output digits allowed in the integer and fraction parts, respectively, of the output number string. When H+K is less than the number of significant digits in the number being formatted, digits are moved between the integer and fraction parts so that the absolute value of the associated power-of-ten exponent is minimized. When H+K is larger than the number of significant digits, digits are distributed between the integer and fraction fields of the number so as to maximize the number of digits displayed, unless directed differently by the value specified for R in the case of negative power-of-ten exponents.

If $H < 0$, the number of digits in the integer and fractional parts of the number are allowed to vary but must add up to a maximum of K. Let h be the maximum number of integer digits and let k be the maximum number of fraction digits to be used. When $H < 0$, the values of h and k are chosen so that $h + k \leq K$; this is equivalent to placing the decimal point among the digits of the number being formatted and adjusting the associated power-of-10 exponent accordingly. The decimal point is placed to permit a zero exponent if possible. If a zero power-of-ten exponent is not possible, then three alternatives are supported, depending upon the specified value of H.

If H = -1, the decimal point is placed to minimize the absolute value of the power-of-ten exponent.
If H = -2, the decimal point is placed before any of the digits ($h = 0, k \leq K$).
If H = -3, the decimal point is placed immediately after the 1st digit ($h = 1, k \leq K - 1$).

Alternatively, if H = -4, the decimal point is placed immediately after the first digit, and the power-of-ten exponent is determined by this placement.

If the number of significant digits to be placed in the resulting formatted string is less than K, then if $h > 0$, k is taken to be K-h, so that the extra "fillable" positions are considered to be fractional-part digits; otherwise h is taken to be 1 and k is taken to be K-1.

Various examples for A = B = R = E = 0 include:

H	K	v=12.34	v=.01234
2	2	12.34	12.34E-3
-1 or -2	2	12	.12E-1
-1 or -2	4	12.34	.1234E-1
-1	1	1E1	.1E-1
-2	1	.1E2	.1E-1

E is the optional/mandatory exponent switch. If E does not equal zero, the result number string is expressed with E-notation, i.e. with an explicit power-of-ten exponent part. When E = 0, only number strings which require non-zero exponents include an explicit power-of-ten exponent part.

When R is not zero, it is used to indicate the extent to which significant trailing digits can be discarded by rounding in order to make an associated negative power-of-ten exponent closer to zero. If $R > 0$, and a zero exponent is not possible and the number of output digits exceeds R, then the number of output digits is reduced to R. If $R < 0$, only one output digit need be used if it permits a zero exponent. If a zero exponent is still not possible, however, R is converted to its absolute value and the case of $R > 0$ is assumed. The use of a non-zero value for R can often be justified in cases of fractions between -1 and 1. Preserving as many significant digits as possible sometimes means that clarity is sacrificed for useless extra precision. For example, output in the form .00012 (as produced with R = -2) may be preferable to 1.23456789e-4.

abs(R) should be less than the number of output digits: $h + k$. If this is not the case R is taken to be zero. If $R > 0$, then only R significant digits need be necessary if K is too small to permit a zero exponent. $R < 0$ means that only 1 significant (rounded) digit need be used if by doing so the exponent can be "increased" to zero. If a zero exponent is still not possible, however, then R is treated as its absolute value and the case of $R > 0$ is assumed. If R = 0, then none of the above special treatment should occur. For example, if H=2, K=3 and A=B=2 then, depending on R, we get the following results:

R	v=.0023456	v=.9876543	v=-.000489265
0	23.456E-4	98.765E-2	-48.927E-5
-1	.002	1.0	-.005E-1
1	.002	1.0	-.005E-1
2	.023E-1	.99	-.049E-2
-2	.002	1.0	-.049E-2

After the desired number of integer-part digits, h, and fraction-part digits, k, have been determined, then, when the number v to be printed is different from zero, it is scaled by 10^w as required to obtain $10^{-K} \leq 10^w \text{ abs}(v) \leq 10^H - 1$, and w is indicated in the power-of-ten exponent value signaled by the letter 'E'. Let $x = \lfloor 10^w \text{abs}(v) \rfloor$, and $y = 10^w \text{abs}(v) - x$.

A is a format code for the h-digit integer part, x, of the number v. The value A is to be interpreted as described below.

0: no leading 0's or blanks, except if $x = 0$ then one 0
1: no leading 0's or blanks, except if $v = 0$ then one 0
2: no leading 0's or blanks
3: one leading 0, no blanks
4: all leading positions are 0
5: no leading 0's, except if $x = 0$ then one 0 and all further leading positions are blanks
6: no leading 0's, except if $v = 0$ then one 0 and all further leading positions are blanks
7: all leading positions are blanks
8: one leading 0 and all further leading positions are blanks

B is a format code for the k-digit fraction part, y, of the number v, which is to be interpreted as follows:

0: no trailing 0's and if $y = 0$ then no decimal point appears
1: no trailing 0's but if $y = 0$ then the decimal point appears
2: one 0 when $y = 0$ and no trailing 0's otherwise
3: all trailing positions are 0's
4: all trailing positions are 0's, but if $y = 0$ then no trailing 0's appear
5: all trailing positions are 0's, except if y=0 then no trailing 0's appear and no decimal point appears
6: all trailing positions are 0's, except if $v = 0$ then no trailing 0's
7: all trailing positions are 0's, except if $v = 0$ then no trailing 0's appear and no decimal point appears

Common A and B format combinations are:

• A = B = 0, where padding blanks are placed before and after the decimal point and the value zero is shown merely as 0.

• A = 3, B = 0, where only the fractional part is padded with trailing zeros to indicate the same degree of precision throughout.

• A = 7, B = 3, where all numbers have the same number of places before (padded with blanks) and after (padded with zeros) the decimal point.

If it is desired for integers to have a decimal point (but no trailing zeros), then B should be set to 1.

The combinations: (A = 2, B = 0) and (A = 7, B = 6) result in the number zero not being printed out at all.

Examples of different combinations of A and B with H=K=4 and R=E=0 are:

	v=23.45	v = 0	v=-23	v=.45
A=0,B=0	23.45	0	-23	.45
A=2,B=0	23.45		-23	.45
A=0,B=3	23.4500	0.0000	-23.0000	0.4500
A=4,B=3	0023.4500	0000.0000	-0023.0000	0000.4500
A=6,B=1	23.45	0.	-23.	.45
A=7,B=6	23.4500		-23.0000	.4500
A=8,B=7	023.4500	0	-023.0000	0.4500

Some complete number format examples are:

• NFORMAT = "-4,5 0 3 0 1". All numbers will be in standard scientific notation that will line up in the same form in a column. All subsequently printed numbers will be in the form x.yyyyEz.

• NFORMAT = "2 4 4 3 0 0" . All numbers will have two digits before the decimal point, and four digits after the decimal point, in the form xx.yyyyEz, where the exponent will appear only if required.

The default NFORMAT string is "-3 9 0 0 -2 0", which yields numbers of the form xx...x.yy...y with a total of at most 9 digits, with an exponent only if necessary, and, if a negative exponent is generated, then only 2 significant digits need be retained in order to express the number as .0000000yy with no exponent part.

MFORMAT

The format of matrices which are output by the TYPE and PRINT statements is controlled by the MFORMAT string, the contents of which may be set by assigning a string Q to the string control variable MFORMAT. The string Q must be the text for a sequence of integers of the form "S L G T C R W A" (the separating blanks must be present) where:

S is the column width in characters when L = 0. s is ignored when L ≠ 0.

L determines how the column width (in characters) is to be computed. The options are:

0: use S as the column width

1: deduce each column width separately just large enough to contain all the numbers appearing in the column

2: same as L = 1 but drop leading and trailing blanks when deducing the column width

3: use the same width for each column which is chosen just large enough to contain all the numbers appearing in every column

The default for L is 1.

G is the number of blank spaces between matrix columns. The default for G is 3. G must be non-negative.

T is the number of blank spaces at the end of each row. The default for T is 0. T must be non-negative.

C is the target number of columns to be output on a single line. If $C \leq 0$ the maximum number possible is used. The default for C is 0. If the matrix has more than C columns, additional columns will appear on subsequent lines. However, each row will begin on a new line.

R is the row numbering switch (0: no row numbers, 1: row numbers). The default for R is 1.

W is the number of spaces to indent the second and subsequent lines occupied by a single row. The default for W is 3. W must be non-negative.

A is the column alignment option switch. The options for A are:

0: left-character alignment. Numbers are left-adjusted in each column.

1: right-character alignment. Numbers are right-adjusted in each column.

2: alignment by explicit or implicit decimal point. The decimal point occurs at the same position in each column.

3: alignment by explicit or implicit E of exponent. The E of the exponent occurs at the same position in each column.

4: alignment by explicit or implicit sign. If any number in a column is negative, positive numbers are shifted right one space so that the first digits are all aligned. The way to have plus signs appear explicitly is to use a C-format, e.g "%+le" for NFORMAT.

The default for A is 0.

The use of the column width switch L allows the user either to specify a fixed column width if L = 0, or to deduce an appropriate column width for a given matrix and given NFORMAT string. If L is set to 1 (or any number less than 0 or greater than 3), MLAB chooses a separate width for each column that is just large enought to allow the inclusion of each number with its given amount of digits and blanks. The option L = 2 is the same except that leading and treailing blanks are dropped. If L is set to 3, the same width, which is just large enough to include all digits and blanks, is used for every column. Hence with L = 0 or 3, all columns must be the same size. With L =0, if a number is too large to fit within the size given in S, "***" is printed out instead of the number. It is useful to note that with L= 1,2, or 3, the column width deduced by the program will vary with changes in nformat.

G is the number of spaces placed between entries within any given line. T is the number of spaces placed at the end of each row. G and T must not exceed the line length and, if either is negative, it is converted to 0.

If $C > 0$, MLAB always tries to print out C matrix element entries on each line. In general, it is a good idea to have C greater than or equal to the number of columns in the matrix so that each row occupies exactly one line. However, if the number of columns present is too large to be displayed on one line, some entries in the given row will have to be placed on the next line and possibly even on subsequent lines. If $C \leq 0$ the program places the largest number of entries possible in each line.

The row number is automatically printed out at the beginning of each row (but not each line if a row has more than one line) unless R = 0. The second and each subsequent line of a row will start in the column that is W spaces beyond the column where the first line begins. W cannot exceed the line length and if W is negative, it is converted to 0.

With left-character alignment (A = 0), the first character in each entry is directly above or below the first character of each other entry in the same column. Right-character alignment (A = 1) accomplishes the same thing but for the last character instead of the first.

For A = 2 or 3, all entries within each column are lined-up so that all implicit and explicit decimal points or E's are directly above or below each other within any given column. A = 4 requests sign alignment. For sign alignment, plus signs are inevitibly missing (unless nformat specifies "%+..." using a C-format option). Hence, when A = 4 and at least one minus sign is present in a column, MLAB will place an extra space in front of all positive numbers for an invisible "+" and line this extra space up with the minus signs.

If some number in a matrix cannot be correctly printed with the current MFORMAT (and NFORMAT), a string of asterisks (****) will be printed out instead.

The default MFORMAT string is "9 1 3 0 0 1 3 0" which specifies that each column width will be deduced, that 3 spaces occur between columns, that no blanks occur at the end of each row, that as many entries as possible be placed on each line, that row numbers be supplied, that secondary lines be indented 3 columns, and that numbers within columns be left-character aligned.

The MLAB control variable CUTOFF is used to modify the printed values of matrix elements. A matrix element value $V = M[i, j]$ is printed as zero if $\text{abs}(V) < \text{CUTOFF} \cdot \max_{i,j}(\text{abs}(M[i, j]))$.

The control variable LINELEN defines the maximum number of characters allowed in a single line of a matrix typed to the screen, the printer, or a file. By default, LINELEN is set at 78, because most display screens use 80 characters per line. When using the PRINT statement to print a matrix to a file, LINELEN can be set larger to allow for longer lines than can appear on the screen. The minimum value of LINELEN is 20. If it is set to a smaller value, 20 will be used.

15. PRINT Statement

SYNTAX:

PRINT U,...[IN F]
PRINT FILE F
PRINT <SCALARS | MATRICES | FUNCTIONS | INITIALS | CONSTANTS |
WINDOWS | CURVES | AXES | TITLES | CONSETS | SYMBOLS> [IN F]

EXAMPLES:

```
PRINT Q COL (1,3,5)
PRINT G,G DIFF A,G DIFF B
PRINT M,N,P+3 IN FILX
PRINT SYMBOLS IN DMP
PRINT FILE a.do
```

The PRINT statement is identical to the TYPE statement except that the text output of the various entities to be "printed" is sent to a disk file instead of to the display. The name of the disk file to be written may be specified in the IN-clause.

If no IN-clause is given, the file will be printed on the printer attached to the DOS device PRN. To print directly to a printer or some other device, use an in-clause with the appropriate device name (COM1, COM2, LPT1, etc. for DOS) where the printer is attached. If the specified device does not have a printer attached, the computer may lock up and need to be rebooted. If an IN-clause is given with some other name than the reserved DOS device names, it is treated as an output file name.

The FILEDIR prefix will be prefixed to the filename, except for the reserved DOS device names. If the filename in the IN-clause has an extension, it will be used. If it has no extension, the default extension .lst will be used. If a file with the same name already exists, then if the MLAB control variable APPENDSW is 0 (FALSE), the existing file will be replaced. Otherwise, the current output will be appended at the end.

Text files produced by the PRINT statement may be later read by other programs (including MLAB via the READ operator). When producing files for subsequent reading by MLAB, it is usually convenient to suppress row number printing (by setting the R element of the MFORMAT string to 0) or by setting NAMESW to FALSE (0).

Files produced by the PRINT statement may be printed later outside of MLAB using standard operating system facilities or other software.

In DOS and in Windows, and also in the MacOS, the PRINT FILE form of the print command may be used to obtain listings of disk files, either pre-existing, or created by other MLAB print commands, on the printer from within an MLAB session. Graphics files, in either Postscript or HP LaserJet II format may be printed as pictures by this command on printers, e.g. the HP Laserjet 4M, that have sufficient internal intelligence to cooperate with the operating system. Also, for DOS, the DOS printer driver (PRN) must have been previously loaded.

In general, an ASCII file F can be printed in DOS by using the command PRINT F. In Windows, an ASCII file F can be printed by "dropping into DOS" and using the DOS print command. On a Macintosh, an ASCII file F can be printed by selecting F in its folder and then selecting the PRINT option in the FILE menu.

The MLAB command PRINT FILE X IN Y has the effect of copying the file X to a file named Y.LST. To copy to a file with a completely specified name, e.g. Y, use a quoted string for Y.

16. FUNCTION Statement

SYNTAX:

FUNCTION F(X,...) = E
FUNCTION F DIFF A(X,...) = E
FUNCTION F DIFF X(X) = E
FUNCTION F′ X(X) = E

EXAMPLES:

```
FUNCTION F(X,Y)=A(I+INT(X))/Y
FUNCTION J1(T) = INTEGRAL(X,0,T,EXP(2*(-X*X)))
FUNCTION L(N) = F DIFF X(T)+LOOKUP(N,G)
FUNCTION R() = ROOT(X,0,100,P(X))
```

F is a previously undefined name or a function name, each occurrence of X is a distinct arbitrary name, and E is a scalar-valued expression.

The symbol FUNCTION may be abbreviated with the symbol FCT.

The FUNCTION statement is used to define a user function. For example, the statement `FUNCTION F(X,Y) = SQRT(X)+A*Y` will define a function of two arguments, X and Y, and one free variable parameter A. User functions may be defined in terms of other user functions and the built-in operators and functions. Unfortunately, however, no user function may reference a built-in function which returns a matrix as a result.

Definition of recursive functions is permitted. For example

```
FUNCTION FACT(X) = IF X <= 0 THEN 1 ELSE X*FACT(X-1)
```

correctly defines the factorial function (without checking that its argument X is a non-negative integer).

Matrix-valued expressions may not occur in function bodies. However, array element expressions, e.g. M[$i+1$], may occur in function bodies, and matrices may be supplied as as actual arguments to be matched with formal arguments used as matrices in this limited fashion. Functions themselves may be actual arguments of functions, matching formal arguments used as functions in the body of the invoked function.

All functions must have an argument list, which may be empty. Moreover, an argument may be a dummy in that it need not appear in the function body.

You may define functions which are derivatives by using the DIFF or ′ operator in the function name. Thus:

```
FCT F DIFF A(X) = A+X
FCT F DIFF X(X) = A*X*LOG(X)+1
FCT G DIFF T(T) = -G
FCT H DIFF Z(Z) = 1-H(Z-SIN(Z))
FCT F''X(X) = 2*X+F'X(X)^2 +F(X)
```

are valid function definitions. A derivative function is a differential equation-defined function if it satisfies two additional conditions:

• It is a first derivative function with a single argument, of the form F′ T(T). Currently, the use of higher order derivative functions, such as F″T(T), is not documented (but you may be able to guess how to do it).

• Its parent function is an initial condition, defined by an initial statement. The parent of the function F DIFF X is F.

The symbol DIFF may be abbreviated using a single quote ($'$). Thus, F$'$ X means exactly the same as F DIFF X.

Differential equations may contain delay terms. The notation for a delay term in a differential equation is entirely natural: the delayed argument value is used, e.g. F(T-A). Delays should be causal, i.e. not requiring values of the solution which have not yet been computed. When delay differential equations are solved, delays reaching back before the initial "time" will use the value computed in the corresponding INITIAL statement. Delays reaching forward beyond the current "time" will use the current value of the variable, but this convention is not consistent with the notion of a delay.

The way that delays are evaluated is by linear interpolation into the table of solutions as they are stored. In order for this interpolation to be accurate, this table must be sufficiently dense, i.e. the points where solution values are stored must be sufficiently closely-spaced. Although automatic step size control in the differential equation solver will cause the table to be point-wise accurate, linear interpolants will not necessarily be accurate if the table is insufficiently dense.

The matching of actual arguments to formal arguments and the interpretation of free variables is discussed in the *Expressions* section. The appearance of a free variable which is a function name without an argument list is a special situation. The previously-computed value of the function is denoted by such a variable. If no previous evaluation has occurred, the value is a very large number which will be detected, and generate an error, when an expression involving the function is evaluated. Note if a function F is defined with an empty argument list, then F() denotes the value of F with an empty argument list, while F denotes the previous value of F. If F is a differential equation which is being solved, then these two quantities are very nearly the same, since the solution is continuous. Builtin functions have no recorded previous value, and must be evaluated with arguments every time they are used.

There is an anomaly associated with this convention concerning the meaning of function names appearing without arguments in a function body; namely, consider: E(G) = G(H), and F(G,H) = G(H). When G is subscripted or invoked from the body of E, the real-value of H is the argument, but in F, the function H is the argument to G when the actual argument corresponding to H is a function. Thus if G(X) = X/2, and H is a function whose previous-value is 4, then E(G) is 2, but F(G,H) is undefined and an error will result.

Function bodies may include embedded assignment operations. This allows a function to assign values to several objects, in addition to returning a value. Indeed, these side effects, as they are called, may be the main reason for using the function. Assignment within a function body may not be specified with the usual assignment operator, i.e. the equal symbol (=), since this is reserved for the boolean equality test in function bodies. The underscore symbol (_) must be used for embedded assignments.

When a conditional expression, IF A THEN X ELSE Y, is used in a function body, only the selected value, X or Y, will be scanned and evaluated.

There are several functions available in MLAB which can only be used in function bodies, and not at top-level. These are the SUM, PRODUCT, EVAL, ROOT, and INTEGRAL operators.

When a previously-defined function is redefined, it and all of its derivatives are deleted before the new definition is accepted (a notification message is given). Re-defining an initial condition does not delete the associated derivative function.

17. INITIAL Statement (with ODE extension package)

SYNTAX:

INITIAL F(X) = E

EXAMPLES:

```
INITIAL F1(0) = 1
INIT G(Y) = EXP(.2*A)
```

The initial statement is used to define the initial condition for a differential equation-defined function or a difference equation.

The symbol INIT may be used as an abbreviation for the symbol INITIAL.

F is an undefined name, X is a scalar, and E is a scalar expression suitable as a function body.

F is assumed to be defined by a differential or difference equation or by a system of differential or difference equations, so that the derivative of F will be explicitly defined in a FUNCTION statement.

The expression, E, which yields the initial value of F at X is evaluated whenever a system of differential equations involving F is to be solved, or whenever a system of difference equations involving F is to be solved.

F is used as the name of the initial condition for the unknown function F. Since the function F is not explicitly known, this is not a conflict. One may see the initial condition by issuing the command TYPE F. An initial condition may be changed by a subsequent INITIAL statement which supercedes the current initial condition.

The expression E may contain variables which are to be fitting parameters, thus initial conditions may be adjusted by curve-fitting. This provides a means of solving simple boundary-value problems. The scalar X may not be one of the parameters to be fit, unless numerical derivatives are specified.

18. CONSTRAINTS Statement (with curve-fitting extension package)

SYNTAX:

CONSTRAINTS Q = {E,...}

EXAMPLE:

```
CONSTRAINTS CSET = {3+X-4*Y<=Z/5,1<Y<10,X>0}
```

A CONSTRAINTS statement is used to specify a set of linear constraints which can then be specified to be honored during curve-fitting with the FIT statement..

Q is the name of a set of constraints to be replaced or a previously undefined name. Each instance of E is an expression involving a relational operator (<, <=, =, NOT=, >, and >=)

The name Q is used as the name of the set of constraints. The name Q may later be specified in a FIT statement to cause the set of constraints to be used.

Legal linear constraint expressions are equalities or inequalities involving sums of terms which are of the forms: a scalar name, a constant, the product of a scalar and a constant, or a scalar divided by a constant.

Thus the following are legal constraint expressions.

```
X + 2*Y -3 < 0
4 + X/6 = 5
```

A pair of simple constraints of the form: $\{X < A, X > B\}$ involving only a single scalar X, together with any of the relational operators, may be expressed in the following combined form: $\{B < X < A\}$

The following are illegal constraint expressions:

`X*Y < 0`	(product of scalar variables is illegal)
`X^2 < 3`	(^operator is illegal)
`2*(X+1) < 4`	(parentheses are not allowed)
`2*X/3 < 5`	(multiplication and division are not allowed together)
`X > SQRT(2)`	(complicated scalar expressions cannot be evaluated)

19. FIT Statement (with curve-fitting extension package)

SYNTAX:

FIT (A,...), F TO M [WITH WEIGHT <G|N>],... [CONSTRAINTS Q]

EXAMPLES:

```
FIT (P,Q), FH TO ((1:10)&'V)
FIT (N), F1 TO M, G TO M WITH WEIGHT WX
FIT (P,Q,R,S,T),H TO DATA WITH WEIGHT WH
FIT (K,R), F1 DIFF T TO H1
FIT(A1,B1), MODEL1 TO NORM1D CONSTRAINTS Q
FIT (X0,Y0), X DIFF T TO 0&'0, Y DIFF T TO 0&'0
```

The FIT statement causes a carefully-tuned version of the Marquardt-Levenberg iterative curve-fitting algorithm, extended with optional gradient searching, to be applied to adjust the parameters (A,...) so as to minimize the simultaneous sum of squares for the various functions (F) (the model) with respect to the associated data points (M) (the observations) weighted by the associated weights (a weight function G or a matrix N). The parameters may be constrained by an optionally-specified set of linear constraints (Q), where Q is the name of a constraints set defined in a previously-given constraints statement.

