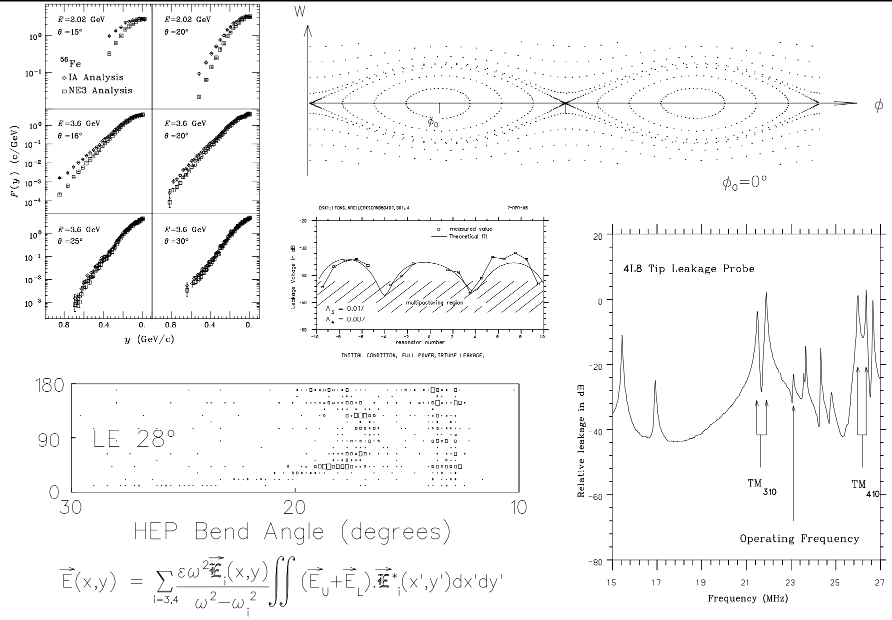
TRIUMF	4004 Wesbrook Mall, Vancouver, B.C., Canada V6T 2A3			
Computing Document	J.L. Chuma	January 1998	TRI-CD-93-01	v1.3

PHYSICA ©
REFERENCE MANUAL

Mathematical Analysis and Data Visualization Software



TRIUMF makes no warranty of any kind with regard to this material.

TRIUMF shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Contents

List of Tables

List of Figures

1 INTRODUCTION

PHYSICA provides a high level, interactive programming environment. The program constitutes a fully procedural programming language, with built-in user friendly graphics and sophisticated mathematical analysis capabilities. Combining an accessible user interface along with comprehensive mathematical and graphical features, makes PHYSICA a general purpose research tool for scientific, engineering and technical applications.

PHYSICA provides you with a wide range of mathematical and graphical operations. Over 200 mathematical functions are available, as well as over 30 operators, providing all of the standard operations of simple calculus, along with powerful curve fitting, filtering and smoothing techniques. The program employs a dynamic array management scheme allowing you a large number of arrays of unlimited size. Algebraic expressions are evaluated using a lexical scanner approach. These expressions can have up to 1500 "tokens," where a token is a literal constant, a variable name, a function name, or an operator. Array evaluations and assignments can be implemented in a simple, direct manner.

Line graphs, histograms and pie-charts, as well as contour, density and surface plots are available. Publication quality graphics can be easily obtained. You have complete control over the appearance of a drawing.

Initial development was for the VAX/VMS operating systems, but the program has been ported to AlphaVMS, ULTRIX, Digital Unix, Silicon Graphics IRIX, HP-UX, IBM AIX, SUNOS and Solaris, and most recently, PC Linux.

The user interacts with the program through the user interface, consisting of monitor dependent routines for display of messages and for reading user input, and device dependent routines for displaying drawings and obtaining hardcopies of user sessions. The user interface is a high-level command language that incorporates a simple to use and easy to learn syntax, based on context-free lexical scanners. The command language incorporates the basic elements of a structured programming language, including conditional branching, looping and subroutine calling constructs.

1.1 What is in this manual

The bulk of this manual is a reference guide to the program commands. These commands are discussed in alphabetical order.

The PHYSICA program provides the user with a large variety of analysis tools, including a fairly comprehensive set of operators and functions. The next chapter describes the operators that are available for use in expressions, followed by a chapter on the built-in functions

that can also be used in any expression.

The first appendix contains definitions for the GPLOT graph and text plot characteristic keywords which are controlled by the SET and GET commands. Other appendices contain the command procedures for creating shareable images of user defined routines, for use with the VAX/VMS and AlphaVMS operating systems. This is followed by the default source code for these user defined routines.

Users are referred to the PHYSICA USER'S GUIDE for examples of program usage. Those who are familiar with the predecessor program, PLOTDATA, are referred to the PLOTDATA TO PHYSICA CONVERSION MANUAL for tips on converting PLOTDATA command macros to PHYSICA.

1.2 Conventions used in this manual

Examples of messages and prompts written by the program, as well as examples of user typed input are displayed in `typewriter type style`.

Commands and other reserved keywords are in `UPPERCASE`.

Curly brackets, `{ }`, enclose parameters that are optional and/or have default values; and indicate that it is not necessary to enter these parameters. Vertical bars, `|`, separate choices for command parameters.

Curly brackets and vertical bars should *not* be entered with commands.

Parentheses, `( )`, enclose formats. The back slash, \, separates a command from a command qualifier or a parameter from its qualifier. The opening quote, `‘`, and the closing quote, `’`, delimit literal strings.

Parentheses, the back slash and quotes *must* be included where indicated.

VMS usually refers to the OpenVMS operating system for either the VAX or the Alpha architectures.

UNIX refers to any UNIX like operating system, including Linux.

2 COMMANDS

3DPLOT

Syntax	`3DPLOT x y z ipen { colr }`
Defaults	`colr = current colour`

The `3DPLOT` command graphs the three vectors `x`, `y`, and `z` in $3d$ space, displayed in 2 dimensions using a perspective projection. The vectors `x`, `y`, and `z` should contain the (x, y, z) coordinates of the points. The `ipen` vector contains the codes for deciding what to draw at each coordinate. The `colr` vector contains the colour codes. The vectors must all be the same length.

This command is strictly interactive, it cannot be entered from batch mode. Note that if the graph seems to be a complete mess, try increasing the “eye to object distance” using the `E` key.

`ipen[j]`	
$= 2$	coordinate set `j` is connected by a line segment to `j-1`
$= 3$	coordinate set `j` is not connected to `j-1`
$= 20$	a point is plotted at coordinate set `j`
< 0	a GPLOT symbol `|ipen[j]|` is plotted at coordinate set `j`

When drawing symbols, the size of the symbols can be controlled with the `SET CHARSZ` command before entering the `3DPLOT` command, and changed inside the command by using the `Z` key. Use the `DISPLAY PCHAR` command to see the possible GPLOT special symbols.

If drawing line segments, `colr[j]` is the colour code for the last line segment `j-1` to `j`. The first colour code is ignored. If drawing a point at the $\mathtt{j}_{th}$ coordinate, that is, `ipen[j]` $= 20$, or if drawing a symbol at the $\mathtt{j}_{th}$ coordinate, that is, `ipen[j]` < 0, then `colr[j]` is the colour of that point or symbol.

On X Window type monitors, the focus must be in the alphanumeric terminal window to use the menu.

The default axis number height is `XNUMSZ`, which can be changed with the `SET XNUMSZ` command before entering the `3DPLOT` command, and changed inside the command by using the `H` key.

The current hardcopy device is disabled while the `3DPLOT` command is active. The `S` key causes the current hardcopy device to be re-enabled, and the plot to be redrawn, thus allowing you to obtain a hardcopy, with the `HARDCOPY` comamnd.

`<`	rotate left 1 angle increment	`L`	rotate left 360 degrees
`>`	rotate right 1 angle increment	`R`	rotate right 360 degrees
`^`	rotate up 1 angle increment	`U`	rotate up 360 degrees
`V`	rotate down 1 angle increment	`D`	rotate down 360 degrees
`I`	zoom in	`O`	zoom out
`A`	angle increment	`E`	eye to object distance
`H`	axis number height	`Z`	symbol size
`S`	save for hardcopy	`M`	display this menu
`Q`	quit		

Table 2.1: The control key menu for the 3DPLOT command

On X Window type monitors, the keyboard focus must be in the alphanumeric terminal window to use the menu.

ALIAS

Syntax	`ALIAS { newcommand command_string }`
Examples	`ALIAS`
	`ALIAS RED COLOUR RED`
	`ALIAS APPEND WRITE\APPEND`
	`ALIAS CURVE GRAPH\NOAXES`
	`ALIAS AXES GRAPH\AXES`
	`ALIAS ACLEAR CLEAR\ALPHANUMERIC`

The `ALIAS` command creates new commands by equating a user defined keyword, `newcommand`, to a string, `command_string`. The command string must begin with a valid command, for example, it cannot be a file name that is to be used with other commands. Everything after `newcommand` is taken for the `command_string`, that is, no quotes should be used.

If the `ALIAS` command is entered with no parameters, then all of the current aliases will be displayed. The maximum number of aliases that can exist at one time is 100./indexalias!maximum number

Use the `DEALIAS` command to eliminate aliases.

ASSIGN

Syntax	`ASSIGN name logical`
Example	`ASSIGN LASER_211 HP$LASER`

The `ASSIGN` command is only relevant for VMS. It is equivalent to the DCL command: `ASSIGN name logical`

The logical assignment takes effect immediately, and remains in effect after PHYSICA is unloaded. Both parameters should be strings.

The `ASSIGN` command is useful for assigning new logical names to the output devices, that is, the logical name for a bitmap device or a plotter.

BESTFIT

Syntax	`BESTFIT pmin pmax penalty error parm pout`
	`BESTFIT\CYCLES n pmin pmax penalty error parm pout`
	`BESTFIT\WEIGHTS w pmin pmax penalty error parm pout`
	`BESTFIT\CYCLES\WEIGHTS w n pmin pmax penalty error parm pout`
Qualifiers	`\CYCLES, \WEIGHTS`
Defaults	`\NOCYCLES, \NOWEIGHTS`, `pmin` = 0, `pmax` = 1, `penalty` = 1

This command calculates parameters for a least-squares fit to an error vector using adjustable parameters.

Suppose you have an error vector, `error`, of length n. Suppose that there are m variable parameters and that the measured effect of a unit change for each of the parameters at each of the n locations is stored in matrix `parm` with n rows and m columns. Vector `penalty`, of length m, represents the penalty functions to changes of the m parameters. The larger `penalty[i]` the smaller the adjustment of the $\mathtt{i}^{th}$ parameter.

The optimal set of changes of the m parameters within their allowed range of `pmin` to `pmax` will be determined in the least-squares sense. The vector `pout` will contain the parameter changes giving this fit, and will be of length m.

Parameter types and sizes

The influence function `parm` must be a matrix. Suppose it has n rows and m columns. The parameter ranges, `pmin` and `pmax`, as well as the penalty function `penalty`, must be vectors with the same length, m, which is the number of parameters for the fit. The error vector `error` must be a vector of length n, which is the number of locations.

Weights

Syntax	`BESTFIT\WEIGHTS w pmin pmax penalty error parm pout`
	`BESTFIT\WEIGHTS\CYCLES w n pmin pmax penalty error parm pout`

If a weight vector, `w`, is entered, you *must* indicate that it is there by using `\WEIGHTS` qualifier. The weight, `w[i]`, corresponds to the importance of reducing the initial error to zero at the ith location. The weight array should be a vector of length n. The closer to zero the value of w[i], the looser will be the fit at the ith location. If the \CYCLES qualifier is also used, the weight array comes before the iteration number in the parameter list.

Cycles

Syntax `BESTFIT\CYCLES n pmin pmax penalty error parm pout`
`BESTFIT\CYCLES\WEIGHTS w n pmin pmax penalty error parm pout`

The \CYCLES qualifier allows the user to specify the maximum number of iteration steps for the fit. When this maximum number is reached, the fit will stop. The fit will also stop if the fit is successful before this maximum iteration number is reached. If the \WEIGHTS qualifier is also used, the weight array comes before the iteration number in the parameter list.

BIN

Syntax `BIN x xbin xcount`
`BIN\NBINS x xbin xcount n { xmin xmax }`

Qualifiers \WEIGHTS, \EDGES, \NBINS, \DISCARD, \EMPTY, \AVERAGE, \LAGRANGE

Defaults \-WEIGHTS, \-EDGES, \-NBINS, \-DISCARD, \-EMPTY, \-AVERAGE, \LAGRANGE xmin = min(x), xmax = max(x)

The BIN command sorts an input vector, x, into a grid of bins and accumulates the counts per bin into an output vector, xcount. Each element of x is considered only once, so elements are never counted as being in more than one bin. By default, the bins are defined by their centres, given in vector xbin, which must be strictly monotonically increasing. If $n =$ len(xbin), define the bin ranges, r_i

$$r_1 = \mathtt{xbin}_1 - (\mathtt{xbin}_2 - \mathtt{xbin}_1)/2$$

$$r_i = \mathtt{xbin}_i - (\mathtt{xbin}_i - \mathtt{xbin}_{i-1})/2 \quad \text{for} \quad i = 2, 3, \ldots, n$$

$$r_{n+1} = \mathtt{xbin}_n + (\mathtt{xbin}_n - \mathtt{xbin}_{n-1})/2$$

For each $i = 1, 2, \ldots, \text{len}(\mathtt{x})$, if $r_j \le \mathtt{x}_i < r_{j+1}$ for some $j = 1, 2, \ldots, n$ then $\mathtt{xcount}_j$ is incremented by 1, or by the weight, $\mathtt{w}_i$. By default, events below r_1 will be placed in the first bin, and events above r_{n+1} will be placed in the last bin. If the \DISCARD qualifier is used, events outside this range will be discarded.

Weights

Syntax `BIN\WEIGHTS w x xbin xcount`
`BIN\AVERAGE\WEIGHTS w x xbin xcount`
`BIN\EMPTY\WEIGHTS w x xbin xcount`
`BIN\EDGES\WEIGHTS w x xbin xcount`
`BIN\EDGES\EMPTY\WEIGHTS w x xbin xcount`
`BIN\EDGES\AVERAGE\WEIGHTS w x xbin xcount`

By default, a bin count is incremented by one (1) for every event that goes in a bin. If a weight vector is entered, you *must* indicate that it is there by using \WEIGHTS qualifier. The weight w must be a vector with the same length as x. The i^{th} event causes the bin count to be incremented by $\mathtt{w}_i$.

Number of bins

Syntax `BIN\NBINS x xbin xcount n { xmin xmax }`

Defaults xmin = min(x), xmax = max(x)

By default, the bins are defined by their centres, given in vector xbin, which must be strictly monotonically increasing. If the \NBINS qualifier is used, the number of bins, n, is expected. A new vector, xbin, will be created which will have n elements. If the numbers xmin and xmax are not entered, they default to the minimum and maximum of vector x.

$$\mathtt{xbin}_i = \mathtt{xmin} + (i - \frac{1}{2})(\mathtt{xmax} - \mathtt{xmin})/\mathtt{n} \quad \text{for} \quad i = 1, 2, \ldots, \mathtt{n}$$

Lagrange

Syntax `BIN\LAGRANGE x xbin xcount`

If the \LAGRANGE qualifier is used, \WEIGHTS, \EDGES, \AVERAGES, and \EMPTY are not allowed.

If $n =$ len(xbin), define the bin ranges, r_i

$$r_1 = \mathtt{xbin}_1 - (\mathtt{xbin}_2 - \mathtt{xbin}_1)/2$$

$$r_i = \mathtt{xbin}_i - (\mathtt{xbin}_i - \mathtt{xbin}_{i-1})/2 \quad \text{for} \quad i = 2, 3, \ldots, n$$

$$r_{n+1} = \mathtt{xbin}_n + (\mathtt{xbin}_n - \mathtt{xbin}_{n-1})/2$$

For each $i = 1, 2, \ldots, \text{len}(\mathtt{x})$ find j so that $r_j \le \mathtt{x}_i < r_{j+1}$ for some $j = 1, 2, \ldots, n$. If $j = n$, then $\mathtt{xcount}_n$ is incremented by 1, otherwise, let $w = (\mathtt{x}_i - \mathtt{xbin}_j)/(r_{j+1} + r_j)/2$ then $\mathtt{xcount}_j$ is incremented by $1 - w$ and $\mathtt{xcount}_{j+1}$ is incremented by w.

Averages

Syntax `BIN\AVERAGE x xbin xcount`
`BIN\AVERAGE\EDGES x xbin xcount`

The `\AVERAGE` qualifier means that the output `xcount` vector will contain the average value for each bin. An internal counter is kept for each bin, and the value for $\mathtt{xcount}_i$ will be divided by the number of events in bin i before it is output. `\AVERAGE` can be used with `\WEIGHTS` and `\EDGES`, but not with `\EMPTY`.