For each function, F, which is to be fit to a matrix of data M, if F has k arguments, M must have $k+1$ columns, where the first k columns are the values of the k independent variables which correspond in order to the arguments of F, and the $(k+1)$-st column of M is the list of corresponding dependent variable values to be predicted by F.

Any of the functions to be fit may be the solution to a differential equation (system), defined by a function statement of the form function F DIFF T(T) = ... , and an initial statement of the form INITIAL F(A) = ... The solution function F and any of its derivatives up to the order of the differential equation for F may be used for curve fitting by using the appropriate function names, of the form F, F DIFF T, ... , etc. in the FIT statement.

The optional WITH WEIGHT clause may be used to provide either a matrix N to specify the weights explicitly in advance of the fit, or a function to compute the weights dynamically during the fit. If a vector of weights N is specified in a WITH WEIGHT clause, then the vector N must have NROWS(M) numbers in a single column. These quantities should be estimates of the reciprocals of the variances of the corresponding observations in the rows of M. WEIGHT may be abbreviated WT. If a weight function G is used, it must have two arguments. The first argument is a matrix argument which corresponds to the data matrix M itself. The second argument is a scalar argument i, which is the row number of the i-th data point of the matrix M. This function must be defined by the user, and must be available when the FIT statement is executed. At each iteration, the vector of weights G(M,1), ..., G(M,NROWS(M)) is computed and used. In addition to its use in iterative reweighting, a weight function may also be used to specify windowed weights, in which the weight at a particular data point depends upon the values of neighboring points, for example by smoothing.

Often the error in an observation can be considered as a fraction, s, of the observed value. In this case the variance of an observation, y, is $(sy)^2$, and an appropriate weight is $1/y^2$ since, except when several functions are being fit simultaneously with shared parameters, we may ignore the constant factor $1/s^2$. It would be preferable to use the reciprocals of the squares of the model values, of course, and this can be done by manually fitting several times with reweighting each time. When several functions are being fit simultaneously, weights are usually necessary to scale the various sum-of-squares residuals into a common range of values.

For proportional errors, the reciprocals of the squares of the values in the h-times smoothed data ($SMOOTH(M,h)$) can sometimes serve as weight values for the corresponding observations in M if

the observations are appropriately ordered. This smoothing provides weights which more closely correspond to the weights which would be derived from the model functions. The builtin function EWT is also available and may be preferable. It provides an enhanced version of this procedure for computing an estimated weight vector. EWT is convenient for fitting multiple functions simultaneously, e.g.

FIT (A,B), F1 TO M1 WITH WEIGHT EWT(M1), F2 TO M2 WITH WEIGHT EWT(M2)

because it provides a way to effectively scale the influence of two or more data sets which may have rather different magnitudes, and which, in unscaled form, might contribute inappropriately to the sum of squares (or p-th powers) which form the objective function being minimized.

The weights to be associated with a data matrix may also be specified to be computed with a specified function. Such functions are called weight functions. A weight function G for the data matrix M must be defined as $G(v, j) = ...$, where v is a matrix with NROWS(M) rows, and j denotes a row designator integer which corresponds to the j-th row of v. When the weight for the k-th row of M is to be computed, G is invoked with the actual arguments M and k. As a special case, a weight function can invoke the model functions themselves. This means that the dynamically-changing predicted values determined by the model may be used to determine the weights.

The parameters A,... must be scalars, or must be matrices whose individual elements are taken as the specified parameters. All parameters must be PRESET by the user to whatever values are desired as initial estimates. For scalar parameters, these estimates will be automatically adjusted to cause the initial guess to satisfy any constraints involved. Matrix parameters may not appear in constraints, however. It is sometimes important to give the best possible initial guesses. Avoid guesses which lead to degenerate situations where several terms of the model and the derivatives of those terms with respect to the parameters in question start out numerically proportional with the same ratio; in some cases such terms may remain proportional during the fit, and the associated parameters may not vary independently.

Some of the functions in the model to be fit may be defined by a system of differential equations. In general, differential-equation-defined functions and/or their derivatives may occur in the model, together with explicitly-defined functions, some or all of which may depend on the differential-equation-defined functions or their derivatives (such differential-equation-dependent functions are called *auxiliary functions*). The data for the several differential-equation-defined functions need not be specified with the same sets of values of the independent variable.

Of course, when a differential-equation-defined model is used, the appropriate differential equations and associated initial condition equations must be established. However, the initial values can be among the parameters to be fit; this is a form of the shooting method for solving boundary value problems.

The Hessian matrix of a function $S(a_1, a_2, \ldots, a_n)$ is a square $n \times n$ matrix H, where $H_{ij} = \partial^2 S/\partial a_i \partial a_j$. The fitting process requires that values of the Hessian matrix of the sum-of-squares (or p-th powers) function be estimated. For the Gauss-Newton approach used in MLAB, this Hessian matrix is estimated based upon the Jacobian matrix of the model functions. The Jacobian matrix of the functions $F_1(a_1, a_2, \ldots, a_n)$, $F_2(a_1, a_2, \ldots, a_n), \ldots, F_m(a_1, a_2, \ldots, a_n)$ is the $m \times n$ matrix J, where $J_{ij} = \partial F_i/\partial a_j$. Computing the Jacobian matrix entails computing derivative values of the model functions.

When differential-equation-defined models are used, or when matrix parameters are present, the derivatives of the model functions with respect to each of the dependent variables, evaluated at each of the observed values of the dependent variables, are computed numerically. Otherwise, symbolic differentiation is employed and the resulting functions are then evaluated as required.

In some cases, symbolic derivatives require too much space or time or are otherwise inappropriate. Numerical derivatives can be explicitly forced, when desired, by setting the SYMDSW control scalar as described below.

In the case of a differential-equation-defined model, MLAB will scan all the function bodies involved to deduce a list of names of derivative functions which constitute the system of differential equations upon which the model is based. This automatically-obtained information is analogous to the arguments for the INTEGRATE operator.

The name of a set of linear constraint relations, which are to be honored during the fit, may be specified in the CONSTRAINTS-clause, except that matrix-element parameters may not occur in constraints.

Additional control information is provided to the curve-fitting process through the following scalar control variables.

LSQRPT: LSQRPT controls the output on the display screen of various information at intermediate and final computational stages during the curve-fit. The bits of LSQRPT are interpreted as follows:

1: notification of each major iteration

2: results of each subiteration (line search)

4: parameter values at each subiteration (line search)

8: final parameter values, lagrange multipliers, and statistical summary data

Sums of these values may be formed to specify combinations of these options. Thus, for example, 0 gives a "silent" fit; 8 gives a "quiet" fit (only the final result is output); and 15 gives a full report of the fitting process. The default value of LSQRPT is 8.

MAXITER: MAXITER is the maximum number of full iterations to be performed before giving up. The default value of MAXITER is 20. If MAXITER is zero or negative, an error analysis is done, but the parameter values are not changed.

SYMDSW: SYMDSW may take on the values 1 (for symbolic derivatives) or 0 (for numerical derivatives). In the former case, symbolic partial derivatives are computed in construction of the Hessian. In the latter case, numerical partial derivatives are used. For differential equation models, numerical derivatives are always used, since symbolic derivatives are not computable. MLAB has the constant scalars SYMBOLIC (1) or NUMERICAL(0), available for assigning an appropriate value to SYMDSW. The default value of SYMDSW is 1 (SYMBOLIC).

Ordinarily, symbolic partial derivatives are preferable for reasons of accuracy. However, when fitting using complicated functions, the construction of the partial derivatives may occasionally exhaust the space alloted for functions in MLAB. In this unusual circumstance, it is possible to restart MLAB in its initial state, specify SYMDSW as NUMERICAL, and retry the fit.

FITNORM: FITNORM specifies the norm desired for measuring goodness-of-fit. The sum of the absolute residuals to the p-th power will be minimized, where FITNORM $= p$. Selecting the value of FITNORM to be less than 2 is a device for reducing the influence of outliers, sometimes called "robust curve-fitting". The default value for FITNORM is 2.

TOLSOS: TOLSOS is the required relative improvement in the sum of squares (or, more generally, the sum of p-th powers) in order that iterations continue. Let S denote the curent sum of squares (or p-th powers). Convergence will be declared when the present iteration has not improved the sum of squares (or p-th powers) by the factor $(1 - \text{TOLSOS})S$. The default value of TOLSOS is 10^{-3}. Larger values of TOLSOS will cause the curve fit to stop earlier, when a further reduction in the sum of squares (or p-th powers) is still possible. Smaller values of TOLSOS will cause the curve fit to continue, sometimes enabling the program to escape from a local region of the parameter space where convergence is slow. Excessively small values of TOLSOS, less than 10^{-5}, are seldom effective in achieving further significant improvements in the sum-of-squares (or p-th powers) or changes in parameter values.

DGRADSW: DGRADSW is a boolean-interpreted scalar. If DGRADSW is $\neq 0$ (i.e. is TRUE), the curve-fitting process will include a down-hill direction search which may allow a better minimum to be found in some cases of ill-conditioning. The curve-fitting process will proceed in a more conservative manner by searching for a minimum along the path from the Gauss-Newton estimate back to the current parameter point. If DGRADSW is 0 (FALSE), any sufficiently improved value will be accepted. The default value for DGRADSW is 0 (FALSE).

If, during the curve-fitting process, the user types P (for "print"), the LSQRPT status will be overridden and intermediate output of curve-fit data on the display will occur. Typing P is treated as a toggle, so that output will be suppressed when P is typed again.

If Q (for "quit") is typed, the fitting process will terminate after the current iteration. This also terminates any differential equation solving em and exits all do-files.

When termination has occurred, with or without convergence, the final parameter values that produced the least sum-of-squares value which has been discovered will be typed, together with corresponding standard deviations and dependency values unless LSQRPT is 0. In any case, the parameters are changed to the final values dictated by the fitting procedure.

The standard deviations of the parameter estimates have their ordinary meaning only when the model is linear in the parameters, when there are no active constraints, and when the data errors can be considered to be independent random samples of normally-distributed random variables with zero means. Even when these hypotheses are not satisfied, the standard deviations are still suggestive, however, and may be used as tentative confidence limits.

Whenever a FIT statement is executed, any data object named COVP is re-established automatically as an $m \times m$ (where m is the number of parameters in the fit) matrix of estimated parameter covariance values, so that COVP$[i, j]$ is the estimated covariance between parameter i and parameter j. The data object named STDEST is re-established automatically as an $m \times 1$ matrix of estimated parameter standard deviation values, so that STDEST$[i]$ is the estimated parameter standard deviation of the i-th parameter.

The COVP matrix is the estimated covariance matrix of the parameter estimators, obtained from the linear theory, scaled by the square of the RMS error which is SOSQ/(# of data points - # of parameters). This scaling helps correct for incorrect weights which may not be the reciprocal variances of the observations as they should be. The parameter standard deviations in STDEST are just the square roots of the diagonal values of COVP.

The data object named DEPVALS is re-established automatically as an $m \times 1$ matrix of dependency values, so that DEPVALS$[i]$ is the dependency value of the i-th parameter. DEPVALS$[i] = 1 - 1/(\text{COVP}[i,i]\text{COVP}^{-1}[i,i])$. Also, the data object named SOSQ is re-established automatically as a scalar whose value is the final weighted sum-of-squares (or p-th powers), which is the final value of the objective function being minimized. This value is useful for testing goodness-of-fit.

The data object named RSQUARED is re-established automatically as a scalar holding the determination coefficient R^2 defined below.

If the MLAB control variable RESIDSW has a positive value (by having been set with an assignment statement, then a matrix called RESID is created (or re-established) after the curve-fit. RESID has one column and as many rows as all of the data matrices which were used in the curve-fit. It contains all the unweighted residual values row-concatenated together. If RESIDSW has a zero or negative value, no new RESID matrix is created or reestablished.

When RESIDSW lies in (0,1], the elements of RESID are computed by taking the differences ($\text{data}_i - \text{model}_i$), where i runs over the dependent data values for the first model function, then for the second model function, etc. When RESIDSW > 1, these residual values are multiplied by the corresponding final weight matrix to produce "weighted residuals".

Multi-dimensional models are handled by simultaneously fitting the appropriate real-valued "coordinate" functions to the appropriate arrays of data. Simultaneous-fitting is sometimes called

'global-fitting'. For example FIT(P,Q), F1 TO D1, F2 TO D2 will adjust P and Q so as to minimize the combined sum of the sum-of-squares (or p-th powers) for F1 with respect to D1 and the sum of squares (or p-th powers) for F2 with respect to D2.

Each model function may be a function of several arguments (or of no arguments). The number of columns in the corresponding data matrix must be one greater than the number of arguments.

Model functions can call on other user functions in various ways. It is important (except in the case of differential equation-defined models, where numerical differentiation is always used) to remember the somewhat arbitrary choices which are made in MLAB for derivatives of certain functions, e.g. ABS, which do not have well-defined derivatives everywhere, since the derivatives of the model functions with respect to each of the parameters are automatically computed symbolically if not previously defined.

No matter how convenient the curve-fitting process is, there remains the problem of judging and validating the results. Is the model correct or incorrect? A correct model describes the data in accordance with some underlying theory which we accept as appropriate If the model is correct, are the parameter values obtained by curve-fitting near the true values underlying the data? What is the standard error associated with the computed parameter values? These questions are often unanswerable, or at best answerable only by means of subjective judgments. On the other hand, there are various facts and heuristics which can sometimes be applied to aid in the analysis of the results. In particular, the following points are of interest.

• If the model is linear in the parameters, and the error in the data is normally-distributed, and the weight for each data point is equi-proportional to the reciprocal of the variance for that point, then the fitted parameter values are also maximum-likelihood estimates and their standard deviation estimates are correct. When any of the above hypotheses fail, the standard deviation estimates are merely suggestive.

• The RMS (root-mean-square) error of a fit is a dimensional measure of the goodness-of-fit. It is an "average" weighted deviation of a data point from the value predicted by the model. It is not an absolute criterion, however, since equal RMS values could be obtained for a fit that has randomly distributed deviations and for a fit that shows systematic deviations. The RMS value which is typed out takes into account the specified weights, the number of degrees of freedom dictated by the number of data points, and the number of fitting parameters used. If the weights are all identical and correct, then the RMS value is the estimated standard deviation of the error in the data, denoted $\hat{\sigma}$. If you want the unweighted value of the RMS error, together with all other output in unweighted form, you may refit without weights, first setting MAXITER to zero, so that the parameter values will not be changed.

• One of the best ways to judge a fit is by examining a plot of the residuals, i.e. (theoretical–observed) deviations vs. the independent variable. If these points show any systematic deviation about zero, there may be reason to be suspicious. It is easy to generate such a residuals plot in MLAB.

• The determination coefficient, usually called R^2, is the fraction of the total unweighted sum-of-squares, corrected for the mean, which is accounted for by fitting the model to the data, in the case where the model is a straight line. The value R^2 is computed as:

$$1 - \frac{\sum_{i=1}^{n}(f(x_i) - y_i)^2}{\sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2/n}$$

The positive square root, R, is the estimated correlation of a random observed value with the corresponding predicted value produced by the model. R is also called the multiple correlation coefficient. R^2 is not printed-out when there are active constraints at the final estimated point, since it has no obvious meaning in this situation.

• The percent weighted data error is the "root-mean-square weighted average" percentage error over all the observations. The percent weighted data error value is $100v$, where v is the weighted deviation fraction value that minimizes the expression

$$\sum_{i=1}^{n}[|y_i - f(x_i)| - v|f(x_i)|]^2.$$

The weighted deviation fraction value v is output by MLAB.

• Some fits are very precise, that is, small variations in the best-fit parameter values no longer allow the model to be a good fit. Other fits may permit any of a wide range of "answers". In this case, one or more parameters can generally be changed to compensate for any small changes imposed on various other parameters. Dependency values provide a measure of the preciseness of a fit. Parameters with high dependency values (> 0.99) form a set whose values can be varied widely and still maintain a good fit. One may explore the significance of dependency values by varying the value of a parameter above and below the optimum value obtained when fitting, allowing the other parameters to vary to give the best fit under these circumstances, and then plotting the sum-of-squares of the deviations as a function of the value of the parameter being varied.

The dependency value of a fitting parameter b is defined as $1 - \text{evar1}(b)/\text{evar}(b)$, where $\text{evar}(b)$ is the estimated variance of b and $\text{evar1}(b)$ is the estimated variance of the parameter b in a fit where all other fitting parameters are held fixed at the final obtained values. Let σ_i be the standard error for the estimated value of the ith parameter, and let s_i be the standard error for the estimated value of the ith parameter *with all other fitting parameters held fixed at the final obtained values*. Note $\sigma_i \geq s_i$ where the discrepency is due to the greater uncertainty caused by the presence of the other parameters. Let α_i be the *standard error magnifying factor* that "inflates" s_i to the size σ_i; thus $\sigma_i = \alpha_i s_i$. Note $\alpha_i \geq 1$. If α_i is large, say greater than 100, then there are probably too many parameters in our model to be estimated based on the given data. The MLAB dependency value for the ith parameter is equal to $1 - 1/\alpha_i^2$; this value lies in the interval $[0, 1)$. If $\alpha_i = 1$, then the corresponding dependency value is 0, and as α_i becomes large, the corresponding dependency value approaches 1. As a rule of thumb, the occurrence of a set of parameter values with dependency values greater than .99 means the associated curve-fitting problem has too many parameters (too many "degrees-of-freedom") for adequate determination.

20. WINDOW Statement

SYNTAX:

WINDOW [A TO B, C TO D] [ADJUST <A | (A1,A2,A3,A4)>] [IN W1]

EXAMPLES:

```
WINDOW 0 TO 10, -1 TO 1
WINDOW LOG10(2) TO LOG10(5), 0 TO F(5) IN W
WINDOW ADJUST WNICE IN W5
WINDOW ADJUST (WFIX,WFIX,WEXPAND,WEXPAND) IN W1
```

A *window* is a rectangular region of a cartesian plane. In MLAB, curves are always drawn in a cartesian plane which has a single associated window. That portion of the plane which lies within the window rectangle will be visible when the window is viewed on the display. A window is defined by its limits: the minimum and maximum values of the horizontal (x-) and vertical (y-) coordinates of the rectangular region of the associated plane. Every window has an associated *frame*, which is a rectangular region on the screen of the display, and an associated *image*, which is a rectangular region within the frame on the screen to which the window is mapped.

W1 is an unknown or a window, and A, B, C, and D are scalar expressions. The WINDOW statement is used to define a window or to change the size or location of an existing window. The statement WINDOW A TO B, C TO D IN W1 will cause window W1 to have its lower horizontal (x)-axis limit be A, its upper horizontal (x)-axis limit be B, its lower vertical (y)-axis limit be C, and its upper vertical (y)-axis limit be D. If W1 does not exist, it will be created. The window limits used in a WINDOW statement are arbitrary. They can cover as small or as large a rectangular region as is desired and that rectangle can be located anywhere in the plane. Windows may have one or both of x- or y- upper window limits either greater or less than the corresponding lower window limit.

Both TO-clauses must be present or both must be omitted. If the TO-clauses are omitted, the default is 0 TO 10, 0 TO 10. If the IN clause is omitted, the default is IN W, where the name W specifies the default window which is provided by MLAB. If W does not exist, it will be created by using the window W saved in the MLAB save file called DW.sav. If the window specified in the in-clause has a name other than w, say W1, and does not exist, the WINDOW statement serves to create a window named W1, with a default image rectangle (0.125 to 0.875, 0.125 to 0.875 FFRACT) and a default frame rectangle (0 to 1, 0 to 1 FRACT).

The default window, named W, is defined by the following commands:

```
WINDOW 0 TO 10, 0 TO 10 ADJUST WNICE IN W;
FRAME 0 TO 1, 0 TO 1 COLOR VIOLET IN W;
IMAGE .175 TO .925, .125 TO .875 FFRACT COLOR BLUE IN W;
XAXIS  0:10!6&'0 LABEL 0:10!6 LABELSIZE .015 FFRACT OFFSET (-.01,-.025) \
FFRACT FORMAT(-3,5,0,0,2,0) IN W;
YAXIS  0&'0:10!6 LABEL 0:10!6 LABELSIZE .015 FFRACT OFFSET (-.09,-.01) \
FFRACT FORMAT(-3,5,0,0,2,0) IN W;
```

This default window W is supplied in the MLAB save-file DW.sav. It was constructed on a screen with a width-to-height aspect ratio of 1.33. If this default window supplied with MLAB is not satisfactory, for example, if an insufficient top border is present, you should start MLAB, execute the above statements, and then type the command: RESAVE W IN DW in order to reconstruct and resave an appropriate default window for your display. Subsequently, you should copy the new file DW.sav to replace the originally-supplied file; this newly-defined window will be your standard default window.

When a curve is added to a window, some points of the curve in the cartesian plane may be outside the current limits of the window. When this is true, the window limits can be expanded so as to contain all of the points of the curve. When a curve is deleted, it may then be possible to contain all the remaining curves within contracted window limits. When this is true, the window limits may be contracted so as to contain all the points of the remaining curves, exactly. Sometimes, however, the user may desire that the window limits only expand, or not change, and that different such conditions apply to each of the four window limits.

The ADJUST-clause controls window limit changing as curves are added or removed.

ADJUST <v|(a,b,c,d)>

The ADJUST-clause of the WINDOW statement is used to regulate the automatic adjustment of the window minimum-x, maximum-x, mininum-y, and maximum-y limits. The clause takes four integer-valued scalars a, b, c, and d in parentheses separated by commas, which, successively, refer to the minimum-x, maximum-x, mininum-y, and maximum-y limits of the window. If all four adjust codes are to be given the same value, it suffices to give a single integer v, which will be used for all four. Each integer is interpreted bitwise as follows to select any combination of the four options described below.

bit 0	(1)	0 means no shrink	1 means shrink
bit 1	(2)	0 means no expand	1 means expand
bit 2	(4)	0 means no slack	1 means 5% slack
bit 3	(8)	0 means not nice	1 means nice

expand, for a specified window limit, means that the window limit (or limits) is automatically increased, if necessary, when a curve is added to the window, so as to contain exactly all of the curves in the window.

shrink, for a specified window limit, means that the window limit is automatically decreased, if necessary, when a curve is removed from the window, so as to contain all the curves in the window.

fixed, for a specified window limit, means neither expand nor shrink, i.e. that the window limit is not automatically adjusted when curves are added or removed from the window.

slack, for a specified window limit, means that the window limit is increased by 5% of the window extent, either horizontal or vertically, depending on whether an x-limit or a y-limit is referenced, beyond the minimum shrink extent of the window.

nice, for a specified window limit, means that the window limit is increased by as much as 10% of the window extent, either horizontally or vertically, depending on whether an x-limit or a y-limit is referenced, beyond the minimum shrink extent of the window. The increase is chosen so any axis numbers will be "nice", i.e. that they will involve as few significant digits as possible.

match is defined to be slack and nice together, i.e. when both *slack*, (bit 2) and *nice* (bit 3) are set, then *match* is implied. *match* means that the associated window limits are expanded so that the aspect ratio of the window rectangle in world units matches the aspect ratio of the image rectangle in inches. Either the horizontal or the vertical window limits, but not both, will be changed. If *match* is specified for all four window limits, the effect of this expansion is to provide an isometric graph, with equal differences in x- or y-coordinates corresponding to equal lengths on the screen, when the screen dimensions have been accurately provided.

The bits in the ADJUST-clause scalars are set by adding powers of two. The several possibilities can be combined as desired. When both *slack* and *nice* are set, *match* is implied. In order for *nice*, *slack*, or *match* to have any effect, *expand* (bit 1) must be set, because no expand overrides *slack* and *nice*. Some useful constant scalars which may be used in the ADJUST-clause have been predefined in MLAB. These are:

WFIX	(0)
WSHRINK	(1)
WEXPAND	(2)
WFLOAT	(3)
WSLACK	(7)
WNICE	(11)
WMATCH	(15)

If no ADJUST-clause is given in a WINDOW statement, the default is ADJUST WFIX. Windows that are automatically created by DRAW, TITLE, and AXIS statements have the implicit ADJUST-clause WNICE.

21. FRAME Statement

SYNTAX:

FRAME [A TO B, C TO D [UNITS]] [COLOR Q] [PRIORITY P] [IN W1]

EXAMPLES:

```
FRAME 0 TO .5, 0 TO .5 IN W1
FRAME 1 TO 8, 1 TO 4 INCHES IN W1
FRAME COLOR BLUE PRIORITY 1
```

A frame is a rectangular region on the display screen defined by a FRAME statement, which may be implicit. This screen region will ultimately contain a sub-rectangle holding an associated picture, i.e. the image of a specified associated window, which usually has some curves drawn in it. A frame also has additional specialized properties associated with it, namely: a framebox, a priority, and a frame color.

W1 is an unknown or a window and A,B,C, D, and Q are scalar expressions. If W1 does not exist, the FRAME statement serves to create a window named W1, with a default window (0 TO 10, 0 TO 10), a default frame (0 TO 1, 0 TO 1 FFRACT), and a default image (0.125 TO 0.875, 0.125 TO 0.875 FFRACT). If there is no IN-clause, the default window W is referenced. If W does not exist, it will be created. In either case, the properties of the frame of the referenced window will be set by the FRAME statement.

The FRAME statement will cause the frame region associated with the window W1 to occupy the portion of the screen running, in the horizontal sense, from A to B, and, in the vertical sense, from C to D. If no explicit unit is given, A, B, C, and D are interpreted to be in the corresponding horizontal and vertical screen fraction units, called FRACT units. A, B, C, and D may optionally be given with an explicit unit. The unit options are INCHES or FRACT.

INCHES means physical inches with (0,0) at the lower left corner of the screen

FRACT means that 1 unit horizontally is the full horizontal extent of the screen and 1 unit vertically is the full vertical extent of the screen

A, B, C, and D must be non-negative, and may not exceed the limits of the screen, i.e. not greater than 1 in FRACT units, and not greater than the actual size of the screen in INCHES units. The frame must have positive orientation and finite extent, i.e. $A < B$, and $C < D$

Frames placed on the screen may not fill the screen completely. For DOS, the color of that portion of the screen not occupied by any frame may be set using the following special assignment statement.

SCREENCOLOR = C

where C is a color number. A number of useful colors have been predefined in MLAB as constant scalars with appropriate values. These are

CLEAR	YELLOW	AQUA
BLACK	BLUE	CHARTREUSE
WHITE	VIOLET	BROWN
GREY	PURPLE	ROSE
RED	ORANGE	TURQUOISE
GREEN	PINK	

Note that the EGA/VGA family of displays support a maximum of 16 distinct colors on screen simultaneously, chosen from the 64 available colors. The result of attempting to display more than 16 colors is that the oldest color in the current set of colors will be successively replaced by each

new color beyond the 16th. This effect can be avoided by restricting the choice of colors to the color names provided in MLAB.

The optional clauses of the frame statement are described individually below.

COLOR A

The frame region may be given a "background" color using the optional COLOR-clause. A frame color is specified by a scalar. A number of useful colors have been predefined in MLAB as scalars with the appropriate values for the specified display device. These are:

CLEAR	YELLOW	AQUA
BLACK	BLUE	CHARTREUSE
WHITE	VIOLET	BROWN
GREY	PURPLE	ROSE
RED	ORANGE	TURQUOISE
GREEN	PINK	

Note that the EGA/VGA family of displays support a maximum of 16 distinct colors on screen simultaneously, chosen from the 64 available colors. The result of attempting to display more than 16 colors is that the oldest color in the current set of colors will be successively replaced by each new color beyond the 16th. This effect can be avoided by restricting the choice of colors to the color names provided in MLAB.

The color CLEAR is special. It means no color. Whatever color is already on the display screen at the point where a pixel is placed with the color CLEAR will not be changed from what it was previously. Colors are all opaque, except for CLEAR. Only when the frame and image colors are both CLEAR may overlapping frames with a lower priority (set in a PRIORITY-clause) be seen. The default frame color is CLEAR. The default image color is CLEAR.

PRIORITY A

The frame priority, i.e. which frames appear to be on top of the others, may be controlled by the optional PRIORITY-clause. The priority is specified by a single scalar. Larger values of the scalar denote higher priority. The frame with the highest priority appears on top, i.e is drawn last. Frames with lower priority are drawn before frames with higher priority. If the higher priority frame has opaque colors, it will obscure any lower priority frames which it overlaps. If no priority is specified, the priority 0 will be assumed. For frames which do not overlap, which is the usual case, there is no need to specify a priority.

22. IMAGE Statement

SYNTAX:

IMAGE [A TO B, C TO D [UNITS]] [COLOR Q] [IN W1]

EXAMPLES:

```
IMAGE .5 TO .8.5, 1 TO 5 FINCHES IN W1
IMAGE .125 TO .875, .125 TO .875 FFRACT IN W
IMAGE COLOR BLUE IN W1
```

An image is a rectangular subregion of the frame region of the display screen associated with a particular window. All curves which are clipped to the window will appear within the image rectangle on the screen.

W1 is an unknown or a window and A,B,C, D, and Q are scalar expressions. If W1 does not exist, the IMAGE statement serves to create a window named W1, with a default window (0 TO 10, 0 TO 10), a default image (0.125 TO 0.875, 0.125 TO 0.875 FFRACT), and a default frame (0 TO 1, 0 TO 1 FFRACT). If there is no IN-clause, the default window W is referenced. If W does not exist, it will be created. In either case, the properties of the image of the referenced window will be set by the IMAGE statement.

The IMAGE statement will cause the image containing the picture of the window W1 to occupy the portion of the frame running, in the horizontal sense from A to B, and, in the vertical sense, from C to D. If no explicit unit is given, A, B, C, and D are interpreted to be in frame fraction units, called FFRACT units. A, B, C, and D may optionally be given with an explicit unit. The unit options are as follows.

INCHES means physical inches with (0,0) at the lower left corner of the screen

FRACT means that 1 unit horizontally is the full horizontal extent of the screen and 1 unit vertically is the full vertical extent of the screen

FINCHES means physical inches from the lower left corner of the frame

FFRACT means frame fractions, i.e. 1 is the full extent of the frame, both vertically and horizontally.

The user may not place any part of the image outside the frame, or, by extension, outside the screen. A, B, C, and D must be non-negative, and may not exceed the limits of the frame.

Values given in FFRACT units must lie between 0 and 1. Values given in FINCHES units must lie between 0 and the actual size of the frame vertically or horizontally. Values given in FRACT units must lie between the lower frame limit and the upper frame limit for the corresponding direction. Values given in INCHES units must lie between the lower frame limit and the upper frame limit in the corresponding direction.

The image rectangle must have a positive extent, i.e. $A < B$, and $C < D$.

COLOR X

If the clause COLOR X is present, the background of the image region of the associated window will be drawn with color X. If no COLOR clause is given then COLOR CLEAR is assumed. The following scalars are pre-defined in MLAB and are available for use as colors.

CLEAR	YELLOW	AQUA
BLACK	BLUE	CHARTREUSE
WHITE	VIOLET	BROWN
GREY	PURPLE	ROSE
RED	ORANGE	TURQUOISE
GREEN	PINK	

Note that the EGA/VGA family of displays support a maximum of 16 distinct colors on screen simultaneously, chosen from the 64 available colors. The result of attempting to display more than 16 colors is that the oldest color in the current set of colors will be successively replaced by each new color beyond the 16th. This effect can be avoided by restricting the choice of colors to the color names provided in MLAB.

The image colors are all opaque, except for CLEAR. Only when the frame color and the image color are both CLEAR may overlapping images with a lower priority (set in a PRIORITY-clause) be completely seen. The default image color is CLEAR.

23. FRAMEBOX and IMAGEBOX Statements

SYNTAX:

[NO] <IMAGEBOX | FRAMEBOX> [COLOR Q] [IN W1]

EXAMPLES:

```
NO FRAMEBOX IN W1
FRAMEBOX COLOR GREEN
IMAGEBOX COLOR BLUE IN W1
NO IMAGEBOX
```

The image and the frame of a window may each optionally be surrounded by a rectangular borderline of a specified color, called an imagebox and framebox, respectively. W1 is a window. If no IN-clause is given, the default window named W is referenced. If the specified window W1 does not currently exist, it will be created and given the default window limits (0 TO 10, 0 TO 10), the default frame (0 TO 1, 0 TO 1 FRACT), and the default image (0.125 TO 0.875, 0.125 TO 0.875 FFRACT). If no COLOR-clause is given, the borderline color will default to WHITE.

To remove a frame box or image box, the statements

```
NO IMAGEBOX [IN W1]
NO FRAMEBOX [IN W1]
```

may be used.

COLOR X

If the clause COLOR X is present, the imagebox or framebox will be drawn with color X. If no COLOR-clause is given then COLOR WHITE is assumed. The following scalars are pre-defined in MLAB and are available for use as colors.

CLEAR	YELLOW	AQUA
BLACK	BLUE	CHARTREUSE
WHITE	VIOLET	BROWN
GREY	PURPLE	ROSE
RED	ORANGE	TURQUOISE
GREEN	PINK	

Note that the EGA/VGA family of displays support a maximum of 16 distinct colors on screen simultaneously, chosen from the 64 available colors. The result of attempting to display more than 16 colors is that the oldest color in the current set of colors will be successively replaced by each new color beyond the 16th. This effect can be avoided by restricting the choice of colors to the color names provided in MLAB.

IN W

The IN-clause specifies the window in which the imagebox or framebox is to be drawn. If there is no IN-clause, the imagebox or framebox will be drawn in the default window, W.

If the IN-clause is omitted, the default window named W is referenced. If the referenced window does not exist, it will be created and given default window limits (0 TO 10, 0 TO 10), frame limits (0 TO 1, 0 TO 1 FRACT), and image limits (0.125 TO 0.875, 0.125 TO 0.875 FFRACT).

24. DRAW Statement

SYNTAX:

DRAW <C | M | C = M | C1 = C2> (optional clauses)

The optional clauses are:

```
LABEL M
OFFSET (A,B) [UNITS]
PLACE (A,B)
LABELFONT A
LABELSIZE A [UNITS]
ANGLE A
FORMAT (H,K,A,B,P)
COLOR <D | (A,B,C)>
LINETYPE <A | (L1,L2,L3,L4,L5,L6)>
POINTTYPE <A | X>
PTFONT A
PTSIZE <A | M > [UNITS]
IN W1
```

EXAMPLES:

```
DRAW M COL 1:2
DRAW POINTS(F,1:40) LABEL 1:40 OFFSET (-.02,-.02) FFRACT LINETYPE DASHED
DRAW TC3 POINTTYPE XPT, LABEL (1:10:2) PLACE (CENTER,CENTER)
DRAW TC3 LABELS NULLM
DRAW C1 = M COL (1,3) COLOR (WHITE,WHITE,YELLOW) IN W1
DRAW Q POINTTYPE "x" FONT 20
```

C and C1 are unknowns or curves; C2 is a curve; M is a matrix.

The DRAW statement is used to specify a curve and associated information in a specified window for display in the image region of the corresponding frame. A curve in MLAB is a finite set of points in two dimensions defined by the Cartesian coordinates specified by the matrix M. M must have either one or two columns. (If M has 3 columns, 3D-graphics is implied and you should consult section 27.) If M has one column, the 2-column matrix (1:NROWS(M))&′ M is used in place of M. With M having two columns, the first column of M is a list of the x-coordinates, and the second column is a list of the corresponding y-coordinates. M cannot be missing unless the curve is being redrawn. An easy way to create a matrix to be drawn which holds points from the graph of some function (i.e. a way to graph a given function) is to use the POINTS operator to obtain the desired matrix.

If the curve name, C, is omitted, then a new unique name of the form C#, where # is a unique integer, starting with 0 and incremented each time a new default name is required, is constructed and is used as the curve name.

A copy of the matrix M is retained with the curve information. M may thus be subsequently altered or deleted without affecting the curve C. The function CURVEM(C) will return a copy of the matrix associated with C.

When the curve name, C, does not name an already existing curve and no window is specified in an IN-clause, the pre-defined default window, W, will be used. Moreover the size and location of the window limits of W will be adjusted as necessary to accommodate the curve being drawn, i.e. as if an ADJUST WNICE clause had been used in a WINDOW statement with window limits which

just contained the curve. If there is an IN-clause, the curve will be drawn in the window specified in the IN-clause, which will be created, if it does not already exist. If the referenced window does not exist, it will be created to have frame (0 TO 1, 0 TO 1 FRACT) and image (0.125 TO 0.875, 0.125 TO 0.875 FFRACT).

When the curve name, C, is explicitly given, and has been previously drawn, then the DRAW statement is taken as a request to redraw the curve C using any new parameters now specified in the additional clauses. Parameters which are not given explicitly will retain their old values. This form of the DRAW statement may thus be used to change line-types, point-types, windows, curve matrices, etc., without disturbing other parameters of the curve. If there is an IN-clause and the window name W1 is given as an undefined symbol, the window W1 will be constructed, and the curve or axis will be removed from its current window and drawn in the newly-created window W1. If there is an IN-clause, and the window name W1 is an existing window different from that to which the curve belongs, the curve or axis will be removed from its current window and drawn in the existing window W1.

When the DRAW statement is of the form: DRAW C1 = C2 IN W1, where C1 is either unknown or the name of a curve, and C2 is the name of an existing curve, the DRAW statement is taken as a request to create a new curve C1 in window W1 which is a copy of curve C2. (Since curve names in MLAB are unique, it is not necessary to specify the window to which C2 belongs.) If curve C1 already exists, it will be removed and replaced by a copy of the curve C2. If the IN-clause is missing, the default window W is implied. If there is an IN-clause, the curve will be drawn in the window specified in the IN-clause, which will be created, if it does not already exist. If the referenced window does not exist, it will be created to have frame (0 TO 1, 0 TO 1 FRACT) and image (0.125 TO 0.875, 0.125 TO 0.875 FFRACT)

The DRAW statement is used for drawing curves, i.e. sequences of points, specified in WORLD coordinates, i.e. in the window, which will appear in the image.

The various optional clauses of the DRAW statement are discussed below.

IN W

The IN-clause specifies the window in which the curve is to be drawn. If there is no in-clause, the curve will be drawn in the default window, W.

LINETYPE <X | (X1,X2,X3,X4,X5,X6,X7) >

The LINETYPE-clause specifies the line pattern which is used to connect the points of a curve. The linetype is a scalar or, exceptionally, a list of seven scalars in parentheses. A number of linetypes have been pre-defined, and the following pre-defined named scalars are available in MLAB to refer to them.

NONE	(0)	no line
SOLID	(1)	solid line
DASHED	(2)	dashed line with short dashes
LDASH	(3)	dashed line with long dashes
DDASH	(4)	dashed line with alternating long and short dashes
ALTERNATE	(5)	alternating solid-empty line
MARKER	(6)	solid line with marker skipping
VMARKER	(7)	variable dashed line with marker skipping
DOTTED	(8)	dotted line
DOTMARK	(9)	dotted line with marker skipping
MARKER	(10)	fixed-size dashed line with marker skipping
SMARKER	(11)	solid line with marker skipping using spline curve
SVMARKER	(12)	variable dashed line with marker skipping using splines

Linetype ALTERNATE alternates between drawing a solid line and drawing no line at all between successive points.

Linetype MARKER draws solid lines connecting groups of points contained between "marker" rows of the curve matrix. The first row of a matrix drawn with this linetype contains the marker value in column-1. The marker value is not any special number, but should be chosen so that it does not appear as the x-coordinate of any point to be drawn, e.g. MAXPOS. This row is not drawn. Solid lines connect each successive pair of points until the next marker row. This process continues until the matrix is exhausted. The final row need not contain a marker value. Linetype DOTMARK is similar to MARKER except that dotted lines are used for drawing the marker-delimited paths. Linetype DASHMARK is also similar to MARKER except that fixed-size dashed lines are used for drawing the marker-delimited paths.

Linetype VMARKER is identical to to MARKER, except that the value in column-2 of the marker row must be a number between 0 and 1. This value is used to contruct dashed lines. The dash length for each marked group is the value in column-2 of its marker row, in horizontal FFRACT units. The undrawn line between each dash is of length 0.01 FFRACT. If the value in column-2 is 0, a dotted line will be drawn. If the value in column-2 is 1, a solid line will be drawn. If the value in column-2 is not in the range [0,1] no line will be drawn.

Linetypes SMARKER and SVMARKER are similar to MARKER and VMARKER, except that spline curves are used to draw the curves defined by the marked groups of points. The global MLAB variable FLATNESS has been added to allow the user to vary the "flatness" of the spline curve. This parameter is the same as corresponding parameter in the SPLINEP function.

The user may also specify a linetype by use of a 7-tuple GRAPHX-linetype line specification vector. (GRAPHX is the graphics package used in the construction of MLAB.) GRAPHX line specifications are composed of the following components: a 1st line segment, a gap, a 2nd line segment, and a 2nd gap the same size as the first gap. The middle of the 1st gap contains the first tick mark. The middle of the 2nd gap contains the 2nd tick mark. This pattern of line-segments, gaps and tick marks are repeated cyclicly. These components may be defined by means of the following seven scalars. They are listed in the order in which they must be given. These scalars have the following meanings:

line1 length of the 1st solid line segment

gap1 length of the gap between 1st 2nd solid line

line2 length of the 2nd solid line segment

tick1 length of the 1st tick mark

tick2 length of the 2nd tick mark

thick1 thickness of the boxes that form line segments and tickmarks

fillsw thick box fill switch (-1 means no filling; 0 means solid fill; $k > 0$ means use k horizontal fill lines)

The first six of these scalars have an associated unit which may be set with an optional units qualifier. If there is no units qualifier, the default unit is FFRACT. The options for units are

FFRACT horizontal frame fraction units: 1 = width of the frame

INCHES screen inches

FRACT horizontal screen fraction units: 1 = width of the screen

IFRACT horizontal image fraction units: 1 = width of the image

WORLD world vertical (y-axis) units

Some examples:

(0.1,0,0,0.2,0.2,0,0) defines tic-mark lines 0.2 FFRACT units long placed every 0.1 FFRACT units on a solid line.

(1e-4,0.1,0,0.2,0.2,0,0) defines alternating dots and tic-mark lines of length 0.2 FFRACT units placed 0.05 FFRACT units apart.

(0,.1,0,0.2,.2,0,0) defines tic-mark lines of length 0.2 FFRACT units occurring every 0.1 FFRACT units on an invisible line.

POINTTYPE <X | "Q">

The POINTTYPE-clause specifies the point symbol to be used for a curve or axis. X must be an integer. Thirty point-types have been predefined, and the following named scalars are available in MLAB to refer to them.

NONE	(0)	no point symbol
VBAR	(1)	vertical bar
CROSSPT	(2)	± shaped symbol
TRIANGLE	(3)	triangle symbol
SQUARE	(4)	box symbol
STAR	(5)	a five-pointed star
HBAR	(6)	horizontal bar
OCTAGON	(7)	eight sided figure
XPT	(8)	X-shaped symbol
LTICK	(9)	leftward tic mark
RTICK	(10)	rightward tic mark
UTICK	(11)	upward tic mark
DTICK	(12)	downward tic mark
CIRCLE	(13)	variable resolution circle
ARROW	(14)	vector field arrows
DOTPT	(15)	a small dot
DIAMOND	(16)	a diamond symbol
XERRBAR	(17)	horizontal error bars
YERRBAR	(18)	vertical error bars
XLOGTICK	(19)	horizontal logarithmically-spaced tics
YLOGTICK	(20)	vertical logarithmically-spaced tics
NBAR	(21)	normal bar
TBAR	(22)	tangent bar
LBAND	(23)	left solid band
RBAND	(24)	right solid band
UBAND	(25)	up solid band
DBAND	(26)	down solid band
DLBAND	(27)	left dotted band
DRBAND	(28)	right dotted band
DUBAND	(29)	up dotted band
DDBAND	(30)	down dotted band
ARROWTIP	(31)	directed arrowhead

The "TICK" pointtypes are especially useful for axes.

The pointtype UTICK produces tickmarks which start at the point and are drawn vertically up on the screen. Thus, when placed on an axis at the bottom of a graph, this pointtype produces tickmarks pointing into the graph region. When placed at the top of a graph, this pointtype produces tickmarks which point out of the graph region.

The pointtype DTICK produces tickmarks which start at the point and are drawn vertically down on the screen. Thus, when placed on an axis at the top of a graph, this pointtype produces tickmarks pointing into the graph region. When placed at the bottom of a graph, this pointtype produces tickmarks which point out of the graph region.

The pointtype LTICK produces tickmarks which start at the point and are drawn horizontally to the left on the screen. Thus, when placed on an axis at the right of a graph, this pointtype produces tickmarks pointing into the graph region. When placed at the left of a graph, this pointtype produces tickmarks which point out of the graph region.

The pointtype RTICK produces tickmarks which start at the point and are drawn horizontally to the right on the screen. Thus, when placed on an axis at the left of a graph, this pointtype produces tickmarks pointing into the graph region. When placed at the right of a graph, this pointtype produces tickmarks which point out of the graph region.

The pointtype ARROW produces arrows of specified sizes and directions. The PTSIZE-clause for this point type uses a 2-column matrix with n rows corresponding to the n data points being drawn, column 1 of which contains the lengths of the arrows (in ptunits units), and column 2 of which contains the angles (in degrees) of the arrows, which are drawn centered upon the points.