Increment only if empty

Syntax `BIN\EMPTY x xbin xcount`
`BIN\EMPTY\EDGES x xbin xcount`

The `\EMPTY` qualifier means that an event is counted in a bin only if that bin is empty. So only the first event encountered for each bin will be counted in that bin. `\EMPTY` can be used with `\WEIGHTS` and `\EDGES`, but not with `\AVERAGE`.

Edge defined bins

Syntax `BIN\EDGES x xbin xcount`

By default, bins are defined by their centres. If the `\EDGES` qualifier is used, bins are defined by their edges. The bin edges must be given in vector `xbin`. The length of `xcount` will be one less than the length of `xbin`. A weight vector, `w`, may be specified if you use the `\WEIGHTS` qualifier.

If $\mathtt{xbin}_j \le \mathtt{x}_i < \mathtt{xbin}_{j+1}$ then $\mathtt{xcount}_j$ is incremented either by one (1), or by the specified weight, $\mathtt{w}_i$.

BIN2D

Syntax	`BIN2D x y xbin ybin mc nx ny { xmin xmax ymin ymax }` `BIN2D\MATRIX mdata mxin myin mout`
Qualifiers	`\WEIGHTS, \EMPTY, \MATRIX,` `\XDISCARD, \YDISCARD, \DISCARD`
Defaults	`\-WEIGHTS, \-MATRIX, \-EMPTY, \-DISCARD` `xmin` = min(x), `xmax` = max(x), `ymin` = min(y), `ymax` = max(y)

The `BIN2D` command forms a matrix of bins of data by sorting the vectors `x` and `y` into grids of bins which are returned in vectors `xbin` and `ybin`. The accumulated matrix of total counts per bin is returned in matrix `mc`. If the numbers `xmin` and `xmax` are not entered, they default

to the minimum and maximum of `x`. Similarly, if the numbers `ymin` and `ymax` are not entered, they default to the minimum and maximum of `y`.

$$\mathtt{xbin}_i = \mathtt{xmin} + (i - \frac{1}{2})(\mathtt{xmax} - \mathtt{xmin})/|\mathtt{nx}| \quad \text{for} \quad i = 1, 2, \dots, |\mathtt{nx}|$$

$$\mathtt{ybin}_j = \mathtt{ymin} + (j - \frac{1}{2})(\mathtt{ymax} - \mathtt{ymin})/|\mathtt{ny}| \quad \text{for} \quad j = 1, 2, \dots, |\mathtt{ny}|$$

The $(\mathtt{x}_i, \mathtt{y}_i)$ point will be accumulated in $\mathtt{mc}_{i,j}$ where:
row: $i = \mathrm{int}((\mathtt{y}_i - \mathtt{ymin})/(\mathtt{ymax} - \mathtt{ymin})|\mathtt{ny}|) + 1$
column: $j = \mathrm{int}((\mathtt{x}_i - \mathtt{xmin})/(\mathtt{xmax} - \mathtt{xmin})|\mathtt{nx}|) + 1$

Dimensions

The lengths of `x` and `y` must be equal. If a weight vector, `w`, is supplied, it must also be the same length.

The vectors `xbin` and `ybin` and the matrix `mc` will be created. `xbin` will have `|nx|` elements, `ybin` will have `|ny|` elements, and matrix `mc` will have `|nx|` columns and `|ny|` rows.

Extremes

By default, events below `xmin` are placed in the first bin column, events above `xmax` are placed in the last bin column, events below `ymin` are placed in the first bin row, and events above `ymax` are placed in the last bin row. If the `\DISCARD` qualifier is used, events outside either of these ranges will be discarded. If the `\XDISCARD` qualifier is used, events below `xmin` are discarded, and events above `xmax` are discarded. If the `\YDISCARD` qualifier is used, events below `ymin` are discarded, and events above `ymax` are discarded.

Weights

Syntax `BIN2D\WEIGHTS w x y xbin ybin mc nx ny { xmin xmax ymin ymax }`

If a weight is entered, you *must* indicate that it is there by using `\WEIGHTS` qualifier. The weight `w` must be a vector. The i^{th} event causes the bin count to be incremented by $\mathtt{w}_i$.

Increment only if empty

Syntax `BIN2D\EMPTY x y xbin ybin mc nx ny { xmin xmax ymin ymax }`

If the `\EMPTY` qualifier is used, an event is counted in a bin only if that bin is empty. Only the first event encountered for each bin will be counted in that bin. `\EMPTY` cannot be used with `\MATRIX`.

Defined by box corners

Syntax `BIN2D\MATRIX mdata mx my mc`

The `BIN2D\MATRIX` command calculates the sum of the data points given by matrix `mdata` within a set of boxes. The x-coordinates of the boxes are given in matrix `mx`, the y-coordinates are given in matrix `my`. Matrices `mx` and `my` must be the same size. A data point is taken to be inside a box if it is interior or on an edge. Each data point is considered only once, so a data point is never taken to be in more than one of the boxes. The coordinates of the data points are the row and column indices, for example, `mdata[3,4]` is row 3 and column 4 so it is at (x, y) location $(4, 3)$. The x and y-coordinates in `mx` and `my` should be in this index space of coordinates. The qualifiers `\EMPTY` and `\WEIGHTS` cannot be used with `\MATRIX`.

See the `PICK\MATRIX` command for information on interactively choosing the above mentioned boxes.

BUFFER

Syntax	`BUFFER { n }` `BUFFER\READ filename` `BUFFER\WRITE filename`
Qualifiers	`READ, WRITE, DYNAMIC, STATIC, KEYPAD`
Defaults	`n` = 20, `\DYNAMIC`

The `BUFFER` command controls the input line recall buffers:

- the dynamic buffer
- the static buffer
- the keypad buffer

Table 2.2 on page 13 shows the control keys recognized by this terminal interface. Table 2.3 on page 13 shows the function keys recognized by the terminal interface.

See the `RESTORE\PHYSICA` command, page 211, for information on restoring these buffers from previously saved sessions. There is an option to not restore these buffers, by using the `\NOTTBUFFERS` qualifier.

Parameters

If no parameters are entered and no qualifiers are used, the current length of the dynamic recall buffer is displayed. If `n` is entered, this will be the new length of the dynamic buffer as displayed when the `PF1` key is typed. The value of `n` must be $0 < \mathtt{n} \leq 35$.

Reading the buffers

If the `\READ` qualifier is used, the dynamic buffer will be read from the specified file. You can also read the static buffer or the keypad buffer by also using the `\STATIC` qualifier or the `\KEYPAD` qualifier. For example:

```
BUFFER\KEYPAD\READ FILE.DAT
```

will read the keypad buffer from `FILE.DAT`

Writing the buffers

If the `\WRITE` qualifier is used, the dynamic buffer will be written to the specified file. You can also write the static buffer or the keypad buffer by also using the `\STATIC` qualifier or the `\KEYPAD` qualifier. For example:

```
BUFFER\KEYPAD\WRITE FILE.DAT
```

will write the keypad buffer to `FILE.DAT`

Dynamic buffer

The dynamic buffer is a terminal interface which closely mimics the DCL command recall facility. The arrow, delete, backspace, and most control keys, work as in DCL. An input line is stored automatically in the dynamic buffer when a carriage return is typed. The line is stored at the top of the stack, with previously entered lines being pushed down the stack. The maximum length of the buffer stack is 35 lines. After 35 lines have been stored, the lines at the bottom of the stack begin dropping off and are lost.

Static buffer

The static buffer is similar to the dynamic buffer, but the lines in the static buffer are not updated automatically. To interactively enter a line into the static buffer, type the input line and then type the `PF3` keypad key followed by `control-L`. You will be asked to enter a storage number. Enter a digit from `1` to `9` or a letter from `A` to `Z`, where `A` represents `10`, `B` represents `11`, and so on. To recall a line previously stored in the static buffer, type the `PF3` keypad key, and then type the storage digit or letter of the desired line.

Keypad buffer

The keypad buffer allows the keypad keys: 0 - 9, period, comma, and minus, to be defined. Any of these keys can also be set up such that a carriage return is included. Thus, the command is executed as soon as the key is typed. To interactively enter a keypad key definition, type the input line and then type the `<ENTER>` key on the keypad. You will be asked to type the keypad key to be loaded with the input line. If this is to include a carriage return, type the `<ENTER>` keypad key, otherwise type any other key to resume.

key	*action*
`control-^`	appended to a string recalls the last command containing it
`control-A`	toggles insert/overstrike mode
`control-E`	moves alphanumeric cursor to end of line
`control-H`	(BACKSPACE) moves alphanumeric cursor to the beginning of the input line
`control-K`	disables recall shell, a "!" in column 1 re-enables it
`control-N`	reads the dynamic recall buffer from a file
`control-P`	writes the dynamic recall buffer to a file
`control-R`	refreshes the current input line
`control-X`	(control-U) erases input line to the left of the alphanumeric cursor
Currently `LINEFEED` (`control-J`), `ESC` and `TAB` (`control-I`) are not enabled	

Table 2.2: Control keys recognized by the terminal interface

key	*action*
`PF1`	list and allows selection from dynamic recall buffer
`PF2`	lists the HELP facility
`PF3`	lists, loads (via `control-L`), and selects from static buffer
`PF4`	invokes a simple desk calculator
`ENTER`	lists or loads the keypad buffer
`F14`	toggles insert/overstrike mode

Table 2.3: Function keys recognized by the terminal interface

CALL

Syntax	`CALL { SUBn } arg1 { arg2 ... arg15 }`
Examples	`CALL SUB1 X Y` `CALL X Y`

The `CALL` command uses one of the following:

- a subroutine hardwired into the program
- a subroutine loaded dynamically with the `LOAD` command (VAX/VMS only), or
- a subroutine loaded at run time via a shareable image (VAX/VMS or AlphaVMS).

Functions can not be referenced with the `CALL` command, but the user written functions can be used wherever a function can be used in an expression.

If one of the keyword parameters `SUB1`, `SUB2`, ..., `SUB8` is entered with the `CALL` command, either the built-in subroutine by default, or one of the eight subroutines which were loaded at run time via `PHYSICA_USER_FUNCTIONS`, a shareable image, will be used. The shareable image option is only available under VAX/VMS and AlphaVMS.

If none of these keyword parameters is used, then the object module that was loaded dynamically with the `LOAD` command will be used. The `LOAD` command is only available under VAX/VMS.

For UNIX users, the process to add user defined routines to physica follows:

1. edit the file `phys_user.f` to put in your versions of `user1`, ..., `user8`, `sub1`, ..., `sub8`
2. compile it, e.g., `f77 -c phys_user.f`
3. put it in the archive, e.g., `ar -rsv physica.a phys_user.o`
4. link the program with `physica.link`

The default set of eight (8) subroutines and eight (8) functions is listed in **Appendix D**. For VMS users the sources are provided in the file: `PHYSICA$DIR:PHYSICA_USER_FUNCTIONS.FOR` For UNIX users the sources are provided in the file: `phys_user.f` Off-site UNIX users can find this file in the `physica-link` tar file.

User written subroutine description

If a subroutine is loaded at run time via the sharable image, its name *must* be one of `SUB1`, `SUB2`, ..., `SUB8`. If a subroutine is to be loaded dynamically via the `LOAD` command, its name is irrelevant and can be anything the user desires. In either case, a user written subroutine *must* have the following form:

```
SUBROUTINE subname(IATYPE,ICODE,IUPDATE,IER,arg1,arg2,...)
INTEGER*4 IATYPE(15),ICODE(3,15),IUPDATE(15),IER
```

Other than the required arguments, `IATYPE`, `ICODE`, `IUPDATE`, and `IER`, there may be from 1 to 15 arguments in the subroutine argument list. The user is responsible for insuring that the correct number and type of arguments are used when actually employed with the `CALL` command. The parameters used in the `CALL` command, `argI`, which are passed as arguments to the subroutine, may be constants, scalars, vectors, matrices, literal quote strings, or *scalar* string variables. The number of arguments and the type of argument *must* agree with the actual subroutine.

All of the numeric aguments, except for the required integer arguments `IATYPE`, `ICODE`, `IUPDATE`, and `IER`, must be `REAL*8`. A string argument is passed as a `LOGICAL*1` array.

Note: The integer arguments `IATYPE`, `ICODE`, `IUPDATE`, and `IER` should *not* be mentioned as parameters with the `CALL` command.

See the file: `PHYSICA$DIR:PHYSICA_USER_FUNCTIONS.FOR` for some subroutine examples.

IATYPE

`IATYPE` is an `INTEGER*4` array, length 15, that indicates the type of each of the subroutine arguments `argI`. See Table 2.4 on page 15.

argument type	`IATYPE(i)`
unfilled	−99
string	−1
scalar	0
vector	1
matrix	2

Table 2.4: Interpretation of the `IATYPE` array in user written subroutines

ICODE

`ICODE` is an `INTEGER*4` array, dimensioned 3 by 15, that indicates the dimension of each of the subroutine arguments `argI`. Never extend variables beyond their original size as passed to the subroutine. If a variable is shortened inside the subroutine, the subroutine must update the new dimensions in the `ICODE` array, so that PHYSICA can reduce the variable

dimensions appropriately. See Table 2.5 on page 16.

argument type	`ICODE(1,i)`	`ICODE(2,i)`	`ICODE(3,i)`
string	length	0	0
scalar	0	0	0
vector	length	0	0
matrix	nrows	ncolms	0

Table 2.5: Interpretation of the `ICODE` array in user written subroutines

The `ICODE` array will be filled by PHYSICA with the current dimensions of the arguments, so the user written subroutine can check and, if necessary, update the dimensions of any of the subroutine arguments.

Never extend vectors, matrices, or string variables beyond their original sizes as passed to the user written subroutine. If a variable's size is shortened inside the subroutine, then the subroutine must update the `ICODE` array so that these variable dimensions can be reduced internally by PHYSICA upon return from the subroutine.

IUPDATE

`IUPDATE` is an `INTEGER*4` array, length 15, that the user routine sets to indicate to PHYSICA whether one of the `argI` arguments has been modified inside that subroutine.

The default value for `IUPDATE(i)` is 0. Set `IUPDATE(i)` to 1 to indicate that the ith argument, `argI`, has been modified. Never extend variables beyond their original size as passed to the subroutine. If a variable is shortened inside the subroutine, the subroutine must update the new dimensions in the `ICODE` array, so that PHYSICA can reduce the variable dimensions appropriately.

IER

`IER` is an `INTEGER*4` variable that defaults to the value 0. Your routine can set `IER` to indicate to PHYSICA that an error has occured in the routine. Arithmetic errors, such as division by zero, over/underflow, will be asynchronously trapped. If other error tests are to be done inside the subroutine, the user flags the error by setting `IER = -1` before the `RETURN`. If the `CALL` command was executed from within a script, this error flag causes PHYSICA to abort that script and control is passed back to the keyboard.

Numeric arguments

All the numeric arguments of your subroutine, except for the integer arguments `IATYPE`, ICODE, IUPDATE, and IER, must be REAL*8. A string argument is passed as a LOGICAL*1 array. Dimension numeric array arguments with length 1, for example:

```
REAL*8 X(1), Y(1), Z(1)
```

String arguments

All the string arguments of your subroutine must be LOGICAL*1, and should be dimensioned 1, for example:

```
LOGICAL*1 LFILE(1)
```

You can convert this to a string, say, CHARACTER*80 CFILE, using the following method:

```
LENF = ICODE(1,i)
DO I = 1, LENF
  CFILE(I:I) = CHAR(LFILE(I))
END DO
```

where LFILE is the ith argument.

Accessing matrix data

If a matrix is passed as an argument to a user written subroutine, the elements of the matrix can only be accessed using a calculated index. To access element m[i,j] of the matrix m, use m[i+(j-1)*nrows] for i=1, ..., nrows and j=1, ..., ncols.

Example

If the command is CALL SUB3 A T X M, where A is a scalar, T is a string variable, X is a vector, and M is a matrix, the subroutine should begin as follows:


```
SUBROUTINE SUB3(IATYPE,ICODE,IUPDATE,IER,A,T,X,M)
INTEGER*4  IATYPE(15),ICODE(3,15),IUPDATE(15),IER
REAL*8     A, X(1), M(1)
LOGICAL*1  T(1)
...
LENT =ICODE(1,2)   ! the length of the string variable T
LENX =ICODE(1,3)   ! the length of vector X
NROWS=ICODE(1,4)   ! the number of rows of the matrix M
NCOLS=ICODE(2,4)   ! the number of columns of the matrix M
...
RETURN
END
```

Creating a shareable image

The default set of eight (8) subroutines and eight (8) functions, listed in **Appendix D**, is provided in the file:

```
PHYSICA$DIR:PHYSICA_USER_FUNCTIONS.FOR
```

Copy this file to your own directory. For example:

```
$ copy PHYSICA$DIR:PHYSICA_USER_FUNCTIONS.FOR disk:[directory]TEST.FOR
```

Substitute your source code for the default routines. All eight subroutines and all eight functions *must* be present in your source code file.

A DCL command procedure must be executed *before* you invoke PHYSICA. This command procedure, the VAX/VMS version is listed in **Appendix B** while the AlphaVMS version is listed in **Appendix C**, can be found in: PHYSICA$DIR:PHYSICA_USER_FUNCTIONS.COM

This procedure defines the logical name PHYSICA_USER_FUNCTIONS and creates the sharable image. There should be a system wide default definition of this logical name, so that if the shareable image is not in a user defined location, the default will be used from PHYSICA$DIR.

Copy this procedure to your own directory. For example:

```
$ copy PHYSICA$DIR:PHYSICA_USER_FUNCTIONS.COM disk:[directory]TEST.COM
```

You will need to modify one line:

```
$ define PHYSICA_USER_FUNCTIONS disk:[directory]TEST
```

In this line, replace `disk:[directory]TEST` with the actual location and name of your source code file. Execute the command procedure, which will produce, `PHYSICA_USER_FUNCTIONS.EXE`, a shareable image.

CLEAR

Syntax	`CLEAR`
Qualifiers	`\ALPHANUMERIC`, `\TOGGLE`, `\REPLOTONLY`, `\NOREPLOT`
Defaults	clears graphics and replot option

By default, the `CLEAR` command clears the graphics. It also clears the replot buffers, that is, there will be nothing to replot until something is drawn again. Any hardcopies must be asked for *before* entering the `CLEAR` command.

Alphanumerics

Syntax `CLEAR\ALPHANUMERIC`

The `CLEAR\ALPHANUMERIC` command clears the alphanumeric, transparent, portion of the monitor screen only. This has no affect on graphics hardcopies or the `REPLOT` buffers.

Toggle graphics

Syntax `CLEAR\TOGGLE`

On a CIT467 terminal, the `CLEAR\TOGGLE` command will toggle the graphics screen on or off. The first time this command is entered, the graphics screen is turned off, and any graphics will disappear. The next time `CLEAR\TOGGLE` is entered, the graphics screen will be turned back on, and your graphics will reappear. This has no affect on graphics hardcopies or the `REPLOT` buffers.

Clear the replot buffers only

Syntax `CLEAR\REPLOTONLY`

The `CLEAR\REPLOTONLY` command only clears the replot buffers and does not affect the graphics. There is nothing to replot after `CLEAR\REPLOTONLY` until more data or text is drawn.

Do not clear the replot buffers

Syntax `CLEAR\NOREPLOT`

The `CLEAR\NOREPLOT` command only clears the graphics, it does not affect the replot buffers.

COLOUR

Syntax	`COLOUR colourname` `COLOUR n`
Defaults	default colour is white
Examples	`COLOUR` `COLOUR RED` `COLOUR R\SCALAR`

The `COLOUR` command sets the graphics monitor colour and the associated graphics hardcopy colour number, for subsequent graphics. The colour number may, for example, be a plotter pen number.

If no parameter is entered, a list of the colour names and corresponding numbers is displayed. You may then enter a colour name or number. Table 2.6 shows the recognized colournames and their associated numbers.

Note: The colours black and white have always been a source of confusion. The colour black means the graphics screen background colour. The colour white means white if the background colour is black, and black if the background colour is white.

name	*number*	*name*	*number*
black	0		
white	1	yellow	5
red	2	cyan	6
green	3	magenta	7
blue	4	white	8

Table 2.6: Colour names and associated colour numbers

Note: For pen plotters, the colour of the pens should be confirmed before submitting a plot file to a plotter.

Using a scalar for a colour number

If a scalar variable is to be used for the colour number, it must have the qualifier `\SCALAR` attached to it. For example:

```
COLR=5
COLOUR COLR\SCALAR
```

CONTOUR

Syntax	`CONTOUR { x y } v nctr { min { inc }}` `CONTOUR\SPECIFIC { x y } v lvls`
Qualifiers	`\SPECIFIC, \INTERPSIZE, \POLAR, \LEGEND, \COLOURS, \PARTIAL,` `\RESET, \CONTINUE, \BORDER, \AXES, \COORDINATES, \AREAS, \VOLUMES`
Defaults	`x = [1;2;3;...] y = [1;2;3;...]` `\NOSPECIFIC, \NOINTERPSIZE, \NOPOLAR, \NOLEGEND \NOCOLOURS,` `\NOPARTIAL, \RESET, \BORDER, \AXES \NOCOORDINATES, \NOAREAS,` `\NOVOLUMES, \NOCONTINUE`
Examples	`CONTOUR X Y Z 10` `CONTOUR M 10 MINVAL` `CONTOUR\SPECIFIC\INTERPSIZE 10 X Y M LEVELS`

The CONTOUR command draws contour lines for either data contained in a regular matrix or a scattered set of points contained in three vectors.

Contour level selection

Syntax	`CONTOUR { x y } v nctr { min { inc }}` `CONTOUR\SPECIFIC { x y } v lvls`

By default, the number of contours, `nctr`, must be provided. If `nctr` > 0 and the increment is not specified, the actual number of contours drawn may not be the same as the number that was asked for, since “nice” contour levels will be selected and the range of values may not be neatly divisible by the requested number. If the minimum is provided, but not the increment, a “nice” value close to `min` will be used instead of the actual data minimum. If the minimum and the increment are both specified, those exact values will be used for the contour levels.

Specific contour levels

Specific contour levels can be requested by using the `\SPECIFIC` qualifier. In this case, the vector `lvls` should contain the desired contour levels.

Exact contour levels

Exact contour levels can be requested in three ways.

- Use the `\SPECIFIC` qualifier and enter a vector containing the desired levels.
- Specify the minimum contour level, `min`, and the contour level increment, `inc`. This

produces a set of equally spaced contour levels,
`[min; min+inc; min+2*inc; ...; min+nctr*inc]`.

- Specify a negative number of contours, `nctr` < 0. This produces a set of equally spaced contour levels, as above, using the actual data minimum and the actual data maximum.

Zooming in

When the minimum and maximum contour levels are to be determined, by default, the entire range of the data is used. If the `\PARTIAL` qualifier is used, the minimum and maximum contour levels will be determined by the region contained within the axes. Thus, to zoom in on a particular region for more detail, pre-set the axis scales, using the SCALES command, before entering the CONTOUR command. Of course, the `\PARTIAL` qualifier does not apply when you request specific contour levels.

Contour level colour

Syntax	`CONTOUR\COLOURS colr { x y } v num { min }` `CONTOUR\INTERP\COLOURS colr ntrp { x y } v num { min }` `CONTOUR\SPECIFIC\COLOURS colr { x y } v lvls` `CONTOUR\SPECIFIC\INTERP\COLOURS colr ntrp { x y } v lvls`

Colour contours can be obtained using the `\COLOUR` qualifier. A colour vector, `colr`, is expected as the first parameter. The length of `colr` should be the same as the number of contours requested.

Contour labels

By default, contours are labeled with the actual contour level, using three significant digits. See the discussion on the `\LEGEND` qualifier for an alternate contour labeling facility.

Contour label size

The size of the contour labels is %LABSIZ. If labels are not desired on the contours, use the SET command to set %LABSIZ to zero before entering the CONTOUR command.

Contour label separation

The separation between contour labels is controlled with CNTSEP or %CNTSEP, which can be changed with the SET command. If %CNTSEP is set, the separation is a percentage of the height of the window, that is, YUWIND-YLWIND. If CNTSEP is set, the separation is expressed in inches or centimeters, depending on the units. The default is %CNTSEP $= 50$.

Saving contour levels and coordinates

The contour levels are automatically stored in a vector named CCONT. If the \COORDINATES qualifier is used, the x and y coordinates of each contour level are stored in matrices named XCNT and YCNT. The number of points stored for each level is the first element of each column. For example, XCNT[1,nc] (=n1) is the number of points making up contour number nc, while XCNT[2:n1+1,nc] and YCNT[2:n1+1,nc] would contain the x and y coordinates of the ncth contour level. These vectors are then available to the user for plotting and/or manipulation. Each time the CONTOUR command is entered, these vectors are emptied and replaced, so if you wish to keep them, they should be renamed or copied into other vectors.

Legend

By default, the contours are labeled with the actual contour level, using three significant digits. If \NOAXES is used, then no legend is allowed.

If the \LEGEND qualifier is used, then the contours are labeled with an integer index and the list of indices corresponding to the actual contour levels, the legend, is plotted along the right side of the axes. If areas and/or volumes are requested, using \AREAS and/or \VOLUMES, these values will also appear in the legend.

If you zoom in on a contour plot, by setting the axis scales beforehand, and you use the \PARTIAL qualifier as well as the \AREAS qualifier, then the areas will be percentages of the area currently showing on the graph. By default, the areas are percentages of the total area.

To add more contours to a contour plot, re-issue the same contour command with the \CONTINUE qualifier and the legend will be continued from where it left off. You must have used the \NORESET qualifier on the previous CONTOUR command if you intend to use \CONTINUE on a succeeding CONTOUR command.

Legend size

The size of the legend characters is LEGSIZ. The value of LEGSIZ, or %LEGSIZ, can be changed with the SET command. The default value of %LEGSIZ is 1.6

Axis relocation

The legend requires the right end of the x-axis to be set to 75% of the window, that is, %XUAXIS is set to 75. The value of %XUAXIS can be changed with the SET command.

By default, %XUAXIS is reset to its former value after the CONTOUR command is finished. If the \NORESET qualifier is used, the axis location will not be reset.

Polar coordinates

By default, the vectors x and y are assumed to represent Cartesian coordinates. If the \POLAR qualifier is used, x and y are assumed to represent polar coordinates, with x the radial component and y the angular component, in degrees. The values are converted internally to rectangular coordinates, and the vectors are returned unchanged.

Axes

By default, axes are drawn for the contour plot. If the contour plot is to be overlayed on an existing set of axes, use the \NOAXES qualifier and no axes will be drawn. The axis scales will be left at their current values.

Scattered points

Syntax	`CONTOUR x y z nctr { min { inc }}` `CONTOUR\SPECIFIC x y z lvls`
Qualifiers	`\SPECIFIC, \POLAR, \LEGEND, \COLOURS, \PARTIAL` `\RESET, \AXES, \COORDINATES`
Defaults	`\NOSPECIFIC, \NOPOLAR, \NOLEGEND, \NOCOLOURS,` `\NOPARTIAL, \RESET, \AXES, \NOCOORDINATES`

If z is a vector, the vectors x and y are assumed to represent a scattered set of coordinates, where z[i] is the altitude corresponding to the coordinate location (x[i],y[i]). The vectors x and y must be entered if z is a vector.

Contours are computed by successive solution of quintic polynomial equations. The irregularly distributed data points are organized as triangles and the partial derivatives at each point are estimated from the function values of the neighboring points.

Areas and volumes cannot be calculated from scattered data.

Matrix data

Syntax	`CONTOUR { x y } v nctr { min { inc }}`
	`CONTOUR\SPECIFIC { x y } v lvls`
Qualifiers	`\SPECIFIC, \INTERPSIZE, \POLAR, \LEGEND, \COLOURS, \PARTIAL`
	`\RESET, \BORDER, \AXES, \COORDINATES, \AREAS, \VOLUMES`
Defaults	`x = [1;2;3;...] y = [1;2;3;...]`
	`\NOSPECIFIC, \NOINTERPSIZE, \NOPOLAR, \NOLEGEND`
	`\NOCOLOURS, \NOPARTIAL, \RESET, \BORDER`
	`\AXES, \NOCOORDINATES, \NOAREAS, \NOVOLUMES`

Suppose that `v` is a matrix which has n columns and m rows. The vectors `x` and `y` are optional, and if entered, are used for scaling the axes. Each matrix element, `v[i,j]`, is associated with the coordinates `(x[j],y[i])`. The length of `x` must be greater than or equal to n and the length of `y` must be greater than or equal to m.

If `x` and `y` are not entered, `x` defaults to the set `[1;2;3;...;`n`]`, and `y` defaults to the set `[1;2;3;...;`m`]`, so that matrix element `m[i,j]` is associated with the coordinates `(j,i)`.

Minimum and maximum contour coordinates

The minimum and maximum x value for each contour are automatically stored in vectors named `CXMIN` and `CXMAX`; the minimum and maximum y value for each contour are automatically stored in vectors named `CYMIN` and `CYMAX`. These vectors are then available to the user for plotting and/or manipulation. Each time the `CONTOUR` command is entered, these vectors are emptied and replaced, so if you wish to keep them, they should be renamed or copied into other vectors.

Volume

If the `\VOLUMES` qualifier is used, the volume contained within each contour is calculated as a percentage of the total volume. The volume percentages are automatically stored in a vector named `CVOLM`. Each time the `CONTOUR` command is entered, this vector is emptied and replaced, so if you wish to keep it, it should be renamed or copied into another vector.

Area

If the `\AREAS` qualifier is used, the area contained within each contour is calculated as a percentage of the total area. The area percentages are automatically stored in a vector named `CAREA`. Each time the `CONTOUR` command is entered, this vector is emptied and replaced, so if you wish to keep it, it should be renamed or copied into another vector.

Area and volume calculation

The areas and volumes are calculated in the following way. Two dimensional, four point linear interpolation is used to calculate a fine mesh overlayed on the matrix, see Figure 2.1. Suppose the matrix has n columns and m rows and the size of the fine mesh is n_1 columns by m_1 rows. The total area is $n_1 \times m_1$ and the total volume is the sum of the interpolated values. Each point of the fine mesh is tested against the contour levels, and if a mesh point has a value greater than the contour level, a one is binned for the area vector and the mesh point value is binned for the volume vector. Finally, the area and volume vectors are normalized by conversion to percentages.

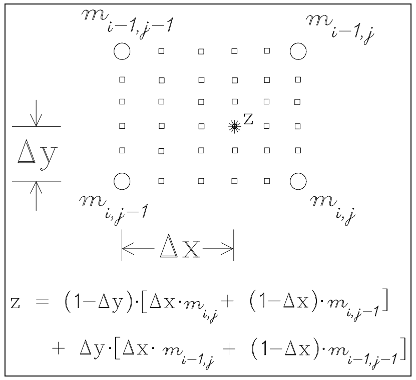


Figure 2.1: Interpolating a fine mesh on the contours of a matrix

Interpolation size

Syntax	`CONTOUR\INTERPSIZE ntrp { x y } v nctr { min { inc }}`
	`CONTOUR\SPECIFIC\INTERPSIZE ntrp { x y } v lvls`

Suppose the matrix has n columns and m rows. The total size of the fine mesh, n_1 columns by m_1 rows, is defined by the following:

$$n_1 = (n-1) \times i_x + 1$$
$$m_1 = (m-1) \times i_y + 1$$

where $i_x - 1$ is the number of interpolation points between matrix points in the x-direction, and $i_y - 1$ is the number of interpolation points between matrix points in the y-direction. The defaults are as below:

$$i_x \left| \begin{array}{rlrcrcr} 10 & \text{if} & & & n & < & 20 \\ 5 & \text{if} & 20 & \leq & n & < & 50 \\ 3 & \text{if} & 50 & \leq & n & < & 100 \\ 2 & \text{if} & 100 & \leq & n & & \end{array} \right. \qquad i_y \left| \begin{array}{rlrcrcr} 10 & \text{if} & & & m & < & 20 \\ 5 & \text{if} & 20 & \leq & m & < & 50 \\ 3 & \text{if} & 50 & \leq & m & < & 100 \\ 2 & \text{if} & 100 & \leq & m & & \end{array} \right.$$

To over-ride these defaults, use the \INTERPSIZE qualifier, and enter `ntrp` as the first parameter. Both i_x and i_y will be set to `ntrp`.

Matrix boundary

By default, the boundary of the matrix is outlined within the axes. If this boundary is not desired, use the \NOBORDER qualifier.

COPY

Syntax	`COPY xin { xin1 ... } xout { xout1 ... } { IFF expression }`
Qualifiers	`\APPEND, \INDEX`
Defaults	`\NOAPPEND, \NOINDEX`
Examples	`COPY X Y Z XX YY ZZ` `COPY\APPEND X XX` `COPY X[1:10] Y[20:11:-1]` `COPY\INDEX X Y Z XX YY ZZ IFF (X>2)`

The `COPY` command copies a subset of vector `xin`, into another vector, `xout`. By default, if `xout` already exists, the `COPY` command overlays the new data on the old.

Multiple input vectors can be entered, but there must be an output vector for each input vector. Vector `xinI` is copied into `xoutI`.

Conditional copy

Syntax `COPY xin1 { xin2 ... } xout1 { xout2 ... } IFF expression`

If the keyword `IFF` is used, an `expression` is expected as the next, the last, parameter. In this case, all the input vectors must have the same length. Index ranges on the input vectors are *not* allowed. The `expression` does not have to involve any of the input vectors, but it must result in a one dimensional array which has the same length as the input vectors.

`xinI[j]` is copied into `xoutI` if and only if the j^{th} element of the `expression` is true. The `expression` is said to be true if its value is non-zero, and false if its value is zero.