The pointtypes XERRBAR and YERRBAR are useful for statistical graphs. XERRBAR produces horizontal lines to the left and right of each point (with optional perpendicular "wingtips". The PTSIZE-clause for this point type specifies a 1- 2- or 3-column matrix M, with n rows corresponding to the n data points being drawn. If M has 1 column, then both the left and right horizontal error bars for the i-th point are the same length, given in $M[i, 1]$, in WORLD units. If M has 2 columns, the length of the left horizontal error bar for the i-th point is given in $M[i, 1]$, and the length of the right horizontal error bar for the i-th point is given in $M[i, 2]$; both of these are in WORLD units. If M has 3 columns, columns 1 and 2 are the lengths of the left and right horizontal error bars in world units, and column 3 is the length of the perpendicular wing tips given in the units optionally specified in the PTSIZE-clause. If no units are specified, FFRACT units are assumed for the wingtip lengths in column 3. If M has fewer than three columns, no wingtips are drawn.

The YERRBAR pointtype interprets the associated PTSIZE matrix in the same way as the XERRBAR pointtype, except that the error bars are vertical instead of horizontal, with column-1 corresponding to up-error bars and column-2 corresponding to down-error bars.

The XLOGTICK pointtype places ticmarks between successive points on a horizontal axis at a spacing corresponding to the base-10 logarithm of 2,3,...,9 multiples of the base points. The scalar specified by the PTSIZE-clause determines the length of the ticmarks of the base points. The ticmarks at the base points are twice as wide as those at the subsidiary ticmarks.

The YLOGTICK pointtype places ticmarks between successive points on a vertical axis at a spacing corresponding to the base-10 logarithm of 2,3,...,9 multiples of the base points. The scalar specified by the PTSIZE-clause determines the length of the ticmarks of the base points . The ticmarks at the base points are twice as wide as those at the subsidiary ticmarks.

The NBAR pointtype places solid line segments centered on the points of a curve. The line segments are normal to the piecewise linear representation of curve on which they lie, taking the mean of the normal to the lines to the two neighboring points for interior points, and the normal to the line segment from the end point to the first interior point for the end points. The scalar specified by the ptsize clause determines the length of the bars.

The TBAR pointtype places solid line segments centered on the points of a curve. The line segments are tangential to the piecewise linear representation of curve on which they lie, taking the mean of the tangent to the lines to the two neighboring points for interior points, and the tangent to the line segment from the end point to the first interior point for the end points. The scalar specified by the PTSIZE-clause determines the length of the bars.

The LBAND pointtype places solid line segments running horizontally left from the points of the curve to the left margin of the image rectangle.

The RBAND pointtype places solid line segments running horizontally right from the points of the curve to the right margin of the image rectangle.

The UBAND pointtype places solid line segments running vertically up from the points of the curve to the top margin of the image rectangle.

The DBAND pointtype places solid line segments running vertically down from the points of the curve to the bottom margin of the image rectangle.

The DLBAND pointtype places dashed line segments running horizontally left from the points of the curve to the left margin of the image rectangle.

The DRBAND pointtype places dashed line segments running horizontally right from the points of the curve to the right margin of the image rectangle.

The DUBAND pointtype places dashed line segments running vertically up from the points of the curve to the top margin of the image rectangle.

The DDBAND pointtype places dashed line segments running vertically from the points of the curve to the bottom margin of the image rectangle.

The "band" point-types described above may be used to place grid lines on a graph as follows. Take the coordinates of the tic-marks on the x-axis as the curve matrix for the vertical grid lines. Draw this curve with linetype none and pointtype duband. Take the coordinates of the tic-marks on the y-axis as the curve matrix for the horizontal grid lines. Draw this curve with linetype NONE and pointtype DRBAND. UBAND and RBAND might also be used, but they make more intense lines which may distract from the curves of the graph.

The ARROWTIP pointtype places arrowhead symbols directed along the curve at each point except the first.

In addition to the point symbols specified by a scalar, any character Q may be used as a point symbol by placing it in double quotes("Q"). In this case, the PTFONT-clause is used to select the font to be used for the specified character.

PTFONT F

A pointtype character may be drawn in a specified font by setting the point-symbol font number in the PTFONT-clause. The characters in all the available fonts are shown in tabular form in Appendix 1. The default font is font 5.

PTSIZE <S | M > [UNITS]

A point symbol may be drawn in a specified size by giving the point size in the PTSIZE-clause. The point size may be a scalar, in which case all points of the curve will have the same size point symbol. However, when the argument of the PTSIZE-clause is a 1-column matrix M, then the successive points of the curve will have differing sizes given by the scalars in the corresponding rows of M. If M has fewer rows than the number of points in the curve, the rows of M will be reused cyclicly. If M has more rows, only the initial rows of M will be used. The point symbol size may be optionally specified using a units qualifier as part of the PTSIZE-clause. The options for units are

FFRACT horizontal frame fraction units: 1 = width of the frame

IFRACT horizontal image fraction units: 1 = width of the image

INCHES screen inches

FRACT horizontal screen fraction units: 1 = width of the screen

WORLD world horizontal axis units

If no units qualifier is specified, the default is FFRACT, which has the effect of making the point symbols smaller when the frame is made smaller.

If no PTSIZE-clause is given, the default is taken to be PTSIZE .02 FFRACT.

LABEL L

The points of a curve or an axis may optionally be labelled with numbers. Each number text-string is placed horizontally at the corresponding point according to the (implicit or explicit) PLACE-clause. When drawing a set of data points, it is often convenient to label the points with their sequence number. In an axis, these numbers are suitable for axis labels. These numbers are specified in the matrix L given in the LABEL-clause. L is a scalar or a 1- or 2-column matrix. If l is a scalar, its value will be used for all the points of the curve.

If L has one column, the elements of L are the labels of the successive points of the curve. If NROWS(L) > NROWS(M) (where M is the curve matrix of C), then only the first NROWS(M) numbers in L are used as labels. If NROWS(M) > NROWS(L) then the elements of L will be reused cyclicly to label the points of the curve.

If L has two columns, $L[i,1]$ is the label for the $L[i,2]$-th point of the curve, where the points of the curve have sequence numbers 1,2, etc. If a number in L COL 2 references a non-existent point of the curve, e.g. 0, the corresponding label in L COL 1 will not be used.

A copy of the label matrix L is retained with the curve, allowing L to be modified or deleted without affecting the labels. The function LABELM(C) returns a copy of the label matrix associated with the curve C.

To remove existing numeric labels of a curve, the curve may be relabeled with an empty matrix. The matrix NULLM is predefined in MLAB for this purpose.

LABELFONT F

The point label numbers will be drawn in a font which may be specified in a LABELFONT-clause. If no label font is specified, the default font (5) will be used. The available fonts are shown in Appendix 1.

LABELSIZE L [UNITS]

The point label numbers will be drawn in a size which may be specified in a LABELSIZE-clause. If no label size is specified, the default size: 0.02 FFRACT will be used. The label size may be given a units qualifier. The options for label size units are

FFRACT horizontal frame fraction units: 1 = width of the frame

IFRACT horizontal image fraction units: 1 = width of the image

INCHES screen inches

FRACT horizontal screen fraction units: 1 = width of the screen

WORLD world horizontal axis units

If no units qualifier is specified, the default is FFRACT, which has the effect of making the point symbols smaller when the frame is made smaller.

OFFSET (X,Y) [UNITS]

The point label number text strings may be offset from the points of the curve in both the horizontal and vertical directions using the OFFSET-clause. It is useful in an x-axis, for example, to offset

the axis numbers above or below the horizontal line which forms the axis. Likewise, in a y-axis, the numbers are ordinarily moved to the left or right of the vertical line which forms the axis. Even when labelling ordinary curves, it is often convenient to move the point labels out of the way of the line conecting the points of the curve, although this is not always possible by a simple offset. X and Y are scalars which are, respectively the horizontal and vertical offset. The offset X and Y values may optionally be given a units qualifier. The options for offset units are

FFRACT frame fraction units (horizontal and vertical separately)

INCHES screen inches

IFRACT image fraction units (horizontal and vertical separately)

FRACT screen fraction units (horizontal and vertical separately)

WORLD user world units (horizontal and vertical separately)

If no units qualifier is specified, the default is FFRACT, which has the effect of making the offset smaller when the frame is made smaller.

PLACE(A,B)

The PLACE-clause is used to define the centering point of a numeric label string, relative to the offset from the point being labeled. The pre-defined MLAB named constants LEFT, CENTER, RIGHT, TOP, and BOTTOM are available for use in the PLACE-clause.

The options for A are LEFT (2), CENTER (0), and RIGHT (3).

The options for B are TOP (1), CENTER (0), and BOTTOM (4).

The pair (A,B) selects one of nine centering points associated with the rectangle containing the label string to be placed on the picture. Each label string will be placed with its centering point at the offset from the point being labeled. If no PLACE-clause is used, the default is (LEFT,BOTTOM).

ANGLE A

The ANGLE-clause is used to define the angle of each numeric label string with respect to a vector pointing horizontally to the right from the offset point, and measured counter-clockwise, in degrees, i.e. the usual understanding for angles. In the absence of an ANGLE clause, label strings will be drawn horizontally.

FORMAT (H,K,A,B,R,E)

A format for numeric labels is set using the FORMAT-clause. The FORMAT-clause has six scalar arguments: H,K,A,B,R,E where:

H is the number of integer part digits

K is the number of fraction part digits or number of digits (precision) when $h < 0$

A is the format code for integer part

B is the format code for fraction part

R is the required number of significant digits for numbers with negative exponents

E is the mandatory exponent code (0 for no exponent)

The arguments of the FORMAT-clause are identical to the corresponding elements of the NFORMAT string, and are explained in detail in connection with the TYPE statement.

COLOR <D | (A,B,C)>

If a COLOR-clause is present, the curve determined by the line-type will be drawn with color A, the point-type symbols will be drawn with color B, and the point labels of the curve will be drawn with color C. If only a single color is specified with a COLOR D clause, the line, the points, and the labels, will all be drawn in COLOR D. If no COLOR-clause is given then COLOR WHITE is assumed. The following scalars are pre-defined in MLAB and are available for use as colors.

CLEAR	YELLOW	AQUA
BLACK	BLUE	CHARTREUSE
WHITE	VIOLET	BROWN
GREY	PURPLE	ROSE
RED	ORANGE	TURQUOISE
GREEN	PINK	

The color CLEAR has the special meaning that it does not change the color of the screen from what it would have been had no drawing been done. Thus objects drawn in color clear are invisible.

Note that the EGA/VGA family of displays support a maximum of 16 distinct colors on screen simultaneously, chosen from the 64 available colors. The result of attempting to display more than 16 colors is that the oldest color in the initial set of colors will be successively replaced by each new color beyond the 16th. This effect can be avoided by restricting the choice of colors to the color names provided in MLAB.

25. AXIS Statement

SYNTAX:

<AXIS | XAXIS | YAXIS> <A | M | A = M | A1=A2> (optional clauses)

The optional clauses are

```
LABEL M
OFFSET (X,Y) [UNITS]
PLACE (X,Y)
LABELFONT X
LABELSIZE X [UNITS]
FORMAT (H,K,A,B,P)]
COLOR <C | (A,B,C)>
LINETYPE <A | (L1,L2,L3,L4,L5,L6,L7)>
POINTTYPE <A | X>
PTFONT A
PTSIZE A [UNITS]
IN W1
```

EXAMPLES:

```
XAXIS M COL 1:2
AXIS POINTS(F,1:40) LINETYPE DASHED
AXIS TC3 POINTTYPE XPT, LABEL (1:10:2,3) PLACE (CENTER,CENTER)
YAXISC1 = M COL (1,3) COLOR (1,2,2) IN W1
XAXIS XM LABEL XM COL 1, OFFSET(-.15,-.3) INCHES PTSIZE 2  FONT
19 IN W1
YAXIS AX1 LABEL Y, FORMAT(4,4,5,3,0) IN WW
```

A and A1 are unknowns or axes; A2 must be an axis. M is a matrix.

The AXIS, XAXIS, and YAXIS statements are used to define axes and associated information for a specified window to be displayed in the corresponding frame. An axis in MLAB is a particular kind of curve defined by a set of points in two dimensions whose Cartesian coordinates in world units are given in the matrix M. M must have exactly two columns; the first column is a list of the x-coordinates, and the second column is a list of the corresponding y-coordinates. M cannot be omitted unless the axis is being redrawn. Within MLAB, an axis is a particular kind of curve data object, so all of the options associated with curves apply to axes.

The several forms of the AXIS statement are used for drawing curves in the frame. *Such axis-curves remain fixed in the frame when the associated clipping window is changed*; this is the main difference between an axis and a curve In addition, unlike a curve, an axis is clipped to the frame region, not the image region.

The XAXIS statement is used for drawing x-axes, in which the x-axis point-label numbers are affinely recomputed when the window is changed. If the axis name, A, is omitted, the default name W1.XAXIS is implied, where W1 is the name of the window in which the axis is drawn, specified in the IN-clause. If no IN-clause appears in the XAXIS statement, the axis is to be drawn in the pre-defined default window W, so the axis name defaults to W.XAXIS.

The YAXIS statement is used for drawing y-axes, in which the y-axis point-label numbers are affinely recomputed when the window is moved. If the axis name, A, is omitted, the default name W1.YAXIS is implied, where W1 is the name of the window in which the axis is drawn, specified in the IN-clause.

If no IN-clause appears in the YAXIS statement, the axis is to be drawn in the pre-defined default window W, so the axis name defaults to W.YAXIS.

The AXIS statement is used for drawing axes which are neither x-axes nor y-axes, i.e. in which the axis numbers are not recomputed when the window is changed. In the AXIS statement, if the axis name, A, is omitted, then a new unique name of the form A#, where # is a unique small number starting with 0 and incremented each time a default name is required, is constructed and is used as the axis name.

A copy of the matrix M is retained with the axis information. M may thus be altered or deleted without affecting the axis A. The function CURVEM(A) will return a copy of the matrix associated with A.

When the axis name A does not name an already existing axis or curve and if a window is not specified in an IN-clause, the pre-defined default window W will be used. If there is an IN-clause, the axis will be drawn in the window specified in the IN-clause, which will be created if it does not already exist.

When the axis name A is explicitly given, and has been previously drawn, either as an axis or as a curve, then the AXIS statement is taken as a request to redraw A as an axis using any new parameters now specified in the additional clauses. Parameters which are not given explicitly will retain their old values. This form of the AXIS command may thus be used to change the linetype, pointtype, window, axis matrix, etc., without disturbing other parameters of the axis. If there is an IN-clause, and the window name W1 is given as an undefined symbol, the window W1 will be constructed, and the previously-existing curve or axis will be removed from its current window and drawn in the newly created window W1. If there is an IN clause, and the window name W1 is an existing window different from that to which the curve or axis belongs, the curve or axis will be removed from its current window and drawn in the existing window W1.

Let A1 be of type unknown or curve (or axis) and A2 be of type curve (or axis). If the form of the AXIS statement is any of

AXIS A1 = A2 IN W1
XAXIS A1 = A2 IN W1
YAXIS A1 = A2 IN W1

then the statement is interpreted as a request to make a copy of the existing curve or axis named A2 to be named A1 and to be located in the window W1. If the curve or axis A1 existed previously, it will be deleted and a new curve or axis named A1 with all the properties of the curve or axis A2, except that it will be placed in the window W1. If the axis A1 did not exist previously, it will be created. If there is no IN-clause, the default window W is referenced. If the referenced window w1 does not exist, it will be created and given the default window limits (0 TO 10, 0 TO 10), frame limits (0 TO 1, 0 TO 1 FRACT), and image limits (0.125 TO 0.875, 0.125 TO 0.875 FFRACT).

The optional clauses in the AXIS statement are all identical in syntax and function to the corresponding clauses in the DRAW statement.

26. TITLE Statement

SYNTAX:

[TOP | LEFT | RIGHT | BOTTOM] TITLE <T=S | T | S | T1 = T2> (optional clauses)

The optional clauses are:

```
AT (X,Y) [UNITS]
SIZE A [UNITS]
PLACE (A,B)
SHIFT F
ANGLE B
FONT C
COLOR D
IN W1
```

EXAMPLES:

```
TOP TITLE "REGRESSION PLOT"
TITLE T1 = "****",at (6,-1.5) WORLD
BOTTOM TITLE "F(X)" SIZE .02 FFRACT IN W1
TITLE T1, AT (6.5,-1.3) WORLD ANGLE 90
TITLE S1 = "peak region" ANGLE 40 COLOR BLUE IN W2
TITLE "A'2S'.5DB'R'19T123" AT (0,0) SHIFT .9
```

The TITLE statement is used to define a string of text associated with a specified window to be displayed in the corresponding frame.

T, T1, and T2 are title names or previously undefined names, S is a string (that is, a sequence of characters enclosed by a pair of double quote symbols("")), W1 is a window, and X,Y,A,B,C,D and F are scalar expressions.

When specified, the string S is written in the frame of the window W1, or of the default window W if no window is specified in an IN-clause. The center of the left edge of the first character is placed at the point (X,Y) with the text running horizontally to the right with symbols of size A. Then it is rotated as required.

If T is explicitly given, the string S is thus defined to be the string belonging to the title T, and any old string is lost. Otherwise a new title name will be used and S will become the string belonging to this new title. If S does not exist, the null string will be used.

If T is explicitly given and already exists, the specified clauses in the title statement will replace the existing attributes of the title with the new ones. Attributes of the title corresponding to unspecified clauses are unchanged.

If the form of the assignment in the command is T1 = T2, the effect is to make a new title, named T1 which is a copy of the existing title T2. T1 is placed in the window specified in the IN-clause.

Unlike the drawing of a curve, titles are drawn in the frame, not the image. If a string is placed to start outside the frame, it will not be drawn and can only be seen by explicit appropriate reframing.

The various optional clauses and qualifiers are described below.

TOP, BOTTOM, LEFT, and RIGHT

Preceding the word TITLE is an optional title kind. The options are TOP, LEFT, RIGHT, BOTTOM, and none. TOP titles are centered within the frame region above the top of the imagebox.

LEFT titles run vertically upward, and are centered in the left half of the frame region to the left of the imagebox. RIGHT titles run vertically downward, and are centered in the frame region to the right of the imagebox. BOTTOM titles are centered in the frame region below the bottom of the imagebox. If no title name was specified, the default title name will consist of the window name, a dot, and the title kind.

IN W

The IN-clause specifies the window in which the title is to be drawn. If there is no IN-clause, the title will be drawn in the default window, W.

When the title statement is of the form: TITLE T1 = T2 IN W1, where T1 is either unknown or the name of a title, and T2 is the name of an existing title, the title statement is taken as a request to create a new title T1 in window W1 which is a copy of title T2. Since title names in MLAB are unique, it is unnecessary to specify the window to which T2 belongs. If title T1 already exists, the old title will be removed and replaced by the new title. If the IN-clause is missing, the default window W is implied. If there is an IN-clause, the title will be drawn in the window specified in the IN-clause, which will be created, if it does not already exist. If the referenced window must be created, it will have frame (0 TO 1, 0 TO 1 FRACT) and image (0.125 TO 0.875, 0.125 TO 0.875 FFRACT).

AT (X,Y) [UNITS]

The AT-clause specifies the position of the place-point of the box surrounding the title string. The place-point is determined by the place-clause. The AT-clause coordinates have an associated units qualifier, which may optionally be specified. The units options are

FFRACT frame fraction units 1 = frame height and width separately

IFRACT image fraction units: 1 = height and width of the image separately

FRACT screen fraction units 1 = display screen height and width separately

INCHES screen inches from lower left corner of display screen

FINCHES frame inches from lower left corner of frame

WORLD world x,y coordinates

If no units qualifier is specified, it defaults to FFRACT.

SIZE X [UNITS]

The SIZE-clause specifies that the nominal height of the characters in the title. The size has an associated units qualifier, with the same options as in the AT-clause above. If no units qualifier is provided, it defaults to FFRACT, which means that the title will remain in proportion to the frame if the frame is re-sized.

The string need not be located within the area of the window or image. However, any characters which would be totally out of the frame will not be shown.

SHIFT F

The SHIFT-clause causes the title string to be placed shifted F units from the point (x, y) (specified by the AT-clause) where it would otherwise start. If no SHIFT-clause is specified, the shift value defaults to 0. The shifting is along a line in the direction implied by the ANGLE-clause if $F \geq 0$,

and in the opposite direction if $F < 0$. The string of length l starts at $|f|(d-l)$ units of the way along the shifting line of length d starting at (x, y). Note if $l > d$, the string starts at a place in the negative direction from (x, y) on the shifting line. The shifting line starts at (x, y) and extends to a point (p, q) where (p, q) is the closest point of intersection of the shifting line with the extended frame lines. If the frame is $x0$ to $x1$, $y0$ to $y1$, then the extended window lines are $x = x0$, $x = x1$, $y = y0$, and $y = y1$. Thus $d = \sqrt{(x-p)^2 + (y-q)^2}$. Note that SHIFT .5 has the effect of centering a string along its shifting line in the direction specified by the angle b specified in the optional ANGLE-clause, and SHIFT -.5 centers the string along its shifting line in the direction $b + 180$ degrees.

PLACE (A,B)

The PLACE-clause is used to define the centering point of a title string, relative to the AT-clause position. The pre-defined MLAB constants LEFT, CENTER, RIGHT, TOP, and BOTTOM are available for use in the PLACE-clause.

The options for A are LEFT (-1), CENTER (0), and RIGHT (1). The options for B are TOP (1), CENTER (0), and BOTTOM (-1).

The pair (A,B) selects one of nine centering points associated with the rectangle containing the title string . This centering point is placed at the point within the picture specified by the AT-clause. If no PLACE-clause is used, the default is (LEFT,BOTTOM).

ANGLE B

The clause ANGLE B, where B is a scalar value between 0 and 360 defines the angle along which the title is drawn.. If no ANGLE-clause is given, the string will run horizontally. In any event, the string is first placed running horizontally as specified by the AT-clause, and it is then rotated to its final position.

FONT F

The characters may be drawn in a specified font using the FONT-clause. There are currently 34 fonts. The available fonts are shown in Appendix 1. If no FONT-clause appears, the title is drawn using the default font (5). For an EGA or VGA display device, font -1 selects the character set on the video card.

Note that, in all fonts, a *carriage-return* in a string will reposition the cursor to overwrite the partially-written string, and a *linefeed* will drop down one line height before continuing.

COLOR D

If the clause COLOR D is present, the string will be drawn with color D. If no COLOR-clause is given then COLOR WHITE is assumed. The following scalars are pre-defined in MLAB and are available for use as colors.

CLEAR	YELLOW	AQUA
BLACK	BLUE	CHARTREUSE
WHITE	VIOLET	BROWN
GREY	PURPLE	ROSE
RED	ORANGE	TURQUOISE
GREEN	PINK	

Note that the basic EGA/VGA family of displays support a maximum of 16 distinct colors on screen simultaneously, chosen from the 64 available colors. The result of attempting to display more than 16 colors is that the oldest color in the initial set of colors will be successively replaced by each new

color beyond the 16th. This effect can be avoided by restricting the choice of colors to the color names provided in MLAB.