By default, the elements of `xinI` are copied in order into `xoutI`, that is, if the `expression` dictates that n elements of `xinI` are to be copied into `xoutI` then these will become the first n elements of `xoutI`.

If the \INDEX qualifier is used, then `xinI[j]` is copied to `xoutI[j]` if and only if the j^{th} element of the `expression` is true. If the j^{th} element of the `expression` is false, `xoutI[j]` is left as is, or is set to zero if `j` is greater than the original length of `xoutI`.

Unconditional copy

Syntax `COPY xin1 { xin2 ... } xout1 { xout2 ... }`

If no expression is entered, then index ranges may be used on the input and/or the output vectors. In some cases, the `COPY` command is equivalent to an assignment. For example:

`COPY XX[10:1:-2] X[1:10:2]` is equivalent to `X[1:10:2]=XX[10:1:-2]`.

Appending with copy

If the \APPEND qualifier is used, the copied elements of `xinI` are appended onto the end of `xoutI`. If `xoutI` does not exist, `COPY\APPEND` is the same as `COPY`.

DCL

Syntax `DCL`

The `DCL` command is only relevant for VMS. The `DCL` command enters DCL mode by spawning a subprocess. To return to the program, type the DCL command `RETURN`. When in DCL mode, any VMS command may be entered, for example, edit a file, run a program.

The first time the `DCL` command is entered, a subprocess is spawned, which can take some time. If the word `RETURN` is typed, the subprocess is not destroyed and a subsequent `DCL` command will attach to this subprocess. Attaching to a subprocess is very fast. If the word `LOGOFF` is typed, the subprocess is destroyed, so that a subsequent `DCL` command will have to spawn a new subprocess.

UNIX equivalent

For UNIX users, just type `control-z` to suspend the program, then type `bg` to put the program into the background. To return to PHYSICA, type `fg`.

DEALIAS

Syntax `DEALIAS ALL`
`DEALIAS aliascommand`

The DEALIAS command allows the user to eliminate aliases that were created with the ALIAS command. If the keyword ALL is entered, all aliases will be eliminated. To display all aliases, enter the ALIAS command with no parameters.

DEFAULTS

Syntax	DEFAULTS
Qualifiers	\INITIALIZE, \WINDOWS
Defaults	initialization file not executed, windows reset
Examples	DEFAULTS DEFAULTS\INIT

The DEFAULTS command resets the original PHYSICA defaults.

Initialization file

If the \INITIALIZE qualifier is used, the initialization script file is executed after the standard defaults have been set. That command script is also executed automatically at the time PHYSICA is run.

It is possible to have individualized sets of PHYSICA defaults by means of initialization script files. Create a script file and assign its name *before* running the program.

VMS: The file assigned to the logical name PHYSICA$INIT is executed.
`$ DEFINE PHYSICA$INIT your_initfile`
You could include this assignment in your DCL login command file.

UNIX: The file assigned to the environment variable PHYSICA_INIT is executed.
`% setenv PHYSICA_INIT your_initfile`
If PHYSICA_INIT is undefined, the file .physicarc in the current directory is executed. If this file doesn't exist, the file $HOME/.physicarc is executed. No further action is taken if this file doesn't exist.

Reset windows

By default, the windows are reset to their original definitions, see Table 2.67 on page 274. If the \-WINDOWS qualifier is used, the windows will be left with their current definitions.

Default values

The PHYSICA keywords and their default values shown in Table 2.24 on page 63. These defaults are the standard GPLOT defaults with the exceptions as listed in Table 2.7.

NLXINC = 2	NLYINC = 2	no plotting symbol
NSXINC = 1	NSYINC = 1	autoscaling on
%XLAXIS = 15	%YLAXIS = 15	
%XUAXIS = 95	%YUAXIS = 90	FONT = TSAN
%XNUMSZ = 3	%YNUMSZ = 3	

Table 2.7: Exceptions to the standard GPLOT defaults

The line types are reset to their original specifications, as shown in Table 2.8 in centimeters. See the SET LINES command on page 228 for information on changing the line type definitions. See the DISPLAY command for information on how to view examples of the line types.

line type	p1	p2	p3
1	0.00	0.00	0.00
2	0.07	0.00	0.00
3	0.50	0.30	0.00
4	0.50	0.30	0.10
5	0.30	0.30	0.00
6	0.30	0.30	0.10
7	0.20	0.20	0.00
8	0.20	0.20	0.05
9	0.05	0.20	0.00
10	0.05	0.30	0.00

Table 2.8: Line type defaults in centimeters

The hatch patterns are reset to their original specifications, as shown in Table 2.9 in centimeters. See the SET HATCH command on page 228 for information on changing the hatch pattern definitions. See the DISPLAY command for information on how to view examples of the hatch patterns.

DENSITY

Syntax	DENSITY { x y } v
Qualifiers	\POLAR, \PARTIAL, \DERIV, \PROFILE, \XPROFILE, \YPROFILE, \BORDER, \AXES, \LOG
Defaults	\-POLAR, \-PARTIAL, \-DERIV, \-PROFILE, \BORDER, \AXES, \-LOG
Examples	DENSITY M DENSITY X Y Z

pattern number	*spacings* 1	2	angle
1	0.01		0
2	0.01		90
3	0.05		0
4	0.05		90
5	0.10		0
6	0.10		90
7	0.20		45
8	0.20		−45
9	0.20	0.10	45
10	0.20	0.10	−45

Table 2.9: Hatch pattern defaults in centimeters

The DENSITY command produces a density plot for either data contained in a matrix or a scattered set of points contained in three vectors.

If the \LOG qualifier is used, the base 10 logarithm of the data is used for the plot. If the \LEGEND qualifier is also used, the scales on the legend will be in the form 10^n. The data is divided into levels whose boundaries are always integral powers of 10. The number of levels will vary depending on the original data.

Density plot types

There are five types of density plot available. The default, requiring no special qualifier, is solid filled regions in colour. Other types are chosen by using the appropriate qualifier. Refer to Table 2.10.

density plot types	*required qualifier*
solid filled regions in colour	none (default)
random points	\RANDOM
dithering with points (grey scales)	\POINTS
diffusion with points (grey scales)	\DIFFUSION
scaled rectangles	\BOXES

Table 2.10: Density plot types and their required qualifiers

Axes

By default, axes are drawn for the density plot. If the density plot is to be overlayed on an

existing set of axes, use the \NOAXES qualifier and no axes will be drawn. The axis scales will be left at their current values.

Matrix boundary

The \BORDER qualifier is valid only if matrix data is entered. By default, the rectangular boundary of the matrix is outlined within the axes. If you do not want this boundary to be drawn, use the \NOBORDER qualifier.

Zooming in

By default, the entire range of possible density levels will be used to determine the minimum and maximum density levels. If the \PARTIAL qualifier is used, the minimum and maximum density levels will be determined by the region contained within the axes. To zoom in on a particular region for more detail, pre-set the axis scales, using the SCALES command, before entering the DENSITY command.

Derivatives

If the \DERIV qualifier is used, the derivative of the data is used for plotting instead of the raw data itself.

Profiles

The qualifiers \PROFILE, \XPROFILE and \YPROFILE are valid only if matrix data is entered.

If the \XPROFILE qualifier is used, the columns of the matrix are summed, the sums are normalized to be between 0 and 1, and a histogram of the normalized sums is drawn horizontally across the top of the graph.

If the \YPROFILE qualifier is used, the rows of the matrix are summed, the sums are normalized to be between 0 and 1, and a histogram of the normalized sums is drawn vertically along the right side of the graph.

If the \PROFILE qualifier is used, both the horizontal and vertical profiles are drawn.

When a profile is drawn, the axis borders must also be set to allow space for the profiles, that is, %XUAXIS is set to 65% if a legend is present or 85% if there is no legend, and %YUAXIS is set to 80.

By default, %XUAXIS and %YUAXIS are reset to their former values. If the \NORESET qualifier is used, the axis locations are not reset.

Polar coordinates

By default, the vectors x and y are assumed to represent Cartesian coordinates. If the \POLAR qualifier is used, x and y are assumed to represent polar coordinates, with x the radial component and y the angular component, in degrees. The values are converted internally to rectangular coordinates, and the vectors are returned unchanged.

Solid filled regions

Syntax	`DENSITY { x y } v { p1 p2 }`
Qualifiers	`\POLAR, \LEGEND, \PARTIAL, \DERIV, \PROFILE, \XPROFILE, \YPROFILE, \BORDER, \HISTOGRAM, \RESET, \AXES`
Defaults	if v is a matrix: `x = [1;2;3;...], y = [1;2;3;...],` `p1 = 0, p2 = 1, \NOPOLAR, \BORDER, \AXES, \NOLEGEND, \NOPROFILE, \RESET`

Solid filled regions in colour is the default density type. No qualifiers are needed to produce this type of drawing. The range of values of the matrix is divided into eight (8) equal levels, and a different colour is associated with each level. By default, a value is interpolated at each pixel location within the matrix region so as to give smoothly joined regions.

PostScript output

For PostScript output, set the POSTRES keyword to the appropriate resolution for your hard-copy device, using the SET command. Use a combination of line thickness and resolution for a quicker resulting picture. For example, the Lexmark inkjet printer has a resolution of 360 dpi. A "good" picture can be obtained with POSTRES $= 180$ and LINTHK $= 2$.

Input variables

If v is a vector, the parameters x and y are expected and must be vectors. x and y are assumed to represent a scattered set of points, where v[i] is the altitude corresponding to the location (x[i],y[i]). A matrix is interpolated on these scattered points by means of a Thiessen triangulation of the plane. The minimum length of the three vectors x, y, and v will be used.

If v is a matrix, the parameters x and y default to [1;2;3;...], but if entered they must be vectors. Each matrix element, m[i,j], is associated with the coordinates (x[j],y[i]). The length of x must be greater than or equal to the number of columns of v and the length of y must be greater than or equal to the number of rows. The vectors x and y are used for scaling the axes.

Not interpolated solid fill

If the \HISTOGRAM qualifier is used, each data location is represented by a rectangle of colour, centred on the data location. The regions are not smoothly joined.

Changing the range of values

The optional parameters p1 and p2 can be used to broaden or shrink the range of data values. If v_{max} is the maximum value of the data and v_{min} is the minimum value of the data, the full colour range will be from a minimum of $min = \mathtt{p1} \times (v_{max} - v_{min}) + v_{min}$ to a maximum of $max = \mathtt{p2} \times (v_{max} - v_{min}) + v_{min}$. If v is a data value and if $v < \mathtt{p1} \times (v_{max} - v_{min}) + v_{min}$, that data value is treated as v_{min}. If $v > \mathtt{p1} \times (v_{max} - v_{min}) + v_{min}$, that data value is treated as v_{max}. The default values are: $\mathtt{p1} = 0$ and $\mathtt{p2} = 1$.

Legend

If the \LEGEND qualifier is used, a legend is drawn along the right side of the axes. The legend requires the right end of the x-axis to be set to 75% of the window, that is, %XUAXIS is set to 75.

When a y-profile is drawn, using the \PROFILE qualifier or the \YPROFILE qualifier, the right edge of the axis box must allow space for the profile as well as a possible legend. If a y-legend profile and a legend are present, then %XUAXIS is set to 65. If a y-legend profile is present but not a legend, then %XUAXIS is set to 85.

The value of %XUAXIS can be changed with the SET command. By default, %XUAXIS is reset to its former value after the DENSITY command. If the \NORESET qualifier is used, the axis location is not reset.

The numeric legend entries are written using the LEGFRMT format and with height given by LEGSIZ, both of which are changed with the SET command. The default values are: LEGFRMT = 1PE10.3 and %LEGSIZ = 1.6.

Random points

Syntax	`DENSITY\RANDOM { x y } v { p1 p2 }`
Qualifiers:	`\POLAR, \PARTIAL, \DERIV, \COLOUR, \PROFILE, \XPROFILE, \YPROFILE, \BORDER, \RESET, \AXES`
Defaults	if v is a matrix: `x = [1;2;3;...], y = [1;2;3;...],` `\NOPOLAR, \BORDER, \AXES, \NOPROFILE, \RESET, \NOCOLOUR` $\mathtt{p1} = 0$, $\mathtt{p2} = 1$

To obtain the random points type of density plot, use the `\RANDOM` qualifier.

A value is interpolated at every pixel location within the data region and the value is then normalized to lie between 0 and 1. This normalized value is compared to a randomly generated number. That pixel location is lit up if the square of the normalized value is greater than the random number.

PostScript output

For PostScript output, set the `POSTRES` keyword to the appropriate resolution for your hardcopy device, using the `SET` command. Use a combination of line thickness and resolution for a quicker resulting picture. For example, the Lexmark inkjet printer has a resolution of 360 dpi. A "good" picture can be obtained with `POSTRES` $=180$ and `LINTHK` $=2$.

Input variables

If `v` is a vector, the parameters `x` and `y` are expected and must be vectors. `x` and `y` are assumed to represent a scattered set of points, where `v[i]` is the altitude corresponding to the location `(x[i],y[i])`. A matrix is interpolated on these scattered points by means of a Thiessen triangulation of the plane. The minimum length of the three vectors `x`, `y`, and `v` will be used.

If `v` is a matrix, the parameters `x` and `y` default to `[1;2;3;...]`, but if entered they must be vectors. Each matrix element, `m[i,j]`, is associated with the coordinates `(x[j],y[i])`. The length of `x` must be greater than or equal to the number of columns of `v` and the length of `y` must be greater than or equal to the number of rows. The vectors `x` and `y` are used for scaling the axes.

Random points in colour

If the `\COLOUR` qualifier is used with `\RANDOM`, the range of data values is divided into eight equal levels and a different colour is associated with each level. The pixel is lit in the colour corresponding to the relative size of the interpolated value.

Changing the range of values

The optional parameters `p1` and `p2` can be used to broaden or shrink the range of data values. If v_{max} is the maximum value of the data and v_{min} is the minimum value of the data, the full colour range will be from a minimum of $min = \texttt{p1} \times (v_{max} - v_{min}) + v_{min}$ to a maximum of $max = \texttt{p2} \times (v_{max} - v_{min}) + v_{min}$. If v is a data value and if $v < \texttt{p1} \times (v_{max} - v_{min}) + v_{min}$, that data value is treated as v_{min}. If $v > \texttt{p1} \times (v_{max} - v_{min}) + v_{min}$, that data value is treated as v_{max}. The default values are: $\texttt{p1} = 0$ and $\texttt{p2} = 1$.

Example

```
R=MOD([0:143],4)+1
SORT\UP R
T=MOD([0:143],36)*10
DENSITY\RANDOM R*COSD(T) R*SIND(T) EXP(-R/2)*COSD(180*(R-1))
```

Diffusion

Syntax	`DENSITY\DIFFUSION { x y } v { p1 p2 }`
Qualifiers	`\POLAR, \PARTIAL, \DERIV, \PROFILE, \XPROFILE, \YPROFILE, \BORDER, \RESET, \AXES`
Defaults	if `v` is a matrix: `x = [1;2;3;...], y = [1;2;3;...]`, `\NOPOLAR, \BORDER, \AXES, \NOPROFILE, \RESET` `p1` $=0$, `p2` $=1$

To obtain the diffusion type of density plot, use the `\DIFFUSION` qualifier.

Diffusion is a form of digital halftoning. A threshold is fixed at $\frac{1}{2}$. Data values are interpolated at each pixel location, and then normalized to be between 0 (white) and 1 (black). The resulting binary output value is compared with the original grey level value. The difference is called the error for that location. The signal consisting of past error values is passed through an error filter to produce a correction factor to be added to future input values. Thus, errors are diffused over a weighted neighborhood.

PostScript output

For PostScript output, set the `POSTRES` keyword to the appropriate resolution for your hardcopy device, using the `SET` command. Use a combination of line thickness and resolution for a quicker resulting picture. For example, the Lexmark InkJet printer has a resolution of 360 dpi. A "good" picture can be obtained with `POSTRES` $=180$ and `LINTHK` $=2$.

Input variables

If `v` is a vector, the parameters `x` and `y` are expected and must be vectors. `x` and `y` are assumed to represent a scattered set of points, where `v[i]` is the altitude corresponding to the location `(x[i],y[i])`. A matrix is interpolated on these scattered points by means of a Thiessen triangulation of the plane. The minimum length of the three vectors `x`, `y`, and `v` will be used.

If `v` is a matrix, the parameters `x` and `y` default to `[1;2;3;...]`, but if entered they must be vectors. Each matrix element, `m[i,j]`, is associated with the coordinates `(x[j],y[i])`. The length of `x` must be greater than or equal to the number of columns of `v` and the length of `y` must be greater than or equal to the number of rows. The vectors `x` and `y` are used for scaling the axes.

Changing the range of values

The optional parameters `p1` and `p2` can be used to broaden or shrink the range of data values. If v_{max} is the maximum value of the data and v_{min} is the minimum value of the data, the full colour range will be from a minimum of $min = \texttt{p1} \times (v_{max} - v_{min}) + v_{min}$ to a maximum of $max = \texttt{p2} \times (v_{max} - v_{min}) + v_{min}$. If v is a data value and if $v < \texttt{p1} \times (v_{max} - v_{min}) + v_{min}$, that data value is treated as v_{min}. If $v > \texttt{p1} \times (v_{max} - v_{min}) + v_{min}$, that data value is treated as v_{max}. The default values are: $\texttt{p1} = 0$ and $\texttt{p2} = 1$.

Example

```
R=MOD([0:143],4)+1
SORT\UP R
T=MOD([0:143],36)*10
DENSITY\DIFFUSION R*COSD(T) R*SIND(T) EXP(-R/2)*COSD(180*(R-1))
```

Dithering patterns

Syntax
```
DENSITY\POINTS { x y } v { p1 p2 }
DENSITY\POINTS\DITHER d { x y } v { p1 p2 }
DENSITY\POINTS\LEVELS lvl { x y } v { p1 p2 }
DENSITY\POINTS\LEVELS\DITHER d lvl { x y } v { p1 p2 }
```

Qualifiers `\POLAR, \LEGEND, \PARTIAL, \DERIV, \PROFILE, \XPROFILE, \YPROFILE, \BORDER, \RESET, \AXES, \DITHER, \CONTOURS, \LEVELS, \AREAS, \VOLUMES, \LINES, \EQUALLY_SPACED`

Defaults if `v` is a matrix: `x = [1;2;3;...], y = [1;2;3;...]`,
`d = [ 1;1; 2;1; 2;2; 3;2; 3;3; 4;3; 4;4; 5;5; 6;6; 0;0 ]`,
`\NOPOLAR, \BORDER, \NOLEGEND, \NOPROFILE, \RESET`
$\texttt{p1} = 0$, $\texttt{p2} = 1$, ten equally spaced contour levels

To obtain the dithering pattern type of density plot, use the `\POINTS` qualifier.

By default, the range of data values is divided into ten (10) equally spaced levels and a different dithering pattern is associated with each level. A value is interpolated at every pixel location within the bounds of the data region to determine the level for that point. The dithering pattern for that level then determines whether that pixel is to be lit up. Thus, the

boundaries of the data are divided up into different dithering pattern regions.

PostScript output

For PostScript output, set the `POSTRES` keyword to the appropriate resolution for your hardcopy device, using the `SET` command. Use a combination of line thickness and resolution for a quicker resulting picture. For example, the Lexmark inkjet printer has a resolution of 360 dpi. A “good” picture can be obtained with $\texttt{POSTRES} = 180$ and $\texttt{LINTHK} = 2$.

Input variables

If `v` is a vector, the parameters `x` and `y` are expected and must be vectors. `x` and `y` are assumed to represent a scattered set of points, where `v[i]` is the altitude corresponding to the location `(x[i],y[i])`. A matrix is interpolated on these scattered points by means of a Thiessen triangulation of the plane. The minimum length of the three vectors `x`, `y`, and `v` will be used.

If `v` is a matrix, the parameters `x` and `y` default to `[1;2;3;...]`, but, if entered, they must be vectors. Each matrix element, `m[i,j]`, is associated with the coordinates `(x[j],y[i])`. The length of `x` must be greater than or equal to the number of columns of `v` and the length of `y` must be greater than or equal to the number of rows. The vectors `x` and `y` are used for scaling the axes.

Changing the range of values

The optional parameters `p1` and `p2` can be used to broaden or shrink the range of data values. If v_{max} is the maximum value of the data and v_{min} is the minimum value of the data, the full colour range will be from a minimum of $min = \texttt{p1} \times (v_{max} - v_{min}) + v_{min}$ to a maximum of $max = \texttt{p2} \times (v_{max} - v_{min}) + v_{min}$. If v is a data value and if $v < \texttt{p1} \times (v_{max} - v_{min}) + v_{min}$, that data value is treated as v_{min}. If $v > \texttt{p1} \times (v_{max} - v_{min}) + v_{min}$, that data value is treated as v_{max}. The default values are: $\texttt{p1} = 0$ and $\texttt{p2} = 1$.

Dithering pattern definition

The default dithering pattern vector is:

```
[ 1;1; 2;1; 2;2; 3;2; 3;3; 4;3; 4;4; 5;5; 6;6; 0;0 ]
```

A user defined dithering pattern can be entered by using the `\DITHER` qualifier and entering the dithering pattern vector, `d`, as the first parameter.

The dithering pattern is determined by pairs of numbers from `d`, so the number of dithering patterns defined by d is $\frac{1}{2}$ the length of d.

For pattern number i, every d[$2\times i-1$]th pixel is lit up horizontally, and every d[$2\times i$]th pixel is lit up vertically. For example, if d[1] = 1 and d[2] = 1, then for level 1 every pixel is lit up, while if d[3] = 2 and d[4] = 3, then for level 2 every second pixel is lit up horizontally and every third pixel is lit up vertically.

Legend

If the \LEGEND qualifier is used, a legend is drawn along the right side of the axes. The legend requires the right end of the x-axis to be set to 75% of the window, that is, %XUAXIS is set to 75.

When a y-profile is drawn, using the \PROFILE qualifier or the \YPROFILE qualifier, the right edge of the axis box must allow space for the profile as well as a possible legend. If a y-legend profile and a legend are present, then %XUAXIS is set to 65. If a y-legend profile is present but not a legend, then %XUAXIS is set to 85.

The value of %XUAXIS can be changed with the SET command. By default, %XUAXIS is reset to its former value after the DENSITY command. If the \NORESET qualifier is used, the axis location is not reset.

The numeric legend entries are written using the LEGFRMT format and with height given by LEGSIZ, both of which are changed with the SET command. The default values are: LEGFRMT = 1PE10.3 and %LEGSIZ = 1.6.

Contours

By default, a contour line is drawn around the boundary of each dithering pattern region. If the \NOLINES qualifier is used, then these contour lines will not be drawn.

If the qualifier \CONTOURS is used, an automatically created vector named DENS$CONT will be made which will contain the boundary values of each region. If there are N regions, the length of DENS$CONT will be $N+1$.

User specified contour levels

A specific set of contour levels can be entered by using \LEVELS and entering a vector of contour level values, lvl, as the first parameter, unless the \DITHER qualifier is also used, in which case the contour level vector should be the second parameter. If both are used, and the length of the dithering vector is N, the length of the level vector must be $\frac{N}{2}-1$. Suppose that v_{min} and v_{max} are the minimum and maximum of the data v. The level vector must be

strictly monotonically increasing, with lvl[1] $> v_{min}$ and lvl[#] $< v_{max}$.

The \EQUALLY_SPACED qualifier only applies to the case of a dithering type density plot with legend, where the user supplies the contour levels, for example:

```
DENSITY\POINTS\LEVELS\LEGEND\EQUALLY_SPACED lvl x y m
```

If the \EQUALLY_SPACED qualifier is used, the legend boxes will all be the same size, irregardless of the values specified in the levels vector, lvl.

Areas and volumes

If the \AREAS qualifier is used, an automatically created vector named DENS$AREA will be made which will contain the percentage areas contained within each region. If the \VOLUMES qualifier is used, an automatically created vector named DENS$VOLM will be made which will contain the percentage volumes contained within each region. If there are N regions, the length of DENS$AREA and DENS$VOLM will both be N. Also, the sum of the elements of each of these vectors will always be 100, that is, sum(DENS$AREA[j],j,1:N) $= 100$.

Example

```
R=MOD([0:143],4)+1
SORT\UP R
T=MOD([0:143],36)*10
D=[ 1;1; 2;2; 4;4; 7;7; 11;11; 0;0 ]
DENSITY\LEGEND\POINTS\DITHER D R*COSD(T) R*SIND(T) EXP(-R/2)*COSD(180*(R-1))
```

Boxes

Syntax	DENSITY\BOXES { x y } v { p1 p2 { q1 q2 { r }}}
Qualifiers	\POLAR, \PARTIAL, \DERIV, \PROFILE, \XPROFILE, \YPROFILE, \BORDER, \RESET, \AXES
Defaults	if v is a matrix: x = [1;2;3;...], y = [1;2;3;...], p1 = 0, p2 = 1, q1 = 0, q2 = 1, r = 1, \NOPOLAR, \BORDER, \AXES, \NOPROFILE, \RESET

To obtain the scaled rectangles type of density plot, use the \BOXES qualifier.

Input variables

If v is a vector, the parameters x and y are expected and must be vectors. x and y are assumed to represent a scattered set of points. A box is drawn, centred at location (x[i],y[i]) with relative size determined by v[i]. No internal matrix is interpolated with the scaled rectangle type of density plot. The minimum length of the three vectors x, y, and v will be used.

If v is a matrix, the parameters x and y default to [1;2;3;...], but if entered they must be vectors. A box is drawn, centred at location (x[j],y[i]) with relative size determined by v[i,j]. The length of x must be greater than or equal to the number of columns of v and the length of y must be greater than or equal to the number of rows.

Delimiting the range of values

The optional parameters p1 and p2 can be used to select a window of values from within the box size range, min to max, as defined above. Suppose that v is the data value at (x,y). A box is drawn at (x,y) if and only if p1 $< \frac{v-min}{max-min} <$ p2. The default values are: p1 = 0 and p2 = 1.

Accentuating a range of values

The optional parameters q1 and q2 can be used to accentuate a range of values. If v_{max} is the maximum value of the data and v_{min} is the minimum value of the data, the full box size range will be from a minimum of $min = \texttt{q1} \times (v_{max} - v_{min}) + v_{min}$ to a maximum of $max = \texttt{q2} \times (v_{max} - v_{min}) + v_{min}$. The default values are: q1 = 0 and q2 = 1.

Box size scale factor

The optional parameter r is a scale factor which controls the size of the boxes. For each box, the width and height is multiplied by r. The default value is: r = 1.

Filled boxes

The boxes can be filled. Use the SET FILL command to change the fill type and pattern. See Table 2.55 on page 233 for a description of the interpretations of the FILL keyword.

Examples

The following script produces Figure 2.2.

```
X=[  1;  0;  1;  0;   .2; .3;   .5;  .8]
Y=[  5;  5;  0;  0;    1;1.5;  2.5;   4]
Z=[ 10; 10; 10; 10; -100; 10; -100; 500]
GRID\XYOUT X Y Z M XOUT YOUT
DENSITY\BOXES\PROFILES XOUT YOUT M
```

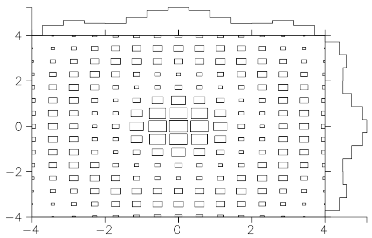



Figure 2.2: An example of a box type density plot with both x and y profiles

The following script produces Figure 2.3.

```
X=[  1;  0;  1;  0;   .2; .3;   .5;  .8]
Y=[  5;  5;  0;  0;    1;1.5;  2.5;   4]
Z=[ 10; 10; 10; 10; -100; 10; -100; 500]
GRID\XYOUT X Y Z M XOUT YOUT
WINDOW 5
LABEL\XAXIS 'DENSITY\BOXES XOUT YOUT M 0 1 0 1'
DENSITY\BOXES XOUT YOUT M 0 1 0 1
WINDOW 7
LABEL\XAXIS 'DENSITY\BOXES XOUT YOUT M .5 1 0 1'
DENSITY\BOXES XOUT YOUT M .5 1 0 1
WINDOW 6
LABEL\XAXIS 'DENSITY\BOXES XOUT YOUT M 0 1 .5 1'
DENSITY\BOXES XOUT YOUT M 0 1 .5 1
WINDOW 8
LABEL\XAXIS 'DENSITY\BOXES XOUT YOUT M .5 1 .5 1'
DENSITY\BOXES XOUT YOUT M .5 1 .5 1
```

The following script produces Figure 2.4.

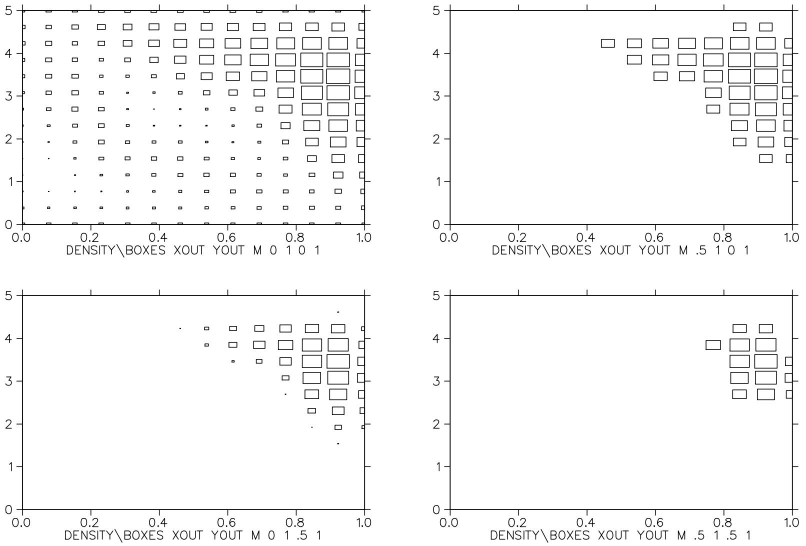


Figure 2.3: Examples of box type density plot with accentuated and delimited values


```
X=[  1;  0;  1;  0;   .2; .3;   .5;  .8]
Y=[  5;  5;  0;  0;    1;1.5;  2.5;   4]
Z=[ 10; 20; 30; 40;   50; 40;   70;  30]
WINDOW 5
LABEL\XAXIS 'density\boxes x y z'
DENSITY\BOXES X Y Z
WINDOW 7
GRID\XYOUT X Y Z M XOUT YOUT
LABEL\XAXIS 'density\boxes xout yout m'
DENSITY\BOXES XOUT YOUT M
```

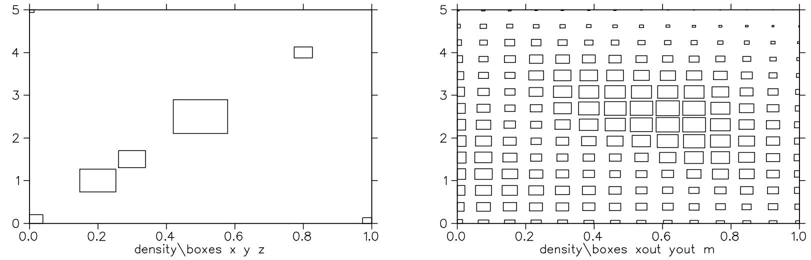


Figure 2.4: Box type density plots with scattered points and with a matrix

DESTROY

Syntax	`DESTROY v1 { v2 ... } { IFF expression }`
Qualifier	`\EXPAND`
Default	`\-EXPAND`
Examples	`DESTROY X Y Z A B C M1 M2 M3 T1 T2 T3` `DESTROY *V M1 M2 *T` `DESTROY X Y Z IFF (X>=2)&(X<=4)` `DESTROY X[1:10] Y[2:20:2] T[3][A:B] Z`

The DESTROY command eliminates subsets of vectors or string variables, or it destroys scalars or matrices. Up to twenty-nine (29) variable names may be entered.

Unconditional

Syntax	`DESTROY v1 { v2 ... }`
Qualifier	`\EXPAND`
Default	`\-EXPAND`

When scalars or matrices are entered, they are completely destroyed. Index ranges are not allowed on scalars or matrices. These variables are simply eliminated. By default, variable names are not constructed or expanded.

If vectors are entered with no index ranges, they are entirely eliminated. Subsets of vectors or of string variables may be eliminated by including an index range.

Expand names

By using the `\EXPAND` qualifier, you can enter names that must be constructed, or string variables that can be expanded.

For example, suppose that scalar `I` has the value 2, and the string variable `TXT` has the value `'X2'`.

`DESTROY\EXPAND 'X'//RCHAR(I)`	will destroy `X2`
`DESTROY TXT`	will destroy `TXT`
`DESTROY\EXPAND TXT`	will destroy `X2`

Index ranges

Index ranges are *not* allowed on scalars or matrices. Index ranges are *not* allowed if an expression is entered.

Subsets of vectors or string variables may be eliminated by including an index range. For example:

```
DESTROY X[5:25:5] Y Z T[1:10]
```

will eliminate elements 5, 10, 15, 20, and 25 from vector `X` and eliminate characters 1 through 10 of string variable `T`. The variables `X` and `T` will be compressed. Variables `Y` and `Z` will be entirely destroyed.

Classes of variables

There are various keywords to simplify the elimination of groups of variables. Refer to Table 2.11. These keywords cannot be used with an expression, but they can be used along with specific variable names.