Embedded String Control Codes

The characters in the string S may be any mixture of upper and lower case. In order to include a quote ("), two adjacent quotes ("") must be entered in the string.

It is sometimes convenient to change font tables, character sizes, and other properties of the text to be displayed within one string. Thus, the following string control codes, may be embedded in a string with the indicated effect occurring. The character (') is used to indicate an embedded string control code. Two adjacent apostrophes in a string denotes the actual character (') which will then appropriately appear in the resulting displayed string.

'nS changes the size of all the following characters to be n times the baseline size.

'nW changes the relative width of the characters to be n times the baseline width, by keeping the height unchanged and modifying the width as required.

'nH changes the relative height of the characters to be n times the baseline height, by keeping the width unchanged and modifying the height as required.

'nT changes the font table number to be used from the current font to font table number n.

'nU shifts the following characters up off the baseline. The number n can be positive or negative where 1 unit of n represents the nominal height of a character. The value n need not be an integer. '.5U works fine for superscripts.

'nD shifts the following characters down from the baseline. The number n can be positive or negative and '1.6667D is equivalent to a linefeed.

'nA rotates each following character by n degrees incrementally, so that the second following character is rotated 2n degrees, and so on. The baseline is thus converted to a circular arc. '0A stops this incremental rotation and '-nA will go in the opposite direction.

'R resets the above parameters to what they were at the start of the string. Note that this allows you to specify a particular font such as Script (font 19) in the middle of a text string and then revert back to whatever font you started with, no matter what the starting font table was.

'Cx converts the character "x" to a control character. For alphabetic characters, upper and lower case are equivalent; thus 'CB and 'Cb both produce the octal ASCII code 002 corresponding to CTRL-b. This is the way in which the non-typable characters in the MLAB font tables are accessed. The defining formulas for the ASCII codes produced by 'Cx are

ASCII('Cx) = ASCII(x)-64 for ASCII(x) $\leq$ 96, and
ASCII('Cx) = ASCII(x)-96 for 96 $<$ ASCII(x) $\leq$ 122.

'M is equivalent to 'CM'CJ. 'CM is a CR(carriage return) (= octal 15), and 'CJ is a LF (line feed) (= octal 12).

'nB is used to specify n sequential backspaces, and fractional backspacing, as in '2.5B, is honored. Fractional forward spacing is also allowed, as in '-1.33B, by using $n < 0$. 'B is equivalent to '1B. Backspaces space back variable amounts according to the widths of the previous characters.

'nF is used to move forward n spaces. Fractional spaces, for example '.3F, are honored. 'nF is equivalent to '-nB.

'X is approximately the same as '.5B, but the amount backspaced is adjusted so that the text of the next character will be precisely centered on the text of the just-previous character.

'Y causes vertical bars to be inserted before each character, until another 'Y is given.

'nO changes the color of the succeeding characters to be color n.

'nI iteratively repeats the next drawn character n times, for a total of $n + 1$ occurrences.

'E extends to the boundary point just after the farthest character from the starting point and continues from there.

'Z underlines the following characters until another 'Z is given.

Title strings with embedded control codes cannot currently be properly shifted with a SHIFT-clause, or properly positioned horizontally with a PLACE-clause, because the true length of the title string cannot easily be determined within MLAB.

27. The 3D-DRAW Statement (with 3D-graphics extension package)

SYNTAX:

DRAW <S|[S=]M>[IN W][,LINETYPE A][,POINTTYPE B]
[NETCOLOR (P,Q)][COLORLIST H]

EXAMPLES:

```
DRAW Q COL 1:3 IN QW
DRAW U=M1, LINETYPE 7,NETCOLOR (2,4)
DRAW POINTS(S,CROSS(XV,YV)) LINETYPE HIDE
```

The 3D-DRAW statement is used to construct the data structures to display surfaces and space curves on a display screen. Syntactically, it has same form as an ordinary (2D) DRAW statement, but the matrix M, if given, must have three columns, S must be a surface (or unknown), and W must be a 3D-window (or an empty 2D-window or an unknown). Thus a 3D-DRAW statement is recognized by the types of its constituents.

A surface in MLAB is a finite set of points in three dimensions defined by the Cartesian coordinate values specified by the matrix M. M must have exactly three columns; the first column is a list of the x-coordinates, the second column is a list of the corresponding y-coordinates, and the third column is a list of the corresponding z-coordinates. M cannot be empty unless the surface S is being redrawn. An easy way to create a matrix to be drawn which holds points from the graph of some function of two arguments, (i.e., a way to graph a given function), is to use the POINTS operator on the CROSS of two lists of coordinates to obtain the desired matrix.

If the surface name, S, is omitted, then a new unique name of the form 'Cn' is invented and is used as the surface name.

A copy of the matrix M is retained with the surface information. M may thus be altered or deleted without affecting the surface S. The function CURVEM(S) will return a copy of the matrix associated with the surface S.

When the surface name, S, does not name an already-existing surface then, if a 3D-window is not specified, the symbol "W3" will be used for W. In any case, if S is not an already existing surface and W is an empty 2D-window or an unknown entity, then W will be established as a 3D-window. When a 3D-window is created, it will assume the default image rectangle on the screen which is appropriate for the current display, unless the 3D-window is transformed from an empty 2D-window, in which case its image rectangle is maintained. Its clipping box is established to be all of computationally-accessible space.

When the surface name, S, is explicitly given, and has been previously drawn, then the 3D-DRAW statement is taken as a request to redraw the surface S using the parameters now specified. Parameters which are not given explicitly will retain their old values. This form of the 3D-DRAW command may thus be used to change 3D-windows, surface matrices, point-types, or line-types without disturbing other parameters of the curve.

If the specified 3D-window will hold only the single surface being currently drawn, then the surface will be scaled so that the box about all the surface objects, including the new object being drawn, is a cube of side length one.

If the surface being drawn is the second or later surface of the specified 3D-window, then scales of the surface are inherited from the most-recently drawn surface in the specified 3D-window.

A surface may be drawn with various symbols at the points in space specified by the surface matrix as specified by a point-type code listed below. Also, various lines may be drawn including lines

from the specified points to the floor plane (needles), or with direct connections from point to point (sequence), or with lines between every other pair of points (alternate), a rectangular network of lines connecting neighboring points in the x and y directions (net), or with removed (hidden) lines, or with bands of lines connecting neighboring points in the x-direction (hbands) or in the y-direction (vbands). All of these options are specified by an appropriate line-type value as shown below.

Hidden-quads are best understood by considering the drawing of a set of quads. When quads are drawn in the proper order away from the point-of-view, a hidden surface effect is achieved.

pointtype	symbol	meaning
0	NONE	null
1	DOT3D	small dot
2	CROSS3D	small 3D-crosses
"x"		generated character x (font 5)

linetype	symbol	meaning
0	NONE	no line
1	NET	net
2	HBAND	horizontal bands
4	VBAND	vertical bands
5	ALTERNATE	halfsequence
6	MARKER	marker skipping sequence
8	NEEDLES	needles
9		needles + net
16	HIDDEN	hidden-line net
32	SEQUENCE	sequence

The default line-type is NEEDLES, and the default point-type is NONE. Line-type MARKER is exactly analogous to line-type MARKER in 2D curves.

If the COLOR-clause, "NETCOLOR (P,Q)" is present, the top net of the surface is drawn in color P, and the bottom net of the surface is drawn in color Q.

The notions of "top" and "bottom" are particularly coupled with line-types VBAND, HBAND, and NET, but point-type symbols are always drawn with the top color and line-types NEEDLES, SEQUENCE, ALTERNATE and MARKER are always drawn with the bottom color. The default is NETCOLOR(1,1).

When the surface is drawn using a hidden-line algorithm, surface elements (either quadrilaterals or triangles) are shaded according to the magnitude of the angle that is formed between the view direction and the vector normal to the surface element. The magnitude of this angle ranges from 0 to 180 degrees. If there are $n \geq 2$ colors in the colorlist specified with the COLORLIST-clause as the integers $c_1, c_2, \ldots$ then the interval [0,180] is divided into n subintervals and color c_1 is used to shade surface elements with angles are within the first interval, c_2 to shade surface elements with angles within the second interval, etc. Note that the first colors in the colorlist correspond to smaller angles and the last colors in the colorlist correspond to larger angles.

Finally, using the interpretation above, the specified surface is drawn or redrawn in the left-handed 3-dimensional coordinate system of the specified 3D-window. Only the lines or parts of lines and points which are inside the clipping box of the 3D-window are drawn. A surface is thus established in the specified 3D-window with the name S.

When the specified surface has been established in the appropriate 3D-window, all the surfaces, plus any auxiliary box, floor, or axes are displayed in the appropriate image patch on the display screen. The combined set of surfaces is shown as though it were the result of a camera which recorded the scene from some point in space. The camera may be controlled by various 3D-VIEW CONTROL statements.

The parameters of the camera are: (1) its location in space (the camera can be located "inside" an object looking out, if desired!), (2) the direction the lens is pointing, (3) the angle of the view (i.e.

a narrow or wide angle lens may be simulated). Actually, changing the viewing angle is equivalent to scaling the object in the plane orthogonal to the viewing direction.

Initially, when the sole surface in a window is drawn, the camera is positioned heuristically looking at the surface with a viewing angle of 90 degrees (i.e. the camera clipping pyramid has apex angles of 90 degrees). The minimum and maximum x, y, and z values determine the positioning of the camera so that the surface is seen more or less centered in the associated 3D-image region. Moreover, unless an explicit SCALE-clause is given, the 3D-space is scaled to obtain "pleasing" cubical proportions; you can always reset the scales using 3D-VIEW CONTROL statements. Automatic camera positioning occurs only when a new 3D-window is established. The RESET command can be used to manually invoke such camera positioning.

A 3D-window contains no text except possibly plotting characters and axis labels. However a 3D-window can be overlaid with a 2D-window containing text strings.

28. The 3D-View Control Statement (with 3D-graphics extension package)

SYNTAX:

WINDOW3D Q |
CURVE3D C |
BLANK [V] |
UNBLANK [V] |
DOLLY A |
TRUCK A |
RAISE A |
TILT A |
PAN A |
TWIST A |
TRACK [AT A,B,C] |
TURN A |
ZOOM A |
ORIGIN |
RESET |
PERSPECTIVE | ORTHOGONAL |

[<SYSTEM | AXES | BOX | SURFACE>] XROTATE A |
[<SYSTEM | AXES | BOX | SURFACE>] YROTATE A |
[<SYSTEM | AXES | BOX | SURFACE>] ZROTATE A |
<XTRANSLATE | YTRANSLATE | ZTRANSLATE> A |
[SPACE] <XSCALE | YSCALE | ZSCALE | XYZSCALE> A |
FLOOR [AT A] | NO FLOOR |
AXES [AT A,B,C] | NO AXES |
GRID [N] | NO GRID |
TICS [N] | NO TICS |
BOXNUMBERS [X0 X1 Y0 Y1 Z0 Z1] [N] |
AXESNUMBERS [X0 X1 Y0 Y1 Z0 Z1] |
XNUMBERS [X0 X1] | NO XNUMBERS |
YNUMBERS [Y0 Y1] | NO XNUMBERS |
ZNUMBERS [Z0 Z1] | NO XNUMBERS |
XTITLE S |
YTITLE S |
ZTITLE S |
AUTOWBOX [X Y Z] |
BOX | NO BOX |
BOXCLIP X0 Y0 Z0 X1 Y1 Z1 | NO BOXCLIP |
WCOLOR C |
WFLOORCOLOR C |
WGRIDCOLOR C |
WBOXCOLOR C |
WBOXNUMBERCOLOR C |
WBOXTITLECOLOR C |
WAXESCOLOR C |
WAXISNUMBERCOLOR C |
WTICCOLOR C |

AUTOCBOX [X Y Z] |

POINTS [N] | NO POINTS |
HALFBOX | NO HALFBOX |
BOXTITLE | NO BOXTITLE |
SEQUENCE | NO SEQUENCE |
ALTERNATE | NO ALTERNATE |
NET | NO NET |
HBAND | NO HBAND |
VBAND | NO VBAND |
NEEDLES | NO NEEDLES |
HIDE | NO HIDE |
PTCOLOR C |
LCOLOR C |
TOPNETCOLOR C |
BOTNETCOLOR C |
COLORLIST L

EXAMPLES:

```
TWIST 15
SPACE ZSCALE .5 AXES XROTATE 30 IN W3
ZOOM 20
AXES AT 1,1,1
NO AXES
DOLLY 2 XROTATE 45 TRUCK -1.5 BOX
ZSCALE -1 TWIST 22.5/2 IN W31
ORIGIN XTRANSLATE 2 SURFACE XROTATE 90 IN S
NO BOX AXES FLOOR AT -1 RESET
```

3D-view control statements control (1) the camera of a 3D-window, or (2) the "structural format" of a 3D-window, involving scaling, the appearance of axes, boxes, floors, etc., or (3) the orientation, positioning, and scaling of a particular surface in a 3D-window. All these types of statements may be intermingled as desired in one input string which applies to a specified surface, or to all the surfaces in a specified 3D-window.

A surface is displayed as a camera would view it. All the surfaces in a 3D-space are bounded by a box ($xmin \le x \le xmax$; $ymin \le y \le ymax$; $zmin \le z \le zmax$) which measures $x1 = xmax - xmin$, $y1 = ymax - ymin$, and $z1 = zmax - zmin$ on the side parallel to the x, y, and z-axis, respectively. The center of the box is at the position ($xcenter = xmin + x1/2$, $ycenter = ymin + y1/2$, $zcenter = zmin + z1/2$). $xmin$, $xmax$, $ymin$, $ymax$, and $zmin$, $zmax$ are the given ranges of the data. This box is adjusted if necessary so that $x1 > 0$, $y1 > 0$, and $z1 > 0$.

Initially the camera is positioned at $(xcenter, ymax+y1, zmax)$, with its lens pointed in the direction which results from rotating a unit vector which is initially parallel to the z-axis by 105 degrees about the x-axis (right hand screw). The camera's initial viewing angle is 90 degrees. A number of commands may then be issued from the keyboard to vary the camera viewing position, camera viewing direction, 3D-curve orientation, and 3D-curve position.

The 3D-window to which the camera belongs is initially the default 3D-window W3. The 3D-window can be changed by the command "WINDOW3D Q" which makes Q the active 3D-window.

The camera parameters of the active 3D-window are controlled by the following commands. (All the various command words used in 3D-view commands may be abbreviated to any unique prefix string, e.g. PER may be used in place of PERSPECTIVE.

DOLLY K: the camera is moved K box units in the viewing direction.

TRUCK K: the camera is moved k box units horizontal to the viewing directions.

RAISE K: the camera is moved K box units vertical to the viewing direction.

PAN K: the viewing direction is rotated K degrees in the $x - z$ plane of the camera coordinate system. This is a left-hand screw rotation about the raise-vector.

TILT K: the viewing direction is rotated K degrees in the $y - z$ plane of the camera coordinate system. This is a right-hand screw rotation about the truck-vector.

TWIST K: the camera is right-hand-screw rotated K degrees about the viewing direction.

TRACK [AT A,B,C]: The camera is rotated so as to point to the point (A,B,C). If no "AT A,B,C"-clause is given then the camera is rotated to point to the box-enter of the box about the various surfaces.

TURN K: the camera is right-hand-screw rotated K degrees about the box-center of the window-box about the various surfaces in the plane that contains the camera and the box center and is normal to the raise vector. The camera's direction of view is modified to maintain the same deviation between the direction of view and the vector to the box center as existed before the turn operation.

ZOOM K: the viewing angle is set to K degrees. (Photographers specify this angle in terms of its cotangent, but we use the actual angle.)

ORIGIN: the camera is moved to 0,0,0 looking up the z-axis with 0 degrees of twist.

RESET: the camera is "automatically" positioned so as to provide the initially-established over-all view.

PERSPECTIVE: draws the 3D window with the $1/z$-perspective transformation.

ORTHOGONAL: draws the 3D window without the $1/z$-perspective trans-formation.

Except for the viewing angle, the various camera parameters are taken as incremental, so that DOLLY, TRUCK, RAISE, PAN, TILT, and TWIST arguments are added to the current values to obtain new quantities which determine the new picture to be displayed.

The arguments to DOLLY, TRUCK, and RAISE are relative box units. Suppose the box surrounding all the surface objects has an absolute size in scaled units of a by b by c. Then one unit of camera motion along the X-direction displaces the camera a scaled units, one unit of motion along the y-direction displaces the camera b scaled units, and one unit of motion along the z-direction displaces the camera c scaled units. One unit of transverse motion displaces the camera a certain combination of a, b, and c scaled units as determined by the Pythagorean theorem.

Actually a real camera has additional degrees of freedom. In particular, one can control the height of the lens relative to the film and the angular position of the film plane relative to the direction of the lens-film vector. These parameters are used to control perspective. Moreover an actual camera has a focus parameter which determines a plane or sphere of focus. All points on the plane of focus are seen as point images in the developed picture. Points which are off the plane of focus are seen as small disks whose radius is proportional to the distance they occur from the plane of focus. The intensity of light in such disks falls off from the center in a symmetric distribution. Thus a true camera causes more or less thick bands with fuzzy boundaries to occur where lines should be. For programming purposes it would suffice to map each point to a circle of appropriate radius. Actually however, the infinite depth of field provided by the simple graphic simulation in MLAB is to be preferred.

There are also clauses for modifying the particular 3D-window established as the current active 3D-window that holds various surfaces.

WINDOW3D Q: establish Q as the current 3D-window to which all further 3D view control commands apply

SPACE XSCALE K: x-coordinates of space are scaled by K. The camera and all surfaces in the active 3D-window are affected.

SPACE YSCALE K: y-coordinates of space are scaled by K. The camera and all surfaces in the active 3D-window are affected.

SPACE ZSCALE K: z-coordinates of space are scaled by K. The camera and all surfaces in the active 3D-window are affected.

SPACE XYZSCALE S: scale x, y, and z space coordinates by S.

AUTOWBOX X Y Z: scale all the surfaces with a common set of x, y, and z scales to make them jointly fit in a box of size X by Y by Z.

FLOOR [**AT K** :] A grid in the $x - y$ plane is drawn at Z=K. If K is not given, K is taken to be 0.

NO FLOOR: the floor grid, if any, disappears from the window.

AXES [**AT A,B,C** :] axes are drawn from the minimum to the maximum value of each coordinate variable, taken over all surfaces, and intersecting at (A,B,C). If no values for A,B,C are entered, (0,0,0) is implied.

NO AXES: the axes and labels disappear from the window.

XNUMBER [**X0 X1** :] the x-axis is marked and numbered at four places with the range X0 to X1.

YNUMBER [**Y0 Y1** :] the y-axis is marked and numbered at four places with the range Y0 to Y1.

ZNUMBER [**Z0 Z1** :] the z-axis is marked and numbered at four places with the range Z0 to Z1.

NO XNUMBER: x-axis marks and numbers disappear from the window.

NO YNUMBER: y-axis marks and numbers disappear from the window.

NO ZNUMBER: z-axis marks and numbers disappear from the window.

AXESNUMBER N: the x, y, and z axis are each marked at N places.

TICS [**K** :] draw 3D-tic marks on the axes

NO TICS: remove 3D tic-marks from the axes

BOX: draws a box around the surface objects.

NO BOX: the box disappears from the 3D-window.

HALFBOX: draw the three faces of the box which contain the vertex furthest from the camera.

NO HALFBOX: erase the three faces of the box which contain the vertex furthest from the camera.

BOXNUMBERS: draw numeric labels along the edges of the halfbox

NO BOXNUMBERS: remove numeric labels along the edges of the halfbox

GRID [**K** :] establish the floor and the halfbox sides to be $K \times K$ grids when drawn.

NO GRID: establish the floor and the halfbox sides to have no interior grid lines when drawn.

XTITLE S: make S the title-string for the x-axis

YTITLE S: make S the title-string for the y-axis

ZTITLE S: make S the title-string for the z-axis

WCOLOR S: set the color for drawing the 3D-window box, axes, axis labels, and tic marks to color S.

WFLOORCOLOR S: set the color for the 3D-window floor to color S.

WBOXCOLOR S: set the color for drawing the 3D-window box to color S.

WAXESCOLOR S: set the color for drawing the 3D-window axes to color S.

WBOXNUMBERCOLOR S: set the color for drawing the 3D-window halfbox numeric labels to color S.

WGRIDCOLOR S: set the color for drawing the floor and halfbox-side grids to color S.

WTICCOLOR S: set the color for drawing 3D-axis tic marks to color S.

WBOXTITLECOLOR: set the color for drawing the 3D-window halfbox to color S.

WAXESNUMBERCOLOR S: set the color for drawing the numeric labels on the x, y, and z-axes to color S.

Note: when space is scaled, the apparent camera directions will be affected. When the camera is at the origin looking at the point (1,0,0) and we have executed "SPACE XSCALE 3", then the command "PAN 45" will cause the camera to look at the point (1,1,0), but (1,1,0) is not on the 45 degree radial from the origin! Thus scaling individual surfaces may be preferable to scaling space.

Finally there are some statements for modifying various surfaces individually, including "moving them" in space before "looking through the camera" at the entire scene. These statements apply to the current active surface in the current active 3D window. The particular surface to which the above clauses apply is initially taken to be the first surface in the active 3D-window. It can be changed by the statement "CURVE3D C". When a surface is drawn it becomes the current active surface. The current active status active surface can be explicitly selected by the CURVE3D statement: CURVE3D C. For any given surface established as the current active surface, these statements are as follows:

XSCALE A: the surface is scaled by A in its x-coordinates.

YSCALE A: the surface is scaled by A in its y-coordinates.

ZSCALE A: the surface is scaled by A in its z-coordinates.

XYZSCALE S: x, y, and z surface coordinates are all scaled by S

AUTOCBOX X Y Z: scale the surface to fit in a box of size X $\times Y \times Z$

XTRANSLATE A: the surface is translated A units in the x direction.

YTRANSLATE A: the surface is translated A units in the y direction.

ZTRANSLATE A: the surface is translated A units in the z direction.

SYSTEM XROTATE A, XROTATE A: the surface is right-hand screw rotated A degrees about the x-axis of the underlying "system" coordinate system. Note the default rotation qualifier which is assumed when none is given is "SYSTEM".

SYSTEM YROTATE A, YROTATE A: the surface is right-hand screw rotated A degrees about the "system" y-axis.

SYSTEM ZROTATE A, ZROTATE A: the surface is right-hand screw rotated A degrees about the "system" z-axis.

AXES XROTATE A: the surface is right-hand screw rotated A degrees about the x-axis of the user-specified coordinate system defined by the axes command.

AXES YROTATE A: the surface is right-hand screw rotated A degrees about the user-specified y-axis.

AXES ZROTATE A: the surface is right-hand screw rotated A degrees about the user-specified z-axis.

BOX XROTATE A: the surface is right-hand screw rotated A degrees about the x-axis of the coordinate system centered in the box enclosing the entire group of objects. This box always has its sides parallel to the system coordinate axes.

BOX YROTATE A: the surface is right-hand screw rotated A degrees about the box y-axis.

BOX ZROTATE A: the surface is right-hand screw rotated A degrees about the box z-axis.

SURFACE XROTATE A: the surface is right-hand screw rotated A degrees about the x-axis of the coordinate system which is centered in the surface object itself, in the sense that the origin for this set of coordinates is the center of the smallest box which has its sides parallel to the system coordinate axes and which encloses the surface object.

SURFACE YROTATE A: the surface is right-hand screw rotated A degrees about the individual surface y-axis.

SURFACE ZROTATE A: the surface is right-hand screw rotated A degrees about the individual surface z-axis.

SEQUENCE: draw the curve by connecting successive points with solid line segments.

NO SEQUENCE: erase the curve by connecting successive points with solid line segments.

ALTERNATE: draw the curve by connecting successive points with alternating solid and invisible line segments.

NO ALTERNATE: erase the curve by connecting successive points with alternating solid and invisible line segments.

NET: draw the surface with a net. This can only be done on a regular rectangular lattice of points.

NO NET: erase the surface net.

HBAND: draw horizontal bands

NO HBAND: erase horizontal bands

VBAND: draw vertical bands

NO VBAND: erase vertical bands

NEEDLES: draw the surface with line segments emanating from the x-y plane to each point.

NO NEEDLES: erase the surface with line segments emanating from the x-y plane to each point.

HIDE: draw a hidden line view of a surface using depth-sort algorithm with quadrilaterals.

NO HIDE: erase a hidden line view of a surface using depth-sort algorithm with quadrilaterals.

POINTS [C]: draw 3D pointtype symbols of type C (0: none, 1: simple dots, 2: 3D-crosses, > 2: the corresponding ASCII character). If C is omitted, 2 is assumed.

PTCOLOR C: point symbols, if any, are drawn in color C.

LCOLOR S: line segments, if any, are drawn in color S

TOPNETCOLOR S: sets the color for the net viewed from above to colornumber S. All other surface drawing (i.e. net, needles, dots) is done in this color, too, by default.

BOTNETCOLOR S: sets the color of a surface's net viewed from below to color number S.

COLORLIST C1 C2 [C3...C16 :] change the current 3D-curve's list of colors to be used for shading surfaces. When the surface is drawn using a hidden line algorithm, surface elements (either quadrilaterals or triangles) are shaded according to the magnitude of the angle that is formed between the view direction and the vector normal to the surface element. The magnitude of this angle ranges from 0 to 180 degrees. If there are $n >= 2$ colors in the colorlist, the interval [0,180] is divided into n subintervals and color c_1 is used to shade surface elements with angles are within the first interval, c_2 to shade surface elements with angles within the second interval, etc. Note that the first arguments correspond to smaller angles and the last arguments correspond to larger angles.

BOXCLIP [X1, X2, Y1, Y2, Z1, Z2 :] the 3D-clipping box associated with the current active 3D-window is established as the box $[X1, X2] \times [Y1, Y2] \times [Z1, Z2]$. Also, boxclipping is enabled. If the arguments $X1, X2, Y1, Y2, Z1, Z2$ are missing, the enabled 3D-clipping box is the 3D-clipping box that was previously established for the current active 3D-window.

NO BOXCLIP: 3D-box clipping is disabled for the current active 3D-window

29. BLANK and UNBLANK Statements

SYNTAX:

BLANK U,...
UNBLANK U,...

EXAMPLES:

```
BLANK W.TOP
BLANK C1,A,B
UNBLANK W,C1,C2
UNBLANK W.TOP
```

U is a curve, axis, title, or window name.

The BLANK statement causes the specified window(s), curve(s), axis(s), or title(s) to be invisible when a VIEW statement is given for the specified windows(s) or for the window(s) containing the blanked objects.

Blanking a window does not blank the objects in the window, although it does make them invisible. If an object from a blanked window is moved or copied to an unblanked window, it will be blanked or unblanked according to its individual blank/unblank status, and not according to the status of the window from which it was obtained or to which it is copied.

The UNBLANK statement causes the window(s), curve(s), title(s), or axis(s) to be visible again when a VIEW command is given for the specified windows(s) or for the window(s) containing the blanked objects.

Unblanking a window does not unblank the objects in the window. If an object from an unblanked window is moved or copied to an blanked window, it will be blanked or unblanked according to its individual blank/unblank status, and not according to the status of the window from which it was obtained or to which it is copied. However, it will be invisible because the destination window is blanked.

30. VIEW and UNVIEW Statements

SYNTAX:

VIEW [W1,...]
UNVIEW

EXAMPLES:

```
VIEW W1,W2,W3,W4
VIEW
```

The VIEW statement is used to place pictures on the display screen. W1,... are the names of 2D or 3D windows to be displayed. If no window names are specified, all existing unblanked 2D or 3D windows will be displayed.

Under DOS, the text on the screen is not saved and the entire screen is reused for the display of the MLAB windows. To return to MLAB command mode, press the *ENTER* key, which will remove the pictures and display the MLAB command prompt (*).

Under DOS-Windows, or the Macintosh operating system, or under an X-windows-based or NeXT-Step Display-Postscript-based UNIX system, a special rectangular "MLAB Graphics Screen Patch" is created or re-created by the VIEW statement. This screen patch will remain on the screen until you exit MLAB or remove it explicitly with an UNVIEW statement.

31. PLOT Statement

SYNTAX:

PLOT [W1,...] [IN F]

EXAMPLES:

```
PLOT W1,W2,W3,W4 IN PLOTFILE
PLOT
PLOT IN PF1
PLOT W1
```

The plot statement is used to produce a plot-file containing all windows, if no arguments are given, or the specified windows, if an explicit argument list is given. The resulting file is in a form suitable for later plotting using a laser printer. F is a file name. W1 is the name of a window. If no IN-clause is present, a default file name of the form MLABPn.PS or MLABPn.LJ will be used, depending upon the value of the control string PLOTDEV. The integer n in the name is the least integer that makes the filename unique in the current directory.

The options for the string PLOTDEV are

- "PSL" (Postscript monochrome printer with landscape orientation)
- "PSCL" (Postscript color printer with landscape orientation)
- "LJ2L" (Hewlett-Packard Laser Jet II or higher printer with landscape orientation)

The MLAB pre-defined string constants PSL (= "PSL") PSCL; (= "PSCL"), and LJ2L (= "LJ2L") are provided and may be used instead of the explicit strings.

The default value of PLOTDEV is PSL. PLOTDEV is a string variable which may be assigned by the user to any of the above values, depending on whether the plotting device to be used employs the Postscript language in monochrome (PSL), the Postscript language in color (PSCL), or the Hewlett-Packard Laser Jet II command language (LJ2L).

If there is an IN F clause, the specified filename F will be used. If there already exists a file of that name, the user will be given the alternatives to replace the existing file, or to abort the command.

Remember that, for DOS or Windows PC's, all entered text, including the text T entered to specify the do-file name, will be converted to upper-case. For Unix systems or Macintoshes, this conversion to upper-case will only occur if the control variable *casesw* is set to 0. Since Unix and Macintosh systems permit lower-case file names, you may specify such names, either by setting *casesw* to 1, or by enclosing the filename in quotes.

The specified filename will be prefixed by the FILEDIR string. The name of the file which is created will be shown on the display.

The value of the MLAB control variable PSLINEW determines the width of the lines in a plot made with a Postscript device. The unit of the PSLINEW value is 1 point (1/72 inches); the default value is 2.

In a monochrome plot, colors used in an MLAB picture are plotted so that background colors are plotted as white (which will not show on the plot), and all other colors are plotted as black (which will show on the plot). Only dots and lines (including the lines which compose characters) are plotted; filled regions, i.e. the screen, frame, and image, are ignored.

Under DOS for the PC and compatibles, after exiting MLAB, a plot-file created by MLAB may be physically printed with a sufficiently capable printer by using the DOS PRINT command, or by

using a non-MLAB program which sends a file to a Postscript or Hewlett-Packard (HP) Laser Jet printer. Postscript files can also be printed from the DOS prompt after exiting MLAB, by using the facilities of the LAN software that interconnects the PC and the printer; if such software is present. The DOS command "COPY F COM2" will print the HP file F on an HP-capable printer attached to the serial port COM2.

Under Windows, a plot-file created by MLAB may be printed by opening a DOS Window, changing the current directory to the directory containing the MLAB plot file, and giving the DOS PRINT command with the name of the MLAB plot file (i.e. mlabp0.ps). Of course PostScript files can only be printed on a PostScript-capable printer, or by using a Postscript interpreter program with dumb printer output capabilities (Ghostscript might work.) Using DOS is necessary because there does not seem to be any direct tool for printing files provided in Windows (although a web-search will reveal many programs to print files, including Postscript files, that you can download.).

Under Macintosh, a plot-file created by MLAB may be printed on an attached PostScript-capable printer by treating the plot file as a "font" to be downloaded to the PostScript printer. This downloading is done by using the Laser Writer Font Utility program. This awkward method may be avoided if you have a utility program for printing PostScript files provided by the PostScript printer manufacturer.

The Laser Writer Font Utility program is provided by Apple. In order to use it you must double click on its icon and then use the menus it provides to specify the PostScript file which you wish to plot on the laser printer.

In all cases, a plot-file can be printed directly by using the MLAB PRINT FILE command.

You may of course delete MLAB-generated plot files at any time, either within DOS or Windows MLAB using a DELETE FILE command, or from outside MLAB using operating system commands. You are free to change your mind about having them plotted.

Plot-files created by MLAB may also be incorporated in documents produced by word processing programs which support embedded graphics. See the instructions for your word processor for details. In particular, plot-files created in MLAB using PLOTDEV = PSL (Postscript) use the "encapsulated Postscript" format, which is required for WordPerfect, and is acceptable for Microsoft Word. It is also easy to link an MLAB plot-file in a .tex file.

How to put MLAB PLOT files in MSWord documents under Windows
In MLAB, as discussed above, you can draw and view a picture, and then use the PLOT command to "save" this picture as a Postscript plot file in your working directory under the name mlabpN.ps, where N is a one or two digit number one greater than the numbers already thus used in .ps files in this directory.

To put an MLAB plot file named mlabp0.ps in a Microsoft Windows Word (MSWord) document, so that it appears as a graph, follow these directions:

First you must modify the MLAB .ps file, mlabp0.ps, to change it into a "portrait-layout" picture, instead of the landscape-layout used by default in MLAB. This is done as follows.

1. Start the MS Word program
2. Open the file mlabp0.ps as a Text File
3. Move the cursor to the fourth line which begins with "%%BoundingBox". Then swap the numbers 593 and 773 that appear there; the line should then read

   %%BoundingBox 19 19 773 593

   (This tells the Postscript interpreter that this picture is in a box with $19 \le x \le 773$ and $19 \le y \le 593$ in "point" units (1 point= 1/72 inch). This size can be modified in your MS-Word file, but the aspect ratio $(773-18)/(593-18)$ will be honored.)

4. Find the line (it is about the 20th line) that reads: "612 0 translate". Insert a percent sign (%) which serves as a "comment" symbol at the beginning of that line AND the immediately-next line. These two lines should then appear as:
   %612 0 translate
   %90 rotate
5. Save this modified file mlabp0.ps as a text file with the name mlabp0.eps (You can now delete the original file mlabp0.ps)

Now, the file mlabp0.eps is ready to be embedded in your MS-Word "document", as follows (These directions apply for earlier versions of MS-Word; for later versions, you do not need to, and may not, create a "frame".)

6. Open the file ("document") in which you would like the MLAB picture in mlabp0.eps to appear
7. Make sure you are viewing the document in "page layout form". Do this as follows. From the [View] menu, select [Page Layout]. This will then allow you to view, resize, and/or reposition the frame, which is a rectangular region in the document that will contain the picture to be embedded.
8. Position the cursor where you want the MLAB graph to appear and create two new paragraphs there; one is for the figure, and the other is for its caption text. These paragraphs are created by striking the Enter-key twice, and they are denoted on the screen by "paragraph"-symbols. This ensures that the figure stays close to the point in the text at which it is first mentioned.
9. "Highlight" both paragraphs by dragging the mouse over them with the left mouse-button held down.
10. From the [Insert] menu, choose [Frame]. This will dreate a frame for the two highlighted paragraphs which will be replaced by the figure and its caption test in that frame; the objects in this frame can later be moved as a unit.
11. Place the text-cursor in the bottom paragraph of the two you have created (by using the keyboard or by placing the mouse cursor there and 'clicking'), and then type in your figure caption text.
12. Now "import" the figure as follows:
    (1)Place the text-cursor at the first paragraph of the two inside your figure frame (in the same way you did above).
    (2)From the [Insert] menu, choose [Picture]. (This will cause a picture-insertion control panel to appear.)
    (3)Specify the file named mlabp0.eps as the picture-file within the showing control panel.
    (4)Specify that the picture-file is to be "copied" into your document, not "linked" to.
    (5)Click 'Insert' in the showing control panel.

    Note: the figure does not appear in the document window, only some descriptive text will show, but it WILL appear on the printed page when the document is printed.
13. To resize the imported graphic, do as follows. Highlight the graphic by clicking on it. Be sure to highlight the actual graphic, and not its frame. Hold the mouse-button down and drag one of the corner "handles". This resizes the figure proportionally. Note: dragging a middle handle will resize the image disproportionally, and may distort the graphic. If you wish to resize proportionally, be sure to choose one of the corner handles.

Now you can continue editing your document and, at some point, you should save it. Final note: before printing your document, make *sure* (at the Windows-level) that the Windows Control Panel for the printer attached to your computer has a PostScript printer driver specified (it must say "Postscript"); otherwise, the figure will NOT appear in the printed document.

32. DELETE Statement

SYNTAX:

DELETE U,...
DELETE FILE F
DEL U,...

EXAMPLES:

```
DEL W
DELETE X,Y,F,G DIFF T
DELETE M ROW (1,3:7,19)
DELETE FILE MLABPSP.4
```

The statement DELETE U will caused the entity named U to be removed, and the space occupied by U to be reclaimed.

DELETE may be abbreviated DEL. U may be the name of a scalar, matrix, function, curve, axis, title, window, initial, or conset. U may be also be a submatrix expression.

If a window is deleted, then the associated frame, image, and all curves, titles, and axes in the window will be deleted also.

Individual curves, titles, and axes may be deleted. To delete titles and axes given default names, these names must be known In particular, the statements which create objects with specific default names are:

```
TOP TITLE ... IN W1        W1.TOP
LEFT TITLE ... IN W1       W1.LEFT
RIGHT TITLE ... IN W1      W1.RIGHT
BOTTOM TITLE ... IN W1     W1.BOTTOM
XAXIS ... IN W1            W1.XAXIS
YAXIS ... IN W1            W1.YAXIS
```

If no window name was specified in an IN-clause, the default window name W, will have been used, and the corresponding names will be W.TOP, W.LEFT, etc.

When a curve, axis or title without a name, is drawn, a default name is assigned, of the form C#, A#, or T#, where # is a unique integer starting with 0. Such names are useful because these objects may be deleted by name only.

When a function is deleted, all of its derivatives are deleted also. If any of these functions appear in the body of some other function, that function is no longer correctly defined. In general, if any symbol which is used in a function definition is deleted, the function is no longer correctly defined; and, moreover, if the symbol is re-established, the function may, upon rare occasions, remain incorrectly defined. In this rare event, each afflicted function must be redefined. Deleting an initial condition does not delete its associated differential equation-defined function.

The statement DELETE M ROW N will cause the rows specified by N to be deleted from the matrix M. Similarly, specific columns may be deleted with DELETE M COL N, and both rows and columns may be deleted simultaneously with DELETE M COL N ROW P or DELETE M ROW P COL N.

The DELETE FILE F, ... statement is used to delete one or more files. F is the name of a file, either given explicitly or as a string enclosed in double quotes (”). The DOS wild-card notation, e.g. "*.do", may be used only when the string is enclosed in double quotes. The FILEDIR string is prefixed to filenames referenced by this statement.

33. SAVE, SSAVE, RESAVE, and RESSAVE Statements

SYNTAX:

SAVE [U,...] IN F
SSAVE [U,...] IN F
RESAVE [U,...] IN F
RESSAVE [U,...] IN F

EXAMPLES:

```
SSAVE IN PIC1
RESAVE DATA, F DIFF A IN FDATA
SAVE A,B,F,M,CONST IN BAG1
RESSAVE IN SFILE
```

The SAVE, SSAVE, RESAVE and RESSAVE statements are used to place a copy of an item or items in a file on disk. The optional list U, ... holds the name(s) of one or more scalars, matrices, functions, initial conditions, constraints, or windows (but not curves, titles or axes, which are saved when the windows to which they belong are saved).

When an SSAVE or RESSAVE statement is given without a specified list of objects to be saved, then every non-constant object is saved. When a SAVE or RESAVE statement is given without a specified list of objects to be saved, then only user-created objects are saved, and the MLAB control variables, such as FILEDIR, MAXITER, NFORMAT, etc. are not saved.

The IN-clause IN F is mandatory. F will be used as a disk file name. It will be prefixed with the FILEDIR string, and .sav (in lower-case) will be appended to the name if no extension is given.

The RESAVE and RESSAVE statements unconditionally replace an existing .sav file without asking for user confirmation. The SAVE and SSAVE statements specify that the user will be prompted if the save file already exists. If it does exist, the user may specify that the existing file be replaced (by typing "R") or that the command be aborted (by typing *ENTER*).

The USE and REUSE statements may be used to retrieve all the items saved in a previously constructed .sav file. It is important to remember that the items stored in the .sav file are a separate copy of the items with the same name inside MLAB. The contents of a .sav file will not change unless another SAVE statement is done to overwrite it, and the information in the .sav file will not be known to MLAB until a USE or REUSE statement is done to read it in.

Note .sav files created by MLAB are a mixture of text and binary information and cannot be listed or edited by text editors.

In DOS, it is possible to save files on a floppy disk by using the FILEDIR string. For example, to save a file on the disk in drive A, set FILEDIR to "A:" i.e. FILEDIR = "A:". Then the statement SAVE A,B,C IN F would save A, B, and C in the file named A:F.sav on a floppy disk in drive A:.

Under the DOS operating system, file names may have no more than eight character and file extensions may have no more than three characters. Longer file names or extensions are truncated by the operating system(without notice) to eight characters or three characters, respectively. For DOS or Windows PC's, all entered text, including the text T entered to specify the sav-file name, will be converted to upper-case. For Unix systems or Macintoshes, this conversion to upper-case will only occur if the control variable *casesw* is set to 0. Since Unix and Macintosh systems permit lower-case file names, you may need to specify such names, either by setting *casesw* to 1, or by enclosing the filename in quotes.

Only scalars, matrices, strings, functions, constraints, initial conditions, and windows (with their current curves, titles, and axes) may be saved.

The SSAVE and RESSAVE statements may be used to preserve the complete state of the current MLAB session in the .sav file. When this file is used, this state will be restored. It is often awkward to save the MLAB control variables, such as FILEDIR, so usually a SAVE or RESAVE statement is more appropriate.

34. USE and REUSE Statements

SYNTAX:

USE U
REUSE U

EXAMPLE:

```
USE GSAVE
USE DW
```

U is the name of a file previously created with the SAVE command. The name U is implicitly prefixed with the FILEDIR string. Note that the extension .sav need not be explicitly specified. It is always implicitly assumed.

The statements USE U and REUSE U will both cause a copy of the item or items which were previously saved using one of the SAVE statements, with file-name U.sav, to be made available to the current MLAB session.

The USE form of this statement avoids name conflicts with existing objects by appending the letter "A" onto the conflicting name (as many times as necessary to yield a unique name).

The REUSE form of this statement does not avoid name conflicts with existing objects. If an item with the same name as an existing entity is used, say a matrix, then the existing entity will be deleted, and replaced by the item being used (a message is given). Note, when a function is used, it replaces any existing version with the same name, and the derivatives of that existing version if any, are deleted!

When a window is used, all curves, titles, and axes in that window are also brought in.

35. FOR Statement

SYNTAX:

FOR A = M DO {S}

EXAMPLES:

```
FOR I=1:10 DO {M COL I=DATA ROW I:NROWS(DATA):10}
FOR I=1:NROWS(Q) DO {X=IF X>Q[I,2] THEN X ELSE Q(I,2)}
FOR J=1:100 DO (IF GCD(J,X)>1 THEN {J_101;BREAK} ELSE  {Y=Y+1; M[Y]=X}
FOR J=1:4 DO {DO S}
```

A is a scalar or undefined name. M is a matrix expression. S may be any MLAB statement or sequence of statements separated by semicolons, including FOR statements or DO statements.

FOR statements allow you to build for-loops. for-loops may be nested indefinitely. The for-loop variable may occur in any context within the body of the loop. It may even be an implicit function argument.

If an error is detected during the execution of a FOR statement the entire nest of loops will be terminated.

Braces () delimit the body of a FOR statement. The statements which form the body of a FOR-statement must be contained in braces. Recall that braces, brackets and parentheses are not distinguished in MLAB, except in the constraints statement.

A disk do-file or a string do-file may be executed within the body of a for-loop.

A for-loop variable may be changed within the loop, but this change does not affect subsequent iterations. However, if the matrix of index values is changed, the changes will be honored in successive iterations. A for-loop may be interrupted using the BREAK and DONE statements described in the next section.

36. BREAK and DONE Statements

SYNTAX:

BREAK
DONE

EXAMPLES:

```
FOR I = 1:K DO {IF I = M THEN BREAK ELSE J = J+1}
FOR I = 1:K DO {FOR J = 1:L DO {IF I > J THEN BREAK}}
FOR I = 1:K DO {FOR J = 1:L DO {IF I > J THEN DONE}}
```

The statement BREAK, appearing in the body of a FOR statement, will abort the execution of the current loop, but not of any other FOR statements in which that FOR statement is nested.

The BREAK statement is also used to exit from do-files (and string DO statements as well). It is only the innermost, i.e. the currently-controlling, do-file or FOR-loop which is interrupted, however.

To break all the way out of a nest of FOR-statements and do- statements and return to MLAB top level with a prompt for keyboard input, the DONE statement is used.

37. IF Statement

SYNTAX:

IF E THEN <S1| {S1;...} > [ELSE <S2 | {S2;...} >]

EXAMPLES:

```
IF X=1 THEN DO FA ELSE TYPE ‘‘X:’’,X
IF I=NROWS(A) OR A[I]<A[I+1] THEN {B[I+1] = A[I]; B[I] = A[I+1]} ELSE I = 1
IF A<0 THEN {A = 1; M = X(1); I = 1} ELSE IF M<X(I) THEN M = X(I) ELSE I = I+1
IF F(X)=0 THEN H = 1
```

E is a scalar expression usually constructed using the Boolean and/or relational operators, and S1 and S2 are arbitrary statements (or groups of statements enclosed in braces).