Conditional

Syntax `DESTROY v1 { v2 ... } IFF expression`

keyword	*result*
`*`	all variables will be destroyed
`*V`	all vectors will be destroyed
`*S`	all scalars will be destroyed
`*M`	all matrices will be destroyed
`*T`	all string variables will be destroyed

Table 2.11: Keywords used to destroy entire classes of variables

If the keyword `IFF` is used, an `expression` is expected as the next, the last, parameter. Elements of *vectors only* can be eliminated conditional on the value of an `expression`. Matrices, scalars and string variables are not allowed, and index ranges on the input vectors are also *not* allowed with a conditional `expression`. The `expression` does not have to involve any of the input vectors, but it must result in a one dimensional array which has the same length as the input vectors.

`vI[j]` is eliminated if and only if the j^{th} element of the `expression` is true. The `expression` is said to be true if its value is non-zero, and false if its value is zero.

Examples

```
DESTROY X[5:25:5] Y Z *M T[1:10] TXT[3][2:5]
```

will destroy elements 5, 10, 15, 20, and 25 from vector `X`; eliminate characters 1 through 10 of string variable `T`; and eliminate characters 2 through 5 of the third string of array string variable `TXT`. The variables `X`, `T`, and `TXT[3]` will be compressed. Variables `Y` and `Z` will be entirely destroyed. All matrices will be eliminated.

It is possible to delete from a vector all elements that have a value between `M` and `N`, assuming that `M` and `N` are scalars.

```
DESTROY X IFF (X>M)&(X<N)
```

The expression is evaluated for each element of `X` and if the expression is true, the corresponding element is deleted from the vector `X`. Other vectors can also be entered. For example:

```
DESTROY X Y Z IFF (X>M)&(X<N)
```

If the j^{th} element of the expression is true, then elements `X[j]`, `Y[j]` and `Z[j]` will be deleted. In this way, sets of data can be kept together.

DEVICE

Syntax	`DEVICE keyword`
Defaults	initial graphics hardcopy device: HP LaserJet 150 dpi
Examples	`DEVICE HPLASER 300` `DEVICE HPPLOTTER` `DEVICE\COLOUR POSTSCRIPT` `DEVICE\GREY POSTSCRIPT A`

The `DEVICE` command is used to select a graphics hardcopy output device. If a device type is chosen, as opposed to entering `OFF` or `ON`, then the graphics will be cleared. The initial default graphics hardcopy device is an HP LaserJet bitmap at 150 dpi resolution.

The graphics hardcopy device determines the world coordinate plotting units. The device should be chosen before opening an `EDGR` file.

ON and OFF

If the user is not interested in graphics hardcopy, it is recommended that hardcopy generation be turned `OFF`, since graphics on the monitor is noticeably faster when a hardcopy is not made.

If `DEVICE OFF` is entered, output to the hardcopy bitmap or file will be disabled. The graphics will not be cleared, but, while the output is disabled, no hardcopy output will be available. To re-enable output to a previously chosen device type, use the `DEVICE ON` command. The device type that was previously chosen will again be available. The graphics will not be cleared.

Device keywords

If the `DEVICE` command is entered with no parameters, the current graphics hardcopy device is displayed. A table of devices and their corresponding code numbers is also displayed, and you can then enter your choice, either by name or by code number.


```
bitmap                        vector                          display
devices                       plotters                        files
----------------------------- ------------------------------- ----------
HPLaserJet {100|150|300} 1-3 HPPlotter {A|B|C|D|E}     12-16  VT640  36
Inkjet {1|2|3|4|5}        4-8 Houston {A|B|C|D|E}       17-21  VT241  37
HPThinkJet                  9 LN03+                     22     Cit467 38
La100                      10 Imagen                    23     TK4010 39
Printronix                 11 GKS                       24     TK4107 40
                              PostScript {A|B|C|D|E|A4} 25-30  PT100G 41
                              Roland {A|B|C|D|E}        31-35  Seiko  42
```

HPLaserJet

Syntax	`DEVICE HPLASERJET { dpi }`
Default	`dpi` = 150

The optional parameter, `dpi`, is only applicable with `HPLaserJet`. It controls the resolution of the hardcopy, specifying the dots per inch. The valid resolutions are: `dpi = 100, 150, 300`. If `dpi` is not entered, `150` is assumed.

Refer to Table 2.12 to see the plotting units for the `HPLaserJet` bitmap graphics hardcopy devices.

orientation	*units*	*horizontal*	*vertical*
`LANDSCAPE`	centimeters	27.94	21.59
	inches	11.00	8.50
`PORTRAIT`	centimeters	21.59	27.94
	inches	8.50	11.00

Table 2.12: Plotting units for `HPLASERJET` devices

InkJet

Syntax	`DEVICE INKJET { np }`
Default	`np` = 1

The HP PaintJet and the LJ250 are colour bitmap devices that allow for a single plot to cover from one (1) to a maximum of five (5) continuous pages. To choose the number of pages, use the optional parameter `np`. If graphics is being monitored on a colour monitor, the colours used on the `INKJET` will be the same as on the monitor, except that black and white are interchanged.

Refer to Table 2.13 to see the plotting units for the InkJet bitmap graphics hardcopy devices.

		pages = 1		pages > 1	
orientation	*units*	*horizontal*	*vertical*	*horizontal*	*vertical*
LANDSCAPE	centimeters	26.67	19.05	27.94	20.32*np
	inches	10.50	7.50	11.00	8.00*np
PORTRAIT	centimeters	19.05	26.67	20.32*np	27.94
	inches	7.50	10.50	8.00*np	11.00

Table 2.13: Plotting units for INKJET devices

Other bitmap devices

Syntax DEVICE PRINTRONIX
DEVICE LA100
DEVICE THINKJET

The PRINTRONIX, LA100, and THINKJET are bitmap devices.

Refer to Table 2.14 to see the plotting units for the PRINTRONIX, LA100, and THINKJET bitmap graphics hardcopy devices.

orientation	*units*	*horizontal*	*vertical*
LANDSCAPE	centimeters	25.00	19.00
	inches	9.84	7.48
PORTRAIT	centimeters	19.00	25.00
	inches	7.48	9.84

Table 2.14: Plotting units for PRINTRONIX, LA100, and THINKJET devices

PostScript devices

Syntax DEVICE POSTSCRIPT { A | B | C | D | E | A4 }
Qualifiers \FLIP, \COLOUR, \GREY
Defaults \-FLIP, \-COLOUR, \-GREY, paper size A

The optional parameter refers to paper sizes. Refer to Table 2.15. To see the plotting units for the PostScript devices, refer to Table 2.16.

Resolution

paper size	*centimeters*	*inches*
A	21.59 × 27.64	8.50 × 11.00
B	27.94 × 43.18	11.00 × 17.00
C	43.18 × 55.88	17.00 × 22.00
D	55.88 × 86.36	22.00 × 34.00
E	86.36 × 111.76	34.00 × 44.00
A4	21.00 × 29.70	8.27 × 11.69

Table 2.15: PostScript paper sizes

The resolution of your PostScript hardcopy output can be changed with the SET POSTRES command. The default value for POSTRES is 180 dpi (dots per inch). This applies to dot filled text characters and to dot types of DENSITY plots.

Upside down drawings

If the \FLIP qualifier is used, the drawing will come out upside down on the paper. This is not mirror image. This feature is included to facilitate the insertion of PostScript plots into TEX or LATEX documents.

Colour

The \COLOUR qualifier only applies to PostScript output. The \COLOUR qualifier means colour changes will be inserted into the PostScript output. The default is \-COLOUR.

Grey scale

The \GREY qualifier only applies to PostScript output. The \GREY qualifier means colour changes will be inserted into the PostScript output as grey scales. The default is \-GREY.

Pen plotters

Syntax DEVICE HPPLOTTER { A | B | C | D | E }
DEVICE HOUSTON { A | B | C | D | E }
DEVICE ROLAND { A | B | C | D | E }
Defaults paper size A

The optional parameter refers to paper sizes. Refer to Table 2.17.

Refer to Table 2.18 to see the plotting units for the pen plotter type graphics hardcopy

paper size	units	LANDSCAPE horizontal	LANDSCAPE vertical	PORTRAIT horizontal	PORTRAIT vertical
A	centimeters	25.00	19.00	19.00	25.00
	inches	9.84	7.48	7.48	9.84
B	centimeters	40.64	25.40	25.40	40.64
	inches	16.00	10.00	10.00	16.00
C	centimeters	53.34	40.64	40.64	53.34
	inches	21.00	16.00	16.00	21.00
D	centimeters	83.82	53.34	53.34	83.82
	inches	33.00	21.00	21.00	33.00
E	centimeters	109.22	83.82	83.82	109.22
	inches	43.00	33.00	33.00	43.00
A4	centimeters	27.16	18.46	18.46	27.16
	inches	10.69	7.27	7.27	10.69

Table 2.16: Plotting units for POSTSCRIPT devices

paper size	centimeters	inches
A	21.59×27.64	8.5×11.0
B	27.94×43.18	11.0×17.0
C	43.18×55.88	17.0×22.0
D	55.88×86.36	22.0×34.0
E	86.36×111.76	34.0×44.0

Table 2.17: Pen plotter paper sizes

devices.

paper size	units	LANDSCAPE horizontal	LANDSCAPE vertical	PORTRAIT horizontal	PORTRAIT vertical
A	centimeters	25.00	19.00	19.00	25.00
	inches	9.84	7.48	7.48	9.84
B	centimeters	40.64	25.40	25.40	40.64
	inches	16.00	10.00	10.00	16.00
C	centimeters	53.34	40.64	40.64	53.34
	inches	21.00	16.00	16.00	21.00
D	centimeters	83.82	53.34	53.34	83.82
	inches	33.00	21.00	21.00	33.00
E	centimeters	109.22	83.82	83.82	109.22
	inches	43.00	33.00	33.00	43.00

Table 2.18: Plotting units for pen plotter devices

Other vector plotters

Syntax
```
DEVICE LN03+
DEVICE IMAGEN
```

The LN03+ and IMAGEN are vector plotters.

Refer to Table 2.19 to see the plotting units for the LN03+ and IMAGEN vector plotter graphics hardcopy devices.

orientation	units	horizontal	vertical
LANDSCAPE	centimeters	25.40	19.05
	inches	10.00	7.50
PORTRAIT	centimeters	19.05	25.40
	inches	7.50	10.00

Table 2.19: Plotting units for LN03+ and IMAGEN devices

GKS metafiles

Syntax
```
DEVICE GKS
```

GKS refers to GKS metafiles, which are available only at sites where PHYSICA is linked with a local GKS library. In this case, there will likely be an interpreter program which allows the metafile to be replayed onto various printers and terminals.

Refer to Table 2.20 to see the plotting units for the GKS graphics metafiles.

orientation	*units*	*horizontal*	*vertical*
LANDSCAPE	centimeters	25.40	19.05
	inches	10.00	7.50
PORTRAIT	centimeters	19.05	25.40
	inches	7.50	10.00

Table 2.20: Plotting units for GKS graphics metafiles

Display files

Syntax
```
DEVICE VT640
DEVICE VT241
DEVICE CIT467
DEVICE TK4010
DEVICE TK4107
DEVICE PT100G
DEVICE GR1105
```

If a monitor type is chosen as a hardcopy device, a "display file" will be created. The graphics will be redisplayed when this file is typed, in DCL mode, on the appropriate terminal type. Any graphics that was drawn after choosing the display file will be replayed on the monitor screen, including any clearing of the graphics. The name of the file that is created will be displayed. The file is named after the plotter type.

Refer to Table 2.21 to see the plotting units for the display file type of graphics output.

orientation	*units*	*horizontal*	*vertical*
LANDSCAPE	centimeters	27.94	21.59
	inches	11.00	8.50
PORTRAIT	centimeters	21.59	27.94
	inches	8.50	11.00

Table 2.21: Plotting units for display file graphics output

DIGITIZE

Syntax `DIGITIZE { xout yout { codes }}`

The `DIGITIZE` command digitizes points off of a graph that is attached to a digitizing pad. To make use of the digitizer, simply secure the graph to the pad, enter the `DIGITIZE` command, and follow the directions.

Digitizing pad types

The only digitizing pad type that is currently supported is the Digi-Pad, Type 5A, made by GTCO Corporation, attached to a terminal.

Optional output variables

If the two optional variable names, `xout` and `yout`, are entered, then two vectors will be created. Any recorded points will be saved in these two vectors, with the horizontal axis values stored in `xout` and the vertical axis values stored in `yout`. If the optional variable name, `codes`, is entered, then a vector will be created with a code number saved in this vector for each recorded point. See the following table for the meanings of these code numbers.

	code
recorded point	1
marked point	2
connected point	3

Preparing for digitizing data

When the `DIGITIZE` command is entered, the graphics screen will be cleared and instructions will be displayed on the monitor screen. You will first be asked to give names for the four digitizer's crosshair buttons. Enter four labels, one for each of the mouse buttons, with a maximum of 15 characters per label, separated by blanks or commas. You will then be asked to type each button to enable the program to coordinate the labels with the buttons.

Now you will be asked to place the digitizer's crosshair on some point of the graph where you know the graph coordinates and to press a button. Usually, this point is the lower left corner of the graph. Then you will be asked to enter the x and y graph coordinates of this point. Repeat this process for two more points, ensuring that the third point is not collinear with the first two points. Usually, the points one enters are the lower right corner and the upper right corner of the graph. These three points define the transformation for the graph and the angle of the axes.

Digitizing data

Now you are ready to digitize data off of the graph. It is a good idea to position the crosshair on the lower left and upper right corners and check the values there.

The action that is taken at any time is determined by the mouse button or the keyboard key that is typed. See Table 2.22 for the mouse buttons and their corresponding definitions, and see Table 2.23 for the keyboard keys and their corresponding definitions.

button	*definition*
1	digitize a point and display the x and y values, do not save these values
2	digitize and display values as above; record these values in the two (optional) vectors `xout` and `yout`
3	digitize, display, and record values as above; place a small marker at the chosen point
4	digitize, display, record, and mark values as above; also connect this point to the last recorded point by drawing a line segment

Table 2.22: Mouse button definitions when digitizing data

key	*definition*
`F`	write the values that have been recorded so far to a file, enter the file name when asked
`?`	display the menu of key control codes
`/`	clear the alphanumeric monitor screen
`Q`	quit; the screen will be cleared

Table 2.23: Keyboard key definitions when digitizing data

DISABLE

Syntax `DISABLE keyword`

The `DISABLE` command allows you to disable certain features denoted by `keyword`. Use the `ENABLE` command to re-enable those features.

Graphics window borders

Syntax `DISABLE BORDER`
Default enabled

Disabling `BORDER` means that the rectangular borders that delimit the hardcopy page boundary and the window edges will not be drawn. These rectangles do not appear on any hardcopies, but may be considered to interfere with the graphics monitor display.

Broadcast messages

Syntax `DISABLE BROADCAST`
Default the terminal state when PHYSICA is invoked

The `DISABLE BROADCAST` command is only relevant for VMS. The `DISABLE BROADCAST` command is equivalent to the DCL command:

```
$ SET TERMINAL/NOBROADCAST
```

This prevents broadcast messages from being accepted by the terminal monitor. If the user is not interested in receiving broadcast messages, for example, “New mail”, then it is recommended that broadcast mode be turned off.

Confirmation requests

Syntax `DISABLE CONFIRM`
Default enabled

If `CONFIRM` is disabled, no confirmation will be requested from the `TEXT`, `FIGURE`, `LEGEND FRAME`, and `WINDOW` commands. The `CONFIRM` setting can be over-ridden on a specific command by using the `\CONFIRM` (or the `\NOCONFIRM`) qualifier.

Echoing from scripts

Syntax `DISABLE ECHO`
Default disabled

Disabling `ECHO` means that commands that are entered via a command script file will not be displayed on the monitor screen as they are executed.

Local effect

If the `ENABLE ECHO` command is encountered within a script, echoing is done only while within that script. For example, suppose you have echoing disabled at the keyboard entry level and you execute a script which has `ENABLE ECHO` within it. Subsequent lines that are read from that script will be echoed, but when that script is finished executing, echoing will be disabled again.

Saving a variable’s history

Syntax `DISABLE HISTORY`
Default enabled

Disabling HISTORY means that when a variable is altered, it's history will not be updated. A variable's history is displayed with the SHOW command. This feature was included because variables altered within large DO loops can have their histories updated so many times that virtual memory limits will be exceeded. Even if HISTORY is disabled, new variables will still have initial history lines.

Journaling input and output

Syntax	DISABLE JOURNAL DISABLE JOURNAL\MACRO
Default	enabled, journal file: PHYSICA.JOURNAL, script journaling disabled

DISABLE JOURNAL means that the journal file is to be closed. Subsequent journaling of program output and user input will be disabled. DISABLE JOURNAL\MACRO means that journaling of script commands and output is disabled, but interactively entered input and resultant output will still be enabled. Enter ENABLE JOURNAL, to reopen the last journal file that was open and append subsequent journal entries to this file. Enter the JOURNAL command to open a new journal file.

Prompting

Syntax	DISABLE PROMPTING
Default	enabled

Disabling PROMPTING means that commands will not prompt you for input when you leave something out or enter some incorrect parameter.

Replotting

Syntax	DISABLE REPLOT
Default	enabled

Disabling REPLOT means that commands that subsequent graphs and text will not be stored for replotting. See the REPLOT for more information. The REPLOT setting can be over-ridden on a specific command by using the \REPLOT (or the \NOREPLOT) qualifier.

X Window graphics replay

Syntax	DISABLE REPLAY
Default	enabled

Disabling REPLAY means that the X Window System graphics replay storage is disabled, hence subsequent graphics will not be available for replay. For example, if the graphics window size is changed, the graphics displayed in that window will not be re-displayed. The virtual memory space for storing graphics vectors is allocated dynamically. Enter DISABLE REPLAY to save virtual memory space for other uses, such as large data arrays. The REPLAY keyword only applies if an X Window type monitor is being used. See the MONITOR command for more information.

Input line recall shell

Syntax	DISABLE SHELL
Default	enabled

Disabling SHELL means that the input line recall shell is turned off. You will not be able to recall keyboard input lines when the shell has been disabled. When the SHELL is re-enabled, the buffer of input lines will be available again. It is useful to disable the shell when reading data across a network, since the terminal I/O may become corrupted if the shell is enabled. See the BUFFER command, page 11, for more information on the input line recall shell.

Stacking commands in a file

Syntax	DISABLE STACK
Default	disabled

Disabling STACK means that subsequently entered commands will not be written to the specified stack file. See the STACK command for more information.

DISPLAY

Syntax	DISPLAY 'message' DISPLAY FONT { fontname } DISPLAY SPECIAL DISPLAY HATCH DISPLAY LINES DISPLAY PCHAR DISPLAY MENU keyword

The DISPLAY command either displays a message on the monitor screen, or interprets the command parameter as a keyword and draws a corresponding table or displays a corresponding list.

Display a message

Syntax	DISPLAY 'message'

If the command parameter is not a recognized keyword, it is assumed to be a message to be displayed on the monitor screen. The message can be a literal string, enclosed in quotes, or a string variable. The message will be displayed even if `ECHO` is disabled.

Note: What is written to the journal file when the `DISPLAY string` command is encountered in a macro:

If `JOURNAL` is enabled, the `DISPLAY string` command will write the string to the journal file the same way it is written to the monitor screen.

If `JOURNAL\MACRO` is enabled, the `DISPLAY` command itself will *also* be written to the journal file.

Font table

Syntax `DISPLAY FONT { fontname }`

If the keyword `FONT` is entered, and if a `fontname` is entered, the font table for the specified `fontname` is drawn. If no `fontname` is entered, the table for the current font is drawn. To view a list of the font names, enter the `GET FONT` command. See Figure 2.5 for an example font table.

Special characters

Syntax `DISPLAY SPECIAL`

If the keyword `SPECIAL` is entered, the table of special character names that can be included as format commands in strings is drawn. This table is shown in Figure 2.6.

Hatch fill patterns

Syntax `DISPLAY HATCH`

If the keyword `HATCH` is entered, the currently defined hatch fill patterns with their corresponding numbers are drawn. For example, see Figure 2.7.

The hatch fill patterns can be used for filling text characters, histogram bars, pie charts, etc. See the `SET HATCH` command for information on redefining the hatch patterns.

Line types

Syntax `DISPLAY LINES`

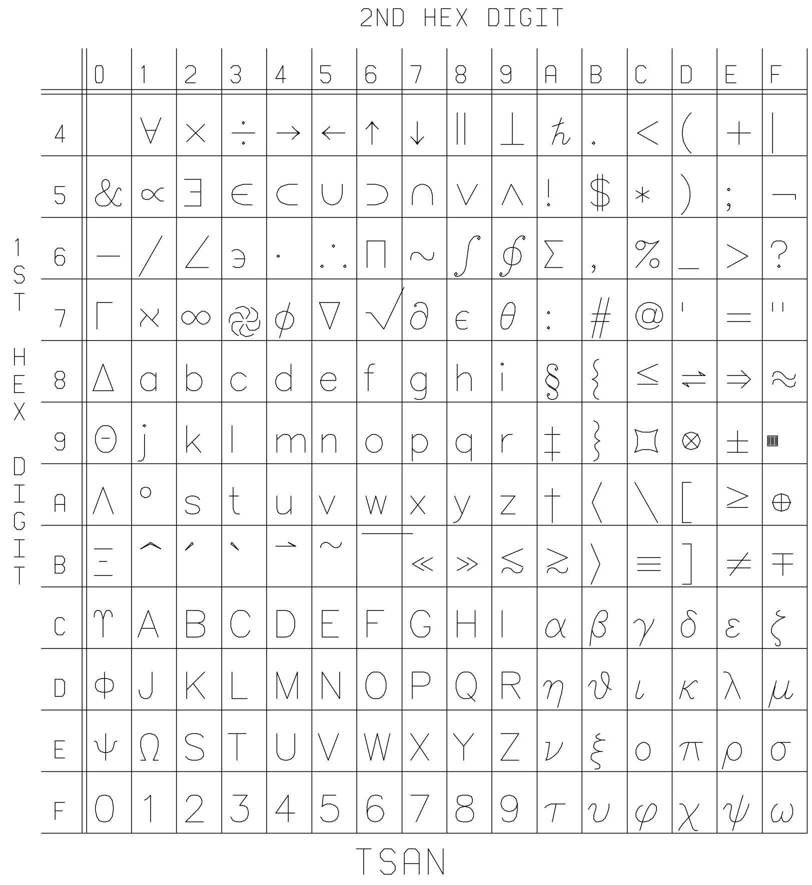


Figure 2.5: An example of a font table produced by the `DISPLAY FONT` command

Name	Uppercase	Lowercase	Name	Uppercase	Lowercase	Name	Uppercase	Lowercase
Alpha	Α	α	Overline	‾	‾	Degree	°	°
Beta	Β	β	Leftarrow	←	←	Dagger	†	†
Gamma	Γ	γ	Uparrow	↑	↑	Ddagger	‡	‡
Delta	Δ	δ	Downarrow	↓	↓	S	§	§
Epsilon	Ε	ϵ	Rightarrow	⇒	→	Langle	⟨	⟨
Zeta	Ζ	ζ	Parallel	∥	∥	Rangle	⟩	⟩
Eta	Η	η	Perp	⊥	⊥	Rlharpoons	⇌	⇌
Theta	Θ	θ	Mid	∣	∣	Vector	⇀	⇀
Iota	Ι	ι	Squarebullet	▪	▪	Neg	¬	¬
Kappa	Κ	κ	Box	□	□	Therefore	∴	∴
Lambda	Λ	λ	Sum	∑	Σ	Angle	∠	∠
Mu	Μ	μ				Vee	∨	∨
Nu	Ν	ν	Prod	∏	Π	Wedge	∧	∧
Xi	Ξ	ξ				Cdot	·	·
Omicron	Ο	ο	Int	∫	∫	Infty	∞	∞
Pi	Π	π				In	∈	∈
Rho	Ρ	ρ				Ni	∋	∋
Sigma	Σ	σ	Surd	√	√	Propto	∝	∝
Tau	Τ	τ				Exists	∃	∃
Upsilon	ϒ	υ	Oint	∮	∮	Forall	∀	∀
Phi	Φ	ϕ	Plus	+	+	Neq	≠	≠
Chi	Χ	χ	Minus	−	−	Equiv	≡	≡
Psi	Ψ	ψ	Pm	±	±	Approx	≈	≈
Omega	Ω	ω	Mp	∓	∓	Sim	∼	∼
Vartheta	ϑ	ϑ	Times	×	×	Lt	<	<
Varphi	φ	φ	Div	÷	÷	Gt	>	>
Varepsilon	ε	ε	Oplus	⊕	⊕	Ll	≪	≪
Aleph	ℵ	ℵ	Otimes	⊗	⊗	Gg	≫	≫
Tlogo			Cap	∩	∩	Lsimeq	≲	≲
Nabla	∇	∇	Subset	⊂	⊂	Gsimeq	≳	≳
Partial	∂	∂	Cup	∪	∪	Leq	≤	≤
Hbar	ℏ	ℏ	Supset	⊃	⊃	Geq	≥	≥

Figure 2.6: The table of special characters

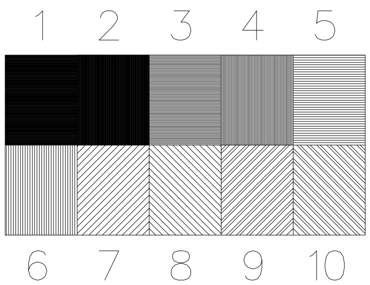


Figure 2.7: An example of the hatch fill patterns

If the keyword `LINES` is entered, the currently defined line types with their corresponding numbers are drawn. For an example of the default line types, see Figure 2.8.

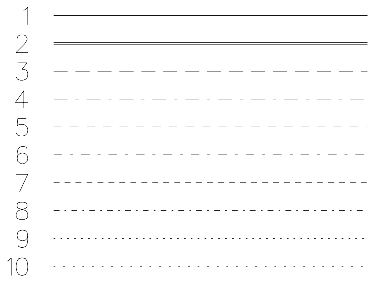


Figure 2.8: An example of the default line types

The line types can be used for drawing data curves on a graph. See the `SET LINES` command for information on redefining the line types.

Plotting symbols

Syntax `DISPLAY PCHAR`

If the keyword `PCHAR` is entered, the plotting symbols with their corresponding numbers will be drawn. These are the special plotting symbols that can be chosen with the `SET PCHAR` command. See Figure 2.9 for an example.

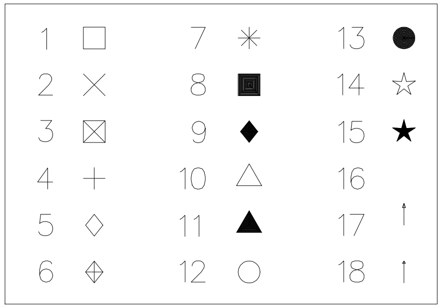


Figure 2.9: The special plotting symbols

Menus

Syntax
```
DISPLAY MENU PHYSICA
DISPLAY MENU FULL
DISPLAY MENU SHORT
DISPLAY MENU GENERAL
DISPLAY MENU XAXIS
DISPLAY MENU YAXIS
```

If the keyword `MENU` is entered, a table of values for special PHYSICA keywords or tables of values for GPLOT keywords is displayed. These menus are displayed in alphanumeric mode on the monitor screen. Refer to **Appendix A** for descriptions of how each of the GPLOT keywords can affect a graph. The plot characteristics: `MASK`, `ALIAS`, `PMODE`, `PTYPE`, and `ERRBAR` should *not* be changed in PHYSICA, as these are altered internally by various commands.

PHYSICA

The PHYSICA specific keywords are described in the `SET` command section, page 228, and the `GET` command section, page 107.

Table 2.24 shows the default values for the PHYSICA specific keywords.

```
VERSION     =  2.10
VERSIONDATE = January 16, 1998

CNTSEP   =    10.795     50.000%  |  LABSIZ  =     0.324      1.500%
LEGSIZ   =     0.345      1.600%  |  LEGFRMT = 1PE10.3
ERRFILL  =     0.000              |  ARROWID =     0.150
ARROLEN  =     0.200              |  ARROTYP =     0.000
TENSION  =     1.000              |  FILL    =         0
SPEED    =        20              |  SEED    =       12345
POSTRES  =       180              |  WIDTH   =        80
XPREV    =     0.000              |  YPREV   =     0.000
NCURVES  =         0              |  WRAP    =     0
SHOWHISTORY =      5              |  MAXHISTORY =      5
Current font = TSAN
Current plotting symbol= 0
```

Table 2.24: The PHYSICA keyword menu of default values

FULL

The full GPLOT keyword menu is displayed when the `DISPLAY MENU FULL` command is entered. This menu requires the monitor to be set to a width of 132, which is done automatically when the command is entered. If you want the monitor to be set back to a width of 80 after issuing the `MENU FULL` command, use the `SET WIDTH` command, page 228. Table 2.25 on page 64 displays the PHYSICA default values for the GPLOT keywords.

MASK	=	-4.000		BOX	=	1.000		CHARSZ	=	0.190	1.000%	CHARA	=	0.000	0.000%
PMODE	=	1.000		ALIAS	=	1.000		HISTYP	=	0.000		LINTYP	=	1.000	
PTYPE	=	0.000		COLOUR	=	1.000		CURSOR	=	1.000		ERRBAR	=	0.000	
XLWIND	=	0.000	0.000%	XUWIND	=	25.400	100.000%	XLAXIS	=	3.810	15.000%	XUAXIS	=	24.130	95.000%
NXDIG	=	5.000		NXDEC	=	-1.000		XPOW	=	0.000		XPAUTO	=	1.000	
XNUMSZ	=	0.572	3.000%	XLABSZ	=	0.572	3.000%	XTICL	=	0.381	2.000%	XTICS	=	0.190	1.000%
XTICA	=	270.000	-90.000%	XCROSS	=	0.000		XMIN	=	0.000		XMAX	=	10.000	
NLXINC	=	2.000		NSXINC	=	1.000		XAUTO	=	2.000		XITICL	=	0.572	3.000%
XITICA	=	270.000	-90.000%	XNUMA	=	0.000	0.000%	XTICTP	=	1.000		BOTTIC	=	1.000	
BOTNUM	=	0.000		TOPTIC	=	-1.000		TOPNUM	=	0.000		NXGRID	=	0.000	
XAXIS	=	1.000		XAXISA	=	0.000		XLOG	=	0.000		XZERO	=	0.000	
YLWIND	=	0.000	0.000%	YUWIND	=	19.050	100.000%	YLAXIS	=	2.857	15.000%	YUAXIS	=	17.145	90.000%
NYDIG	=	5.000		NYDEC	=	-1.000		YPOW	=	0.000		YPAUTO	=	1.000	
YNUMSZ	=	0.572	3.000%	YLABSZ	=	0.572	3.000%	YTICL	=	0.381	2.000%	YTICS	=	0.190	1.000%
YTICA	=	90.000	90.000%	YCROSS	=	0.000		YMIN	=	0.000		YMAX	=	10.000	
NLYINC	=	2.000		NSYINC	=	1.000		YAUTO	=	2.000		YITICL	=	0.572	3.000%
YITICA	=	90.000	90.000%	YNUMA	=	-90.000	-90.000%	YTICTP	=	1.000		LEFTIC	=	1.000	
LEFNUM	=	0.000		RITTIC	=	-1.000		RITNUM	=	0.000		NYGRID	=	0.000	
YAXIS	=	1.000		YAXISA	=	90.000		YLOG	=	0.000		YZERO	=	0.000	
TXTHIT	=	0.572	3.000%	TXTANG	=	0.000		XLOC	=	12.700	50.000%	YLOC	=	9.525	50.000%

Table 2.25: The full menu of GPLOT keywords, with values in centimeters

SHORT

If the `DISPLAY MENU SHORT` command is entered, a short summary table is displayed. See Table 2.26 on page 65. This does not require the terminal to be set to a width of 132.

XAXIS

If the `DISPLAY MENU XAXIS` command is entered, a table of the x-axis characteristics is displayed. See Table 2.27 on page 66. This does not require the terminal to be set to a width of 132.

YAXIS

If the `DISPLAY MENU YAXIS` command is entered, a table of the y-axis characteristics is displayed. See Table 2.28 on page 67. This does not require the terminal to be set to a width of 132.