The ELSE-clause need not be present. If $E \neq 0$, then the THEN clause of the statement (or group of statements) is executed, otherwise the ELSE-clause statement (or group of statements) is executed if it exists. If $E = 0$ and there is no ELSE-clause, nothing is done in response to this statement.

Note that the IF statement is not the same as the conditional expression, since the THEN and ELSE clauses contain statements or statement lists rather than scalar or matrix expressions.

38. DO Statement

SYNTAX:

DO T
DO “DFILE.TXT”
DO a.do
DO CON
DO KLINE

EXAMPLES:

```
DO DOFILE1
S =  ‘‘TYPE 5; IF A>B THEN TYPE A ELSE TYPE B’’; DO S
DO A
```

The DO statement is used to execute a collection of one or more MLAB commands. The name T specifies text which contains a list of MLAB statements to be executed. The object T may be interpreted in any of the following ways.

• T is an identifier specifying the name of a do-file on disk, which will automatically be prefixed by the DOFILEDIR string, if it has been defined, or the FILEDIR string if it has been defined and the DOFILEDIR string has not been defined. The FILEDIR or DOFILEDIR string will be used to define a path to the logical disk drive, directory, and subdirectory containing the do-file. A do-file is a so-called script of MLAB commands.

The FILEDIR string permits files, including do-files, to be read or written in directories different from the current directory. The DOFILEDIR string permits use of just do-files in directories different from the current directory. It is often convenient that the input, plot, print, save and log files be associated with project directories, while the do-files which use or produce them reside in a separate directory, since these project-specific files will typically have the same names, and would in that case overwrite existing files if they were always in the same directory.

If the name T contains a non-null extension, it will be used. If the name T does not contain any extension, .do (in lower-case) will be assumed. If no filename T.do is found, the filename T will be used. If neither is present, an error message will be produced and MLAB will return to accepting commands from the keyboard.

• T is the name of a string variable. The characters of the string are interpreted as an MLAB command or sequence of commands separated by semicolons (;). These commands are then executed.

• T is a constant string contained in double quotes ("") which is interpreted as the name of the do-file on disk to be executed. No .do extension will be added, but the FILEDIR string will be prefixed and the resulting file will be executed. For DOS or Windows PC’s, all entered text, including the text T entered to specify the do-file name, will be converted to upper-case. For Unix systems or Macintoshes, this conversion to upper-case will only occur if the control variable *casesw* is set to 0. Since Unix and Macintosh systems permit lower-case file names, you may need to specify such names, either by setting *casesw* to 1, or by enclosing the filename in quotes.

• T is the name CON: which means “console”, and refers to the keyboard of the DOS computer on which MLAB is running. The text string of one or more lines containing the MLAB statements to be executed as a do-file will be requested to be entered directly into the computer by typing on the keyboard. Input provided in response to DO CON: is terminated by typing *CTRL-Z*, i.e. the control key and the z-key simultaneously) and then the *ENTER* key. Use of CON: may only be applicable for MLAB running on DOS computers.

• T is the name KLINE. KLINE stands for keyboard line; DO KLINE means “accept a single line of commands from the keyboard”. The text string containing the MLAB statements to be executed

as a do-file will be entered by typing a single line on the keyboard. Input provided in response to do kline is terminated with the *ENTER*-key.

If T is the name of a string variable, it will be used in preference to a do-file with the same name.

The specified string or file must consist of legal MLAB statements separated by semicolons or newline characters generated by the enter key. Statements in do-files which extend to several lines must have a back-slash "" at the end of each line. The statements will be executed as if they had been typed in one by one in sequence. There is one difference, however. Ordinarily, statements are echoed to the display screen when they are executed. Within do-files, however, screen echoing is suppressed. This echoing can be restored by setting the scalar ECHODO to 1. Echoing may be disabled again by setting ECHODO to 0. The options for ECHODO are as follows:

0: don't echo the statements from the do-file, either to the screen or to the log file.
1: echo the statements from the do-file to the screen, but not to the log file.
2: don't echo the statements from the do-file to the screen, but do show them in the log file.
3: echo the statements from the do-file, both to the screen and to the log file.

Since an ECHODO assignment is an ordinary assignment statement, it may appear within the do-file itself, so a do-file may regulate its own command echoing.

Note that a comment (i.e. a quoted string of text) is a legal statement, which when executed does nothing. Comments may appear in do-files, separated from neighboring statements by semicolons.

do-files may be constructed or modified inside of MLAB using the EDIT FILE facility within MLAB, or outside of MLAB using a text editor. do-files may also be constructed within MLAB by means of string variables and edited using the internal editor. Thus, suppose that the string variable A was defined using the statement:

A = "M = 1:5; FOR I = M&M DO DO FILE1"

Then the statement DO A will cause the successive statements M = 1:5 and FOR I = M&M DO DO FILE1 to be performed.

Note for DOS PC's: if a do-file contains the command DO CON, this will allow immediately-entered user-specified statements to be interjected at that point in the do-file. In particular, this device can be used to pause. Input to MLAB in response to the DO CON statement must be terminated by a *CTRL-Z* (hold down the control key while pressing the z key) followed by a pressing the *ENTER* key. Messages can be typed to the user before such a pause occurs by including statements of the form TYPE S where S is a text-string enclosed in quotes. CON means "console". However, this method is not portable. DO KLINE is preferable.

The statement DONE, when executed within a do-file, will abort the execution of all do-files and return to the MLAB command interpreter, where you will be prompted with a *.

The statement BREAK, when executed within a do-file, will abort execution of that do-file and return to the previous do-file, if any. It is only the innermost, i.e. currently controlling do-file or for-loop which is aborted, however. To break all the way out of a nest of do-files and/or for statements, the DONE statement may be used.

The MLAB control variable DOSTEP may be set to a non-zero value k by an assignment statement, e.g. DOSTEP = k, to cause MLAB to pause after every k lines are read from the do-file and wait for the *ENTER* key to be pressed before executing the statement or statements on that line of the do-file.

When MLAB starts running, it will look in the current directory for a do-file called mlabinit.do. If such a file is found, it will be executed. This is a means by which various desired initializations may be done automatically.

Typing "Q" while a do-file is executing will cause MLAB to abort do-file execution at the next statement, and go to top-level.

39. EDIT Statement

SYNTAX:

EDIT Z
EDIT
EDIT FILE foo.do

EXAMPLES:

```
EDIT F      (a function)
EDIT S      (a string)
EDIT Q      (a constraint set)
EDIT T      (a title)
EDIT        (the previous command)
EDIT FILE A (the file named A)
```

There are two editing facilities in MLAB. One is a single "line" editing facility, and the other is a file-editing facility. The EDIT command invokes the single-line editor, and the EDIT FILE command invokes the file editor. Currently, the EDIT FILE command is only functional on DOS, Windows, and Macintosh computers. On other systems that provide multi-processing (i.e. Unix/Linux), editing of files may be done using any desired editor program without exiting MLAB.

The EDIT command may be used to modify functions, strings, constraints, and titles, as well as the most-recently-read command. If present, Z is the name of a function, a string, a set of constraints, or a title. The editing facility is also is used to repair incorrect statements. When the EDIT command is given, the character string which constitutes the text of the command used to create the edited object is written to the display, and the cursor is placed at the beginning of the string. In the case of an incorrect statement, an implicit edit statement is executed to place that statement in the single-line editor for editing.

The single-line editor operates on a string which is an MLAB command string defining the object subject to editing. The user may change the name of the object as well as its contents; this results in the creation of a new object. The editor is thus convenient for creating two or more similar, but not quite identical, objects. For example, additional qualifying clauses may be included.

The cursor may be moved about as desired using the arrow keys ($\leftarrow, \rightarrow, \uparrow, \downarrow$). The cursor may be placed at the end of the string using the *end* key, and at the begining of the string using the *home* key.

Initially, the single line editor is in insert mode, which means that each typed character will be inserted to the left of the cursor, which will move over one space. The mode may be toggled between insert mode and overwrite mode using the INS key (*insert* key). In overwrite mode, each typed character will replace the character at the cursor, which will move over one space.

Characters may be deleted using the *del* key (delete) and the $\leftarrow$ (BACK-ARROW key). The *del* key deletes the character at the cursor, and the cursor does not move. The $\leftarrow$ key deletes the character to the left of the cursor, and the cursor moves one character to the left.

The edit process may be aborted at any time by using the *esc* key. The edit process is abandoned and MLAB returns to top level giving an input prompt.

The edited string is accepted and the edit process is completed by using the *ENTER* key. The edited string is now interpreted as an MLAB statement sequence and executed as if it had been typed in directly.

The previous statement in an MLAB session may be edited by typing EDIT with no argument. The editor is also entered when an MLAB statement has been determined to contain a syntactic

error. The offending statement may be modified as described above, and either aborted (*esc* key) or re-executed (*ENTER* key).

Examples: Suppose we had a function $f(x) = ae^{bx}$ and wished to define a new function $g(x) = ae^{bx} + c$. The MLAB statement EDIT F would enter the editor with the string:

```
FCT F(X) = A*EXP(B*X)
```

Using the arrow keys, we would move the cursor to the name "F", delete it and insert the new name "G". Then we would use the end key to move to the end of the string, and insert the characters "+C". Finally, the new function would be created after the *ENTER* key was pressed.

Suppose we had incorrectly typed a statement, e.g.

DRAW M COL 1:2 POINTYPE CIRCLE

(The keyword POINTTYPE has been mispelled.). MLAB would identify that this statement is incorrect and place the above string in the editor. To correct it, we use the cursor keys to move to the T and insert an additional T, thus correcting the statement. The corrected statement would be executed after the *ENTER* key was pressed.

The EDIT FILE command (in DOS, Windows and Mac-OS systems only)

The EDIT FILE command can be used in DOS, Windows, or Macintosh systems to edit any ASCII (standard text) file. It is particularly useful in the case of do-files, containing a series of MLAB commands, because it allows you to create, inspect and modify the series of commands in the do-file without having to terminate an on-going MLAB session.

For Unix or Linux systems, it is not necessary for MLAB to contain integrated file editing features. For these systems it is possible to invoke a concurrently running editor program completely independent of MLAB. Thus you might run an editor in one window while you are running MLAB in another and switch focus between the two windows as you choose.

The optional argument FILENAME can be a quoted string constant, or an MLAB identifier. If it is a string constant or an MLAB identifier that references a string, the string is taken as the name of the file to edit. In the case of an MLAB identifier that does not represent a string, the identifier itself is taken as the name of the file to edit.

If no extension is present in the obtained file name, the extension .do is automatically added. If no extension is desired, FILENAME must be presented as a string terminating with ".". For example, the command EDIT FILE SCORES references the file SCORES.do by default, while EDIT FILE SCORES. references SCORES. If FILENAME is omitted, MLAB uses the name TEMP.DO.

MLAB automatically prefixes the path specified in FILEDIR to FILENAME unless FILENAME begins with either a "\" or ".". If the user wishes to include a drive letter, he must include the drive letter in FILEDIR or at the beginning of FILENAME if FILEDIR is empty. If no path is specified in FILEDIR or FILENAME, MLAB references only the current directory.

When the EDIT FILE command is issued to DOS MLAB, the screen is cleared and the MLAB file editor is activated. If the referenced file exists, it appears on the screen ready to be edited; otherwise you are asked if you wish to create a new file.

The standard control keys (arrow keys, page up, page down, home, and end) operate as expected in the editor. Special control key combinations for searching, replacing, cutting, pasting and other activities, may listed by typing F1. To return to the on-going MLAB session type either F3 (save file and return) or *CTRL-F3* (do not save but return).

The following is a complete list of DOS editing commands with accompanying keystrokes:

COMMAND DESCRIPTION	KEYSTROKE
Find a String (All,Forward,Backward)	F5
Change a String (All,Range,Block)	F6
Change a String With Verify (All,Range,Block)	Shift F6
Highlight/Stop Highlighting a Line Block	F8
Highlight/Stop Highlighting a Character Block	F7
Copy a Highlighed Block of Text	Alt C
Move a Highlighted Block of Text	Alt M
Delete a Highlighted Block of Text	Alt D
Delete Current Character	Del
Delete Current Line	Shift F9
Insert a Line After the Current Line	F9
Insert a Line Before the Current Line	Ctrl F5
Join Next Line At The End Of The Current Line	Alt J
Split Current Line At Current Position	Alt S
Toggle Between Insert/Replace Mode	Ins
Backspace Or Delete Previous Character	Backspace
Carriage Return Or Next Line	<CR>
Page Down	PgDn
Page Up	PgUp
Position At the Beginning of the File	Ctrl Home
Position At the End of The File	Ctrl End
Position At the Beginning of the Line	Home
Position At the End of The Line	End
Position Down One Line	Cursor-Down
Position On the Next Character	Cursor-Right
Position On the Previous Character	Cursor-Left
Position On the Next Word	Ctrl ->
Position On the Previous Word	Ctrl <-
Position Up One Line	Cursor-Up
Tab Toward Right	Tab
Tab Toward Left	Shift Tab
Quit the Current File	Ctrl F3
Save the Current File and Continue Editing	Shift F3
Save the Current File and Quit the file	F3
Save the Current File, Delete the backup file, and Quit	F2
Memory Status Display	Shift F5
Help Screens	F1

When the EDIT FILE command is issued to Windows MLAB, MLAB execution is suspended and the Microsoft NOTEPAD program starts. If the referenced file exists, it is displayed in a separate text window ready to be edited; otherwise NOTEPAD asks if you wish to create a new file before an editing window is established. When you are done editing with NOTEPAD, you may select the FILE menu and click SAVE to save your edited file, and then select the FILE menu again and click EXIT to return to MLAB.

40. HELP Statement

SYNTAX:

HELP [S]

EXAMPLES:

```
HELP
HELP SIN
HELP ‘‘EXIT’’
HELP ‘‘+’’
```

The HELP statement is used to invoke the MLAB online help system. The help system provides information from the MLAB Reference Manual. Typing HELP with no argument brings in the top level screen. From the top level, the user can “navigate”, often from most general down several levels to specific MLAB functions, statements, operators or constants.

Typing HELP S, where S is any topic expressed as a string enclosed in double quotes, brings up information on the topic. If the topic is not included in the help system, the user is offered a series of alternative choices that begin with the same character.

A numbered list of choices appears at the end of each topic. The first 3 choices, 1 (top level), 2 (last topic), and 3 (quit), are always available. They are followed by one or more additional choices that relate to the topic. To move next to a specified choice, the user can type either the name of the choice or its associated number. In addition, the user is free to enter the name of any other topic. The user can exit from the help system and return to the MLAB command line, by selecting either 3 (quit) as a choice or by continually entering “2” until the system cycles back through all the previous topics.

Short topics can be displayed in their entirety on one screen. In the case of longer topics, that require more than one screen for their complete display, at the end of each screen or page of information, the user is given the option of pressing a control key. Depending upon location, this control key can be Home, End, PageUp or PageDown. Home brings up the first page of the topic, while End brings up the last page, which contains the list of choices for moving on to new topics. PageUp moves back 1 page, while PageDown moves forward 1 page. While moving within a topic, all non-control keys operate in the same manner as PageDown, i.e. they move down 1 page.

When MLAB is run under X-Windows, the HELP system does not provide Home, End and PageUp options. Any key pressed will move down long topics. Standard window scrolling approaches can be used to scoll around within a topic.

41. RESET Statement

SYNTAX:

RESET

EXAMPLE:

```
RESET
```

All existing data objects (matrices, scalars, functions, windows, constraints, etc.) are deleted. This statement provides a way to restart MLAB without exiting the program. There are some differences between the initial state of MLAB and the state after a RESET. In particular, some memory will not be freed, and the various MLAB control variables will not be reinitialized.

42. EXIT Statement

SYNTAX:

EXIT

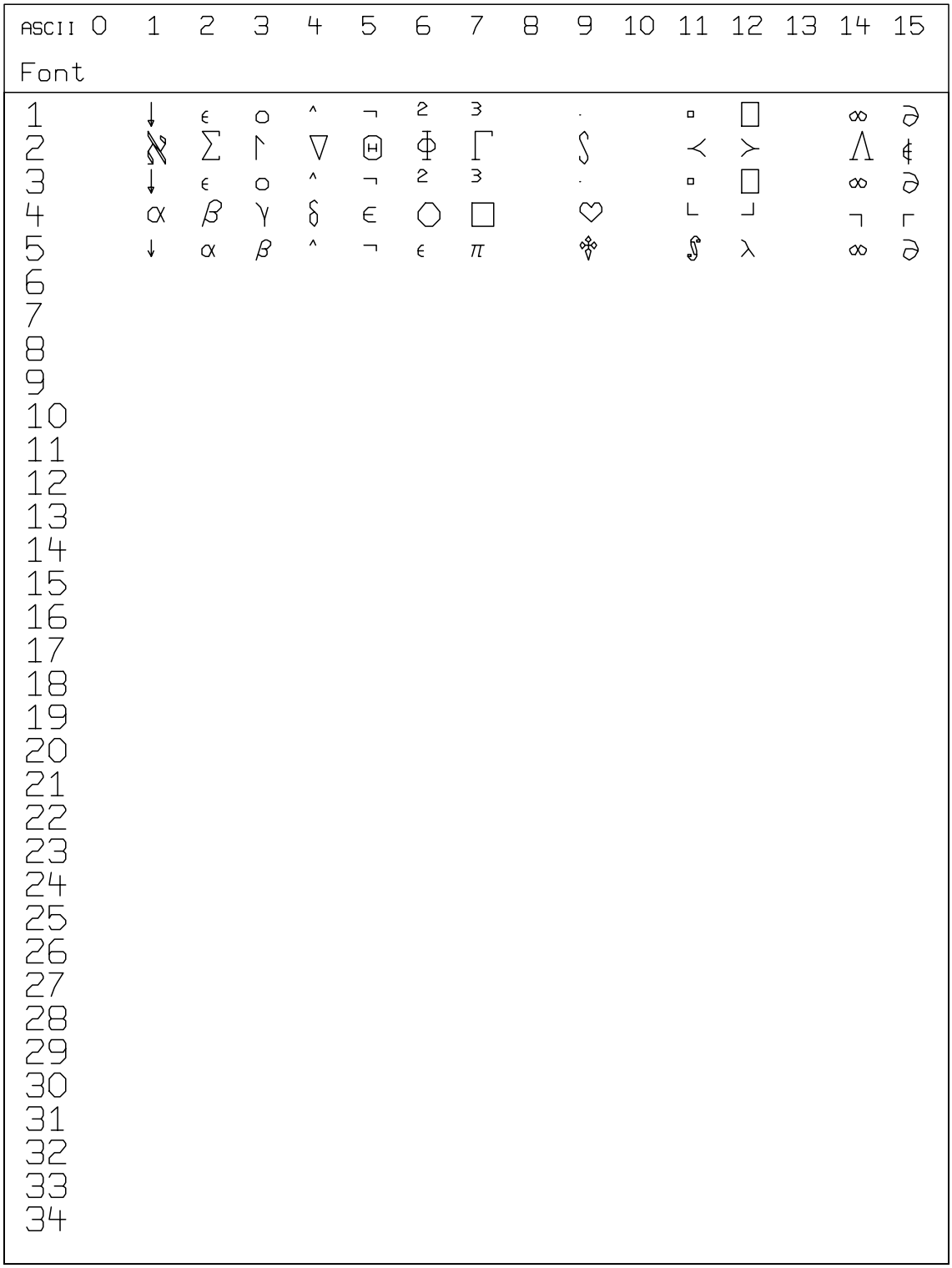
EXAMPLE:

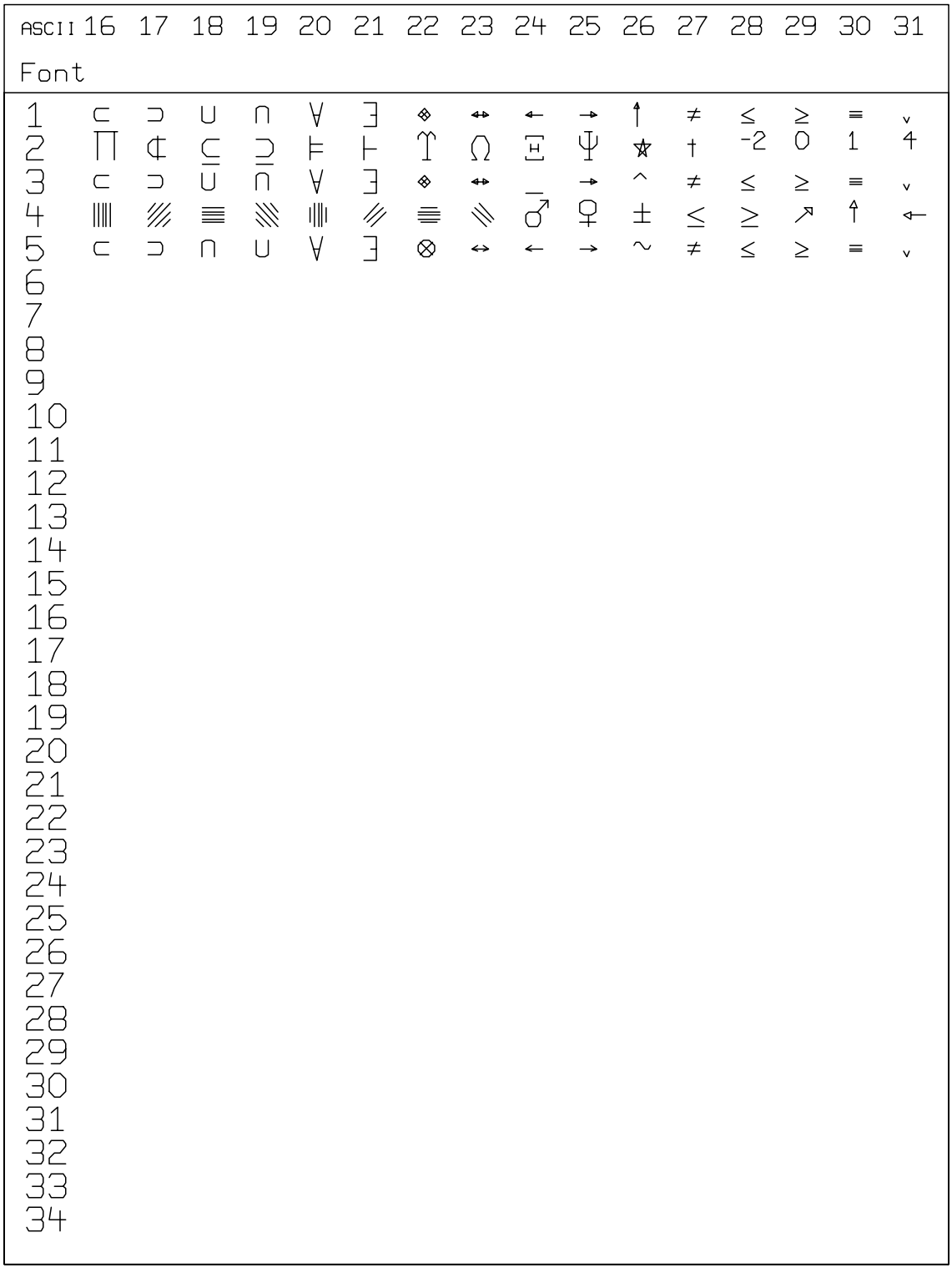
```
EXIT
```

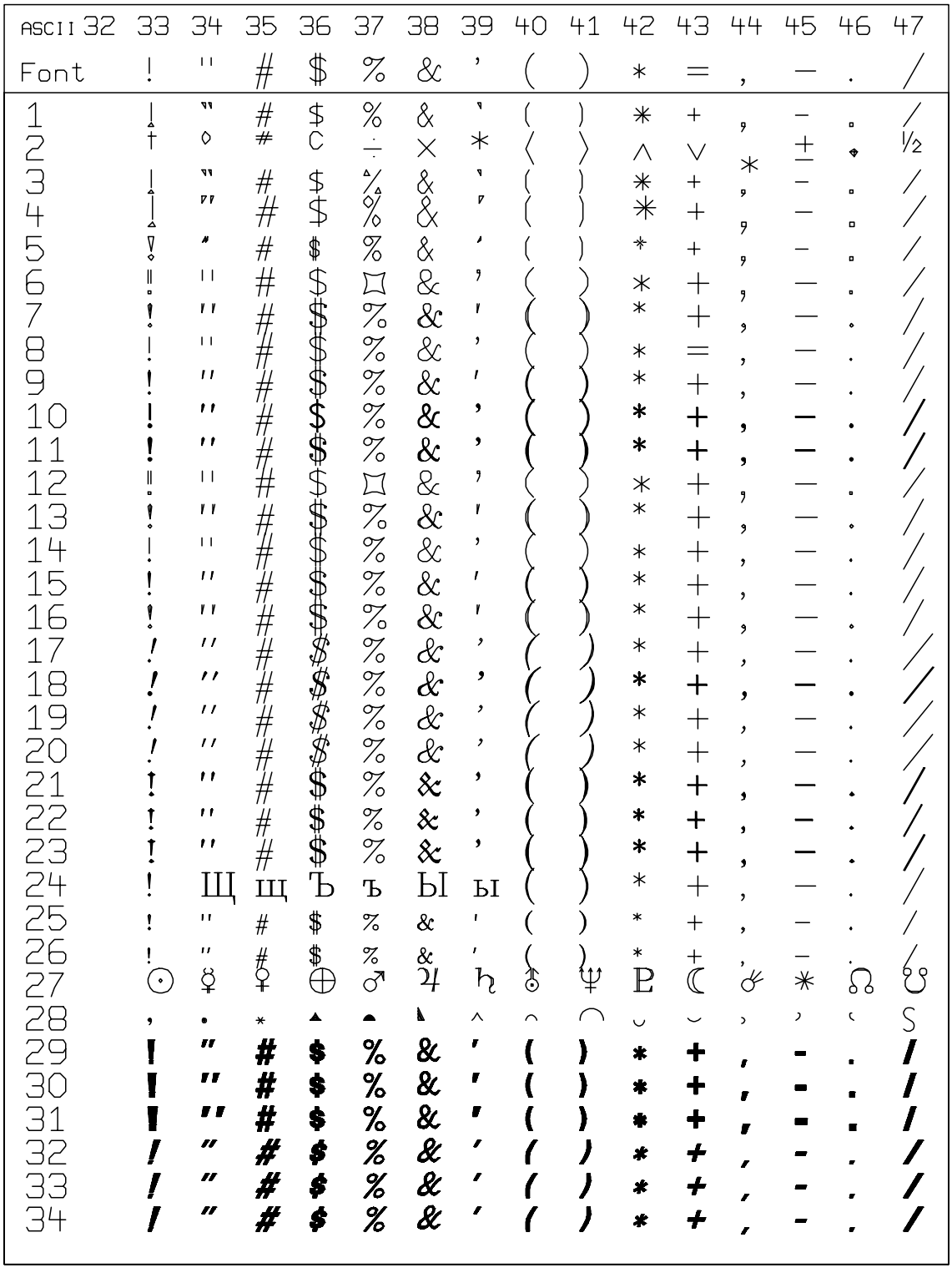
The EXIT statement is used to terminate an MLAB session. The file MLAB.LOG is closed, thereby saving the record of the session. An MLAB session may also sometimes be terminated by typing *CTRL-C* (hold down the control key and type the c-key), but this means of exiting from MLAB does not close the file MLAB.LOG and the hard disk file structure can be corrupted.

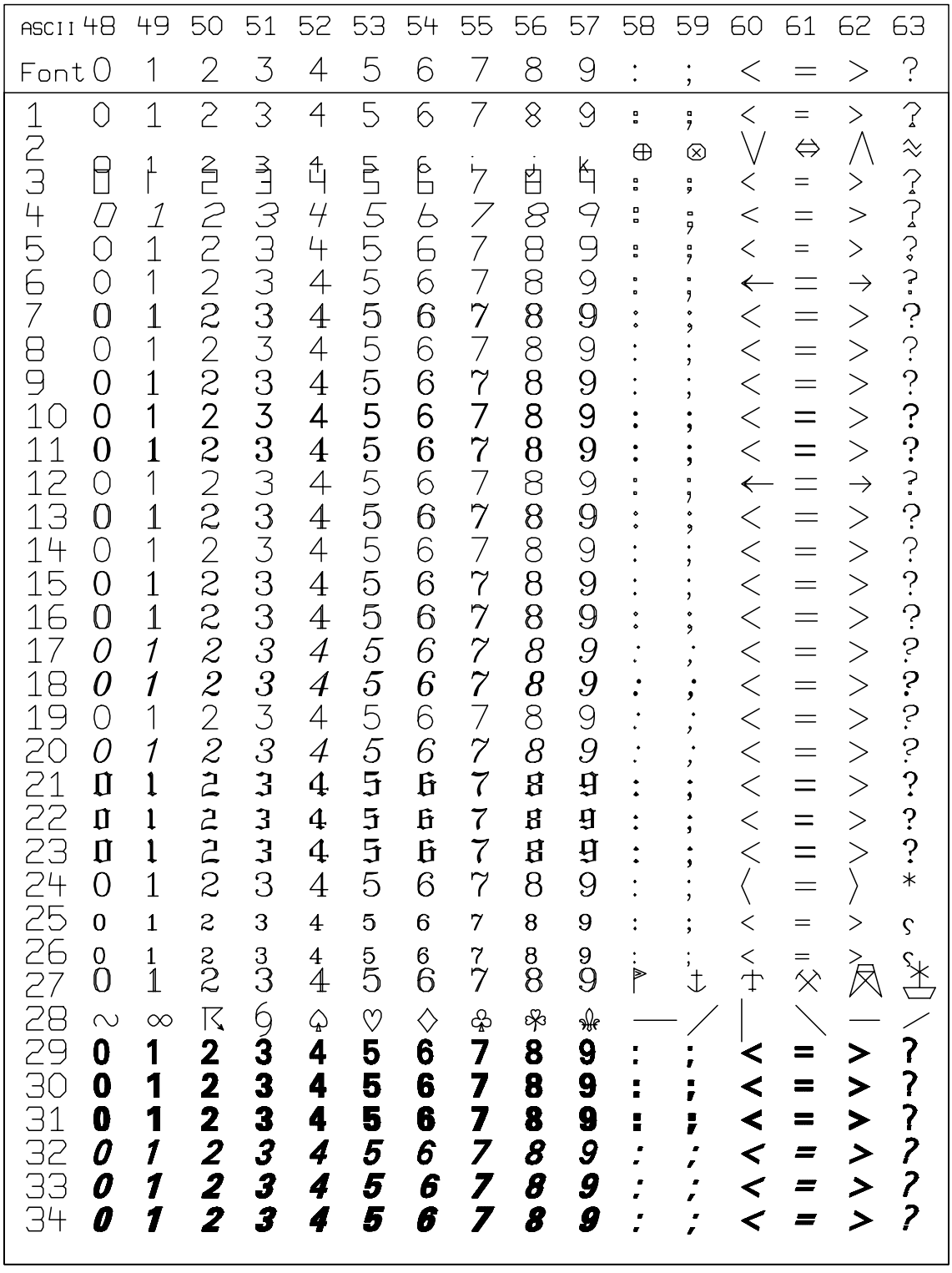
To interrupt an MLAB session while a command is being executed, type *CTRL-BREAK* (hold down the control key and the *break* key simultaneously). MLAB will return to top level and emit a prompt. At this point, the user may save any desired objects using the save statement, and then use the EXIT statement to leave MLAB.

43. Font Tables




ASCII 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Font
1 ⊂ ⊃ ∪ ∩ ∀ ∃ ⟷ ← → ≠ ≤ ≥ ≡
2 ∏ ⊄ ⊆ ⊇ ⊨ ⊢ Ω ♂ ♀ ± ≤ ≥ ↗ ↑ ←
3 ⊂ ⊃ ∪ ∩ ∀ ∃ ⟷ → ≠ ≤ ≥ ≡
4
5 ⊂ ⊃ ∩ ∪ ∀ ∃ ↔ ← → ≠ ≤ ≥ ≡
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

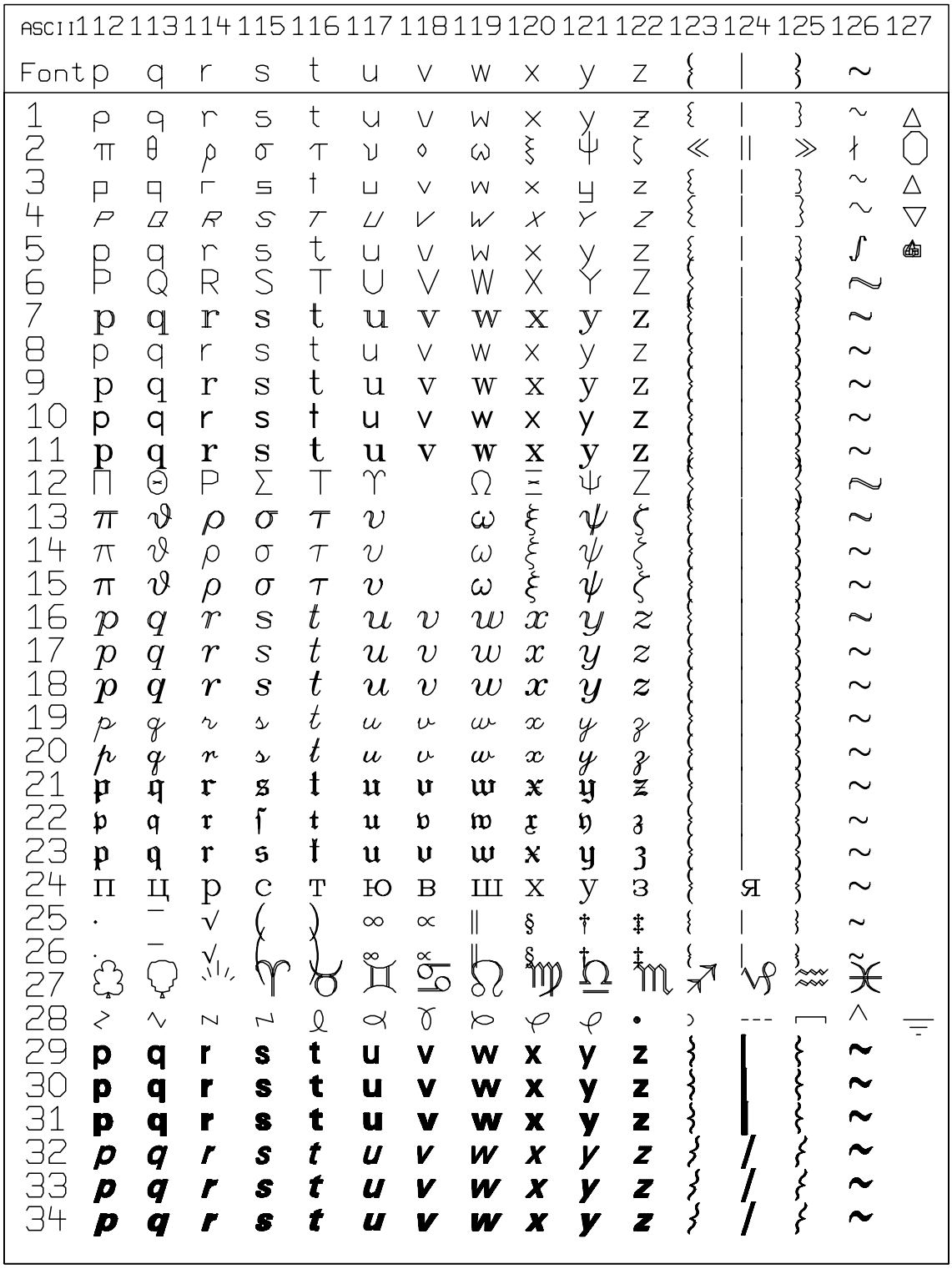

ASCII 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
Font ! '' # $ % & ' ( ) * = , — . /
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34


ASCII 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
Font 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34

ASCII	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
Font	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	℮	🄰	🄱	🄲	🄳	🄴	🄵	🄶	🄷	🄸	🄹	🄺	🄻	🄼	🄽	🄾
2	∅	🅀	🅁	🅂	🅃	🅄	🅅	🅆	🅇	🅈	🅉	🅊	🅋	🅌	🅍	🅎
3	℮	🄰	🄱	🄲	🄳	🄴	🄵	🄶	🄷	🄸	🄹	🄺	🄻	🄼	🄽	🄾
4	℮	𝐴	𝐵	𝐶	𝐷	𝐸	𝐹	𝐺	𝐻	𝐼	𝐽	𝐾	𝐿	𝑀	𝑁	𝑂
5	℮	🄰	🄱	🄲	🄳	🄴	🄵	🄶	🄷	🄸	🄹	🄺	🄻	🄼	🄽	🄾
6	▪	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
7	@	𝔸	𝔹	ℂ	𝔻	𝔼	𝔽	𝔾	ℍ	𝕀	𝕁	𝕂	𝕃	𝕄	ℕ	𝕆
8	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
9	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
10	@	𝗔	𝗕	𝗖	𝗗	𝗘	𝗙	𝗚	𝗛	𝗜	𝗝	𝗞	𝗟	𝗠	𝗡	𝗢
11	@	𝗔	𝗕	𝗖	𝗗	𝗘	𝗙	𝗚	𝗛	𝗜	𝗝	𝗞	𝗟	𝗠	𝗡	𝗢
12	▪	Α	Β	Χ	Δ	Ε	Φ	Γ	Η	Ι		Κ	Λ	Μ	Ν	Ο
13	@	𝛢	𝛣	𝛸	𝛥	𝛦	𝛷	𝛤	𝛨	𝛪		𝛫	𝛬	𝛭	𝛮	𝛰
14	@	Α	Β	Χ	Δ	Ε	Φ	Γ	Η	Ι		Κ	Λ	Μ	Ν	Ο
15	@	Α	Β	Χ	Δ	Ε	Φ	Γ	Η	Ι		Κ	Λ	Μ	Ν	Ο
16	@	𝐴	𝐵	𝐶	𝐷	𝐸	𝐹	𝐺	𝐻	𝐼	𝐽	𝐾	𝐿	𝑀	𝑁	𝑂
17	@	𝐴	𝐵	𝐶	𝐷	𝐸	𝐹	𝐺	𝐻	𝐼	𝐽	𝐾	𝐿	𝑀	𝑁	𝑂
18	@	𝑨	𝑩	𝑪	𝑫	𝑬	𝑭	𝑮	𝑯	𝑰	𝑱	𝑲	𝑳	𝑴	𝑵	𝑶
19	@	𝒜	ℬ	𝒞	𝒟	ℰ	ℱ	𝒢	ℋ	ℐ	𝒥	𝒦	ℒ	ℳ	𝒩	𝒪
20	@	𝒜	ℬ	𝒞	𝒟	ℰ	ℱ	𝒢	ℋ	ℐ	𝒥	𝒦	ℒ	ℳ	𝒩	𝒪
21	@	𝕬	𝕭	𝕮	𝕯	𝕰	𝕱	𝕲	𝕳	𝕴	𝕵	𝕶	𝕷	𝕸	𝕹	𝕺
22	@	𝔄	𝔅	ℭ	𝔇	𝔈	𝔉	𝔊	ℌ	ℑ	𝔍	𝔎	𝔏	𝔐	𝔑	𝔒
23	@	Ꮋ	Ᏼ	Ꮯ	Ꭰ	Ꭼ	Ꮀ	Ꮆ	Ꮋ	Ꮖ	Ꭻ	Ꮶ	Ꮮ	Ꮇ	Ꮑ	Ꭷ
24	Ь	А	Б	Ч	Д	Е	Ф	Г	Ж	И	Й	К	Л	М	Н	О
25	⊛	⟨	⟩	∮	↑	∃	±	∓	÷	∫	≠	≡	≦	≧	∇	ϕ
26	@	⟨	⟩	∮	↑	∃	±	∓	÷	∫	≠	≡	≦	≧	∇	ϕ
27	⛷	○	□	△	◇	☆	┼	×	✳	◍	▦	⟁	⧏	⧐	⫸	✯
28	/	.	‿	⁀	₀	₀	●	♯	♮	♭	‒	-	ˣ	˥	𝄞	𝄿
29	**Å**	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**
30	**Å**	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**
31	**Å**	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**
32	***Á***	***A***	***B***	***C***	***D***	***E***	***F***	***G***	***H***	***I***	***J***	***K***	***L***	***M***	***N***	***O***
33	***Á***	***A***	***B***	***C***	***D***	***E***	***F***	***G***	***H***	***I***	***J***	***K***	***L***	***M***	***N***	***O***
34	***Á***	***A***	***B***	***C***	***D***	***E***	***F***	***G***	***H***	***I***	***J***	***K***	***L***	***M***	***N***	***O***

ASCII	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
Font	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	←
1	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
2	ℙ	ℚ	ℝ	𝕊	𝕋	𝕌	𝕍	𝕎	𝕏	𝕐	ℤ	⋃	-½	⋂	-1	⇒
3	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	↑	←
4	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*	[	\	]	^	_
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	↑	_
6	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	←
7	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	[	\	]	^	←
8	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	←
9	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	←
10	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	[	\	]	^	←
11	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	[	\	]	^	←
12	Π	Θ	P	Σ	T	ϒ		Ω	Ξ	Ψ	Z	[	\	]	^	←
13	Π	Θ	P	Σ	T	ϒ		Ω	Ξ	Ψ	Z	[	\	]	^	←
14	Π	Θ	P	Σ	T	ϒ		Ω	Ξ	Ψ	Z	[	\	]	^	←
15	Π	Θ	P	Σ	T	ϒ		Ω	Ξ	Ψ	Z	[	\	]	^	←
16	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*	[	\	]	^	←
17	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*	[	\	]	^	←
18	***P***	***Q***	***R***	***S***	***T***	***U***	***V***	***W***	***X***	***Y***	***Z***	[	\	]	^	←
19	𝒫	𝒬	ℛ	𝒮	𝒯	𝒰	𝒱	𝒲	𝒳	𝒴	𝒵	[	\	]	^	←
20	𝒫	𝒬	ℛ	𝒮	𝒯	𝒰	𝒱	𝒲	𝒳	𝒴	𝒵	[	\	]	^	←
21	𝔓	𝔔	ℜ	𝔖	𝔗	𝔘	𝔙	𝔚	𝔛	𝔜	ℨ	[	\	]	^	←
22	𝔓	𝔔	ℜ	𝔖	𝔗	𝔲	𝔙	𝔚	𝔛	𝔜	ℨ	[	\	]	^	←
23	Ꝑ	Ꝗ	Ꝛ	Ꞩ	Ꞇ	Ꞟ	∀	Ⓦ	Ⓧ	Ⓨ	Ⓩ	[	\	]	^	←
24	П	Ц	Р	С	Т	Ю	В	Ш	Х	У	З	[	ь	]	Э	э
25	П	‾	√	Σ	⊂	∪	⊃	∩	×	→	↓	[	\	]	^	←
26	Π	‾	√	Σ	⊂	∪	⊃	∩	×	→	↓	[	\	]	^	←
27	♊	♉	☉	♀	♀	⊕	♂	♃	♄	⊙	♆	♇	☾	♌	✳	♋
28	𝄫	𝅘𝅥	𝅗𝅥	𝅗𝅥	♭	𝅝	𝅗𝅥	♯	♮	♭	𝄪	𝄽	𝄾	𝄿	𝄞	𝄢
29	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	[	\	]	^	←
30	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	[	\	]	^	←
31	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	[	\	]	^	←
32	***P***	***Q***	***R***	***S***	***T***	***U***	***V***	***W***	***X***	***Y***	***Z***	[	\	]	^	←
33	***P***	***Q***	***R***	***S***	***T***	***U***	***V***	***W***	***X***	***Y***	***Z***	[	\	]	^	←
34	***P***	***Q***	***R***	***S***	***T***	***U***	***V***	***W***	***X***	***Y***	***Z***	[	\	]	^	←

ASCII	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
Font	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
1	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
2	⊕	α	β	χ	δ	ε	φ	γ	η	ι		κ	λ	μ	ν	ο
3	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
4	△	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*
5	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
6	‘	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
7	‘	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**
8	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
9	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
10	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
11	‘	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**
12	‘	A	B	X	△	E	Φ	Γ	H	I		K	Λ	M	N	O
13	‘	α	β	χ	δ	ε	φ	γ	η	ι		κ	λ	μ	ν	ο
14	‘	α	β	χ	δ	ε	φ	γ	η	ι		κ	λ	μ	ν	ο
15	‘	α	β	χ	δ	ε	φ	γ	η	ι		κ	λ	μ	ν	ο
16	‘	*α*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*	*n*	*o*
17	‘	*α*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*	*n*	*o*
18	‘	***α***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***	***k***	***l***	***m***	***n***	***o***
19	‘	*α*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*	*n*	*o*
20	‘	*α*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*	*n*	*o*
21	‘	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**
22	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
23	‘	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**
24	Я	а	б	ч	д	е	ф	г	ж	и	й	к	л	м	н	о
25	‘	ℵ	⊢	⊣	∂	∈	∫	∫	θ	∫	—	√	∫	∫	˘	∘
26	‘	ℵ	⊢	⊣	∂	∈	∫	∫	θ	∫	✠	√	∫	∫	˘	∘
27	‘	ℵ	⊢	⊣	○	○	∫	∫	θ	∫	✠	√	✡	🔔	🌴	🌲
28	‘	|	\	\	–	/	|	\	(	(	)	)	)	(	)	(
29	‘	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**
30	‘	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**
31	‘	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**
32	‘	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***	***k***	***l***	***m***	***n***	***o***
33	‘	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***	***k***	***l***	***m***	***n***	***o***
34	‘	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***	***k***	***l***	***m***	***n***	***o***


ASCII 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
Font p q r s t u v w x y z { | } ~
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34

Index