GENERAL

If the `DISPLAY MENU GENERAL` command is entered, a table of the general GPLOT keywords is displayed. See Table 2.29 on page 68. This does not require the terminal to be set to a width of 132.

```
+---------------------------------+---------------------------------+
| MASK     =    -4.000            | CHARSZ   =     0.190     1.000% |
| PMODE    =     1.000            | HISTYP   =     0.000            |
| LINTYP   =     1.000 | LINTHK   =     1.000 | COLOUR   =     1.000 |
| XLWIND   =     0.000     0.000% | XUWIND   =    25.000   100.000% |
| XLAXIS   =     3.750    15.000% | XUAXIS   =    23.750    95.000% |
| NXDIG    =     5.000            | NXDEC    =    -1.000            |
| XPOW     =     0.000            | XPAUTO   =     1.000            |
| XMIN     =     0.000     0.000  | XMAX     =    10.000    10.000  |
| NLXINC   =     2.000            | NSXINC   =     1.000            |
| XAUTO    =     2.000            | XLOG     =     0.000            |
| YLWIND   =     0.000     0.000% | YUWIND   =    19.000   100.000% |
| YLAXIS   =     2.850    15.000% | YUAXIS   =    17.100    90.000% |
| NYDIG    =     5.000            | NYDEC    =    -1.000            |
| YPOW     =     0.000            | YPAUTO   =     1.000            |
| YMIN     =     0.000     0.000  | YMAX     =    10.000    10.000  |
| NLYINC   =     2.000            | NSYINC   =     1.000            |
| YAUTO    =     2.000            | YLOG     =     0.000            |
| XLOC     =    12.500    50.000% | YLOC     =     9.500    50.000% |
+---------------------------------+---------------------------------+
XLABEL =
YLABEL =
+---------------------------------+---------------------------------+
```

Table 2.26: The short menu of GPLOT keywords, with values in centimeters


```
XLWIND   =     0.000     0.000%  |  XUWIND   =    25.000   100.000%
XLAXIS   =     3.750    15.000%  |  XUAXIS   =    23.750    95.000%
XAXIS    =     1.000             |  XAXISA   =     0.000
---------------------------------+---------------------------------
NXDIG    =     5.000             |  NXDEC    =    -1.000
XPOW     =     0.000             |  XPAUTO   =     1.000
NLXINC   =     2.000             |  NSXINC   =     1.000
---------------------------------+---------------------------------
XNUMSZ   =     0.570     3.000%  |  XNUMA    =     0.000     0.000%
XTICL    =     0.380     2.000%  |  XTICS    =     0.190     1.000%
XTICA    =   270.000   -90.000%  |  XITICA   =   270.000   -90.000%
XITICL   =     0.570     3.000%  |  XTICTP   =     1.000
BOTTIC   =     1.000             |  BOTNUM   =     0.000
TOPTIC   =    -1.000             |  TOPNUM   =     0.000
XMIN     =     0.000             |  XMAX     =    10.000
XVMIN    =     0.000             |  XVMAX    =    10.000
---------------------------------+---------------------------------
NXGRID   =     0.000             |  XCROSS   =     0.000
XLOG     =     0.000             |  XZERO    =     0.000
XMOD     =     0.000             |  XLEADZ   =     0.000
XOFF     =     0.000             |  XLABSZ   =     0.570     3.000%
```

Table 2.27: The menu of GPLOT x-axis characteristics, with values in centimeters

YLWIND	=	0.000	0.000%	YUWIND	=	19.000	100.000%
YLAXIS	=	2.850	15.000%	YUAXIS	=	17.100	90.000%
YAXIS	=	1.000		YAXISA	=	90.000	
NYDIG	=	5.000		NYDEC	=	-1.000	
YPOW	=	0.000		YPAUTO	=	1.000	
NLYINC	=	2.000		NSYINC	=	1.000	
YNUMSZ	=	0.570	3.000%	YNUMA	=	-90.000	-90.000%
YTICL	=	0.380	2.000%	YTICS	=	0.190	1.000%
YTICA	=	90.000	90.000%	YITICA	=	90.000	90.000%
YITICL	=	0.570	3.000%	YTICTP	=	1.000	
LEFTIC	=	1.000		LEFNUM	=	0.000	
RITTIC	=	-1.000		RITNUM	=	0.000	
YMIN	=	0.000		YMAX	=	10.000	
YVMIN	=	0.000		YVMAX	=	10.000	
NYGRID	=	0.000		YCROSS	=	0.000	
YLOG	=	0.000		YZERO	=	0.000	
YMOD	=	0.000		YLEADZ	=	0.000	
YOFF	=	0.000		YLABSZ	=	0.570	3.000%

Table 2.28: The menu of GPLOT y-axis characteristics, with values in centimeters

CHARSZ	=	0.190	1.000%	CHARA	=	0.000	0.000%
BOX	=	1.000		HISTYP	=	0.000	
LINTYP	=	1.000		LINTHK	=	1.000	
COLOUR	=	1.000		CLIP	=	1.000	
NUMBLD	=	0.000					
CURSOR	=	1.000					
TXTHIT	=	0.570	3.000%	TXTANG	=	0.570	3.000%
XLOC	=	12.500	50.000%	YLOC	=	9.500	50.000%
XLABEL	=						
YLABEL	=						
XLABSZ	=	0.570	3.000%	YLABSZ	=	0.570	3.000%

Table 2.29: The menu of general GPLOT keywords, with values in centimeters

EDGR

Syntax `EDGR OPEN { filename }`
`EDGR EDIT`
`EDGR FRAME`
`EDGR CLOSE`

Examples `EDGR OPEN`
`EDGR OPEN TESTFILE`

The `EDGR` command interfaces to the graphical editor. The `keyword` determines what action is taken by EDGR.

Please refer to the EDGR USER'S GUIDE for details on using the graphical editor.

Open a drawing file

Syntax `EDGR OPEN { filename }`

If the keyword `OPEN` is entered, an EDGR drawing file will be opened. If the file name is not entered, it will be interactively requested. Do *not* give a file extension. For example, just enter `FILE`, do not enter `FILE.extension`. All graphics subsequently done will be entered into the drawing file, that is, `filename`.DWG and `filename`.DWT, until `EDGR CLOSE`, or `EDGR OPEN` again, is entered.

Commands that alter the plotting units should be entered *before* opening an EDGR file:

`ORIENTATION`, `SET UNITS`, and `DEVICE`.

Note: EDGR has its own hardcopy facility, so it is suggested that the user disable the graphics hardcopy output before opening an EDGR drawing. Use the command: `DEVICE OFF`.

Edit a drawing file

Syntax `EDGR EDIT`

If the keyword `EDIT` is entered, the graphical editor is invoked which allows you to edit your drawing.

Close a drawing file

Syntax `EDGR CLOSE`

If the keyword `CLOSE` is entered, and if a drawing file has been previously opened with

the `EDGR OPEN` command, then this command will close that file, no more graphics will be inserted into that file. If no file is currently open, then this command does nothing.

Open a new frame

Syntax `EDGR FRAME`

If the keyword `FRAME` is entered, and if a drawing file has been previously opened with the `EDGR OPEN` command, this command will open another frame within that file. If no file is currently open, then this command does nothing.

ELLIPSE

Syntax `ELLIPSE a b cx cy angle`
`ELLIPSE\FIT xin yin`

Qualifiers `\FIT, \NPTS, \XYOUT, \REPLOT`

Defaults `\-FIT, \-NPTS, \-XYOUT, \REPLOT`

The `ELLIPSE` command can uniformly populate the perimeter of an ellipse in two ways:

1. Given the major axis radius, the minor axis radius, centre coordinates and angle of the major axis
2. First fit an ellipse so a certain fraction of the data points are within the ellipse, then determine the major axis radius, the minor axis radius, centre coordinates and angle of the major axis

Output vectors

Syntax `ELLIPSE\XYOUT a b cx cy angle xout yout`
`ELLIPSE\FIT\XYOUT xin yin xout yout`

By default, the ellipse perimeter will be plotted automatically. It is assumed that a graph has been drawn already. The ellipse will be overlayed on this graph, with no plotting symbol.

If the `\XYOUT` qualifier is used, then two output vector names, `xout` and `yout`, are expected. No automatic plotting is done, and the horizontal and vertical coordinates of the ellipse perimeter will be placed in these two vectors.

Replotting

The `REPLOT` command will replot any curves that have been drawn as well as the automatically drawn ellipse, all on a common scale large enough to accommodate all curves.

If the \NOREPLOT qualifier is used, the automatically drawn ellipse will not be stored in the replot buffers, and thus will not be available for replotting.

Number of points

Syntax	`ELLIPSE\NPTS a b cx cy angle n` `ELLIPSE\FIT\NPTS xin yin n`

By default, the ellipse perimeter is populated by 260 points. If the \NPTS qualifier is used, the number of points with which to populate the perimeter is expected. This number should be divisible by four (4).

Explicitly defined

Syntax	`ELLIPSE a b cx cy angle`
Qualifiers	`\NPTS, \XYOUT, \REPLOT`
Defaults	`\-NPTS, \-XYOUT, \REPLOT`
Examples	`ELLIPSE\NOREPLOT MAJOR MINOR XCENT YCENT ANG` `ELLIPSE\XYOUT\NPTS MAJOR MINOR XC YC ANG N XOUT YOUT`

By default, the input parameters are assumed to be scalars representing the major axis radius, `a`, the minor axis radius, `b`, the coordinates of the centre, `cx` and `cy`, and the angle of the ellipse, `angle` in degrees, measured counter-clockwise from the horizontal.

Parameter order

The order in which the qualifiers appear is irrelevant. The order in which the command parameters appear is fixed: `a b xc yc angle { npts } { xout yout }`

Fit an ellipse

Syntax	`ELLIPSE\FIT xin yin` `ELLIPSE\FIT\FRACTION xin yin frac` `ELLIPSE\FIT\PARAMETERS xin yin a b cx cy angle`
Qualifiers	`\NPTS, \XYOUT, \REPLOT, \FRACTION, \PARAMETERS, \MESSAGES`
Defaults	`\-NPTS, \-XYOUT, \REPLOT, \-FRACTION, \-PARAMETERS, \MESSAGES`
Examples	`ELLIPSE\FIT\FRAC\NPTS XIN YIN FRAC N` `ELLIPSE\FIT\XYOUT\NPTS XIN YIN N XO YO` `ELLIPSE\FIT\XYOUT\FRAC XIN YIN FRAC XO YO` `ELLIPSE\FIT\XYOUT\FRAC\NPTS XIN YIN FRAC N XO YO` `ELLIPSE\FIT\PARAM\FRAC\NPTS XIN YIN FRAC A B CX CY ANG N` `ELLIPSE\FIT\PARAM\XYOUT\FRAC\NPTS XIN YIN F A B CX CY ANG N XO YO`

If the \FIT qualifier is used, then the first two parameters, `xin` and `yin`, are assumed to be vectors which contain data points to which an ellipse is to be fitted.

Parameter order

The order that the qualifiers appear is irrelevant. The order that the command parameters appear is fixed: `xin yin { fraction } { a b cx cy angle } { npts } { xout yout }`

Method

The major axis radius and the centre are found by least squares fitting a line through the data points. The ratio of the major axis radius to the minor axis radius is found by computing the standard deviations about the major and minor axes. The minimum value of the major axis is found for each point so that the point will be inside the ellipse, then a value for the major axis is picked so as to be greater than or equal to the specified fraction of the values.

Fraction of points within ellipse

Syntax	`ELLIPSE\FIT\FRACTION xin yin frac`
Defaults	$\texttt{frac} = 0.9$

By default, the ellipse is fit to the data points so that it encloses 90% of the data points.

If the \FRACTION qualifier is used, then a scalar, `frac`, representing the fraction of data points to be within the ellipse will be expected, $0 < \texttt{frac} < 1$.

Messages

By default, the major axis radius, the minor axis radius, the coordinates of the centre, and the angle of the fitted ellipse will be displayed on the monitor screen. To suppress this, use the `\-MESSAGES` qualifier.

Output parameters

Syntax `ELLIPSE\FIT\PARAMETERS xin yin a b cx cy angle`

If the `\PARAMETERS` qualifier is used, output scalar names will be expected to receive the resulting ellipse parameters:

`a`	major axis radius
`b`	minor axis radius
`cx` and `cy`	x and y coordinates of the centre
`angle`	angle of the major axis, in degrees, measured counter-clockwise from the horizontal

ENABLE

Syntax `ENABLE keyword`

The `ENABLE` command allows you to re-enable features that have been disabled with the `DISABLE` command.

Graphics window borders

Syntax `ENABLE BORDER`
Default enabled

Enabling `BORDER` means that the rectangular borders that delimit the hardcopy page boundary and the window edges will again be drawn. These rectangles do not appear on any hardcopies, but may be considered to interfere with the graphics monitor display.

Broadcast messages

Syntax `ENABLE BROADCAST`
Default the terminal state when PHYSICA is invoked

The `ENABLE BROADCAST` command is only relevant for VMS. The `ENABLE BROADCAST` command is equivalent to the DCL command:

```
$ SET TERMINAL/BROADCAST
```


This allows a broadcast message to be accepted by the terminal. If the user is not interested in receiving broadcast messages, for example, “New mail”, then it is recommended that broadcast mode be turned off.

Confirmation requests

Syntax `ENABLE CONFIRM`
Default enabled

If `CONFIRM` is enabled, confirmation will be requested from the `TEXT`, `FIGURE`, `LEGEND FRAME`, and `WINDOW` commands. The `CONFIRM` setting can be over-ridden on a specific command by using the `\NOCONFIRM` (or the `\CONFIRM`) qualifier.

Echoing from scripts

Syntax `ENABLE ECHO`
Default disabled

Enabling `ECHO` means that commands that are entered via a command script file will be displayed on the monitor screen as they are executed.

Local effect

If the `ENABLE ECHO` command is encountered within a script, echoing is done only while within that script. For example, suppose you have echoing disabled at the keyboard entry level and you execute a script which has `ENABLE ECHO` within it. Subsequent lines that are read from that script will be echoed, but when that script is finished executing, echoing will be disabled again.

Saving a variable’s history

Syntax `ENABLE HISTORY`
Default enabled

When `HISTORY` is disabled, if a variable is altered, it’s history will not be updated. A variable’s history is displayed with the `SHOW` command. This feature was included because variables altered within large `DO` loops can have their histories updated so many times that virtual memory limits will be exceeded. Even if `HISTORY` is disabled, new variables will still have initial history lines.

Journaling input and output

Syntax `ENABLE JOURNAL`
`ENABLE JOURNAL\MACRO`
Default enabled, journal file: `PHYSICA.JOURNAL`, script journaling disabled

`ENABLE JOURNAL` means that subsequent journal entries, that is, program output and user input, will be appended to the last journal file that was opened. By default, program input from script files and resultant output will not be journaled. Use the `ENABLE JOURNAL\MACRO` to also journal script input and output. See the `JOURNAL` command and the `DISABLE JOURNAL` command for more information.

Prompting

Syntax `ENABLE PROMPTING`
Default enabled

Enabling `PROMPTING` means that commands will prompt you for input when you leave something out or enter some incorrect parameter.

Replotting

Syntax `ENABLE REPLOT`
Default enabled

Enabling `REPLOT` means that subsequent graphs and text will be stored for replotting. See the `REPLOT` for more information. The `REPLOT` setting can be over-ridden on a specific command by using the `\NOREPLOT` (or the `\REPLOT`) qualifier.

X Window graphics replay

Syntax `ENABLE REPLAY`
Default enabled

Enabling `REPLAY` means that the X Window System graphics replay storage is enabled, hence subsequent graphics will be available for replay. For example, if the graphics window size is changed, the graphics displayed in that window will be re-displayed. The virtual memory space for storing graphics vectors is allocated dynamically. Enter `DISABLE REPLAY` to save virtual memory space for other uses, such as large data arrays. The `REPLAY` keyword only applies if an X Window type monitor is being used. See the `MONITOR` command for more information.

Input line recall shell

Commands

Syntax `ENABLE SHELL`
Default enabled

Disabling `SHELL` means that the input line recall shell is turned off. You will not be able to recall keyboard input lines when the shell has been disabled. When the `SHELL` is re-enabled, the buffer of input lines will be available again. It is useful to disable the shell when reading data across a network, since the terminal I/O may become corrupted if the shell is enabled. See the `BUFFER` command, page 11, for more information on the input line recall shell.

Stacking commands in a file

Syntax `ENABLE STACK`
Default disabled

Enabling `STACK` means that subsequently entered commands will again be written to the specified stack file. See the `STACK` command for more information.

ERASEWINDOW

Syntax `ERASEWINDOW { n }`
Defaults `n` = current window number

The `ERASEWINDOW` command erases the graphics within a pre-defined window. It will erase the graphics within a window on the monitor screen, and on graphics hardcopy PostScript output, and on graphics hardcopy bitmap output.

This *does not apply* to any other graphics hardcopy output or to EDGR drawings, that is, any erased graphics will still appear on that hardcopy and will still be in an EDGR drawing.

The parameter `n` refers to a pre-defined window number. If `n` is not entered, it defaults to the current window number.

EXECUTE

Syntax `EXECUTE filename { p1 p2 ... }`
`@filename { p1 p2 ... }`
Defaults the default filename extension = `.PCM`
Examples `EXECUTE FILE.PCOM`
`@FILE 1.2 ‘string’ X Y`

The `EXECUTE` command reads program input from a file. When the end of file is reached, input will again be expected to be entered from the keyboard, or from the a calling executable file. You may have up to twenty (20) nested executable files. The ‘at’ sign, `@`, is equivalent to

'EXECUTE '.

Within script files, it is possible to have labels, GOTO statements, IF blocks, and DO loops.

Environment variables in file names

For UNIX users, it is now possible to use an environment variable in a file name, if the environment variable is preceeded by a $. For example,

```
setenv FILE dum.pcm
physica
@$FILE
```

The environment variable can be just the first part of the filename, for example,

```
setenv FILE dum
physica
@$FILE.pcm
```

Filename extensions

If the file name is entered without a filename extension, the default extension, .PCM, is automatically appended to the filename. The EXTENSION command, page 81, is used to redefine the default file extension. The default file extension applies to the EXECUTE command *only*.

script library

VMS: The logical name PHYSICA$LIB can be used to point to the disk and directory for script files. For example, suppose you define:

```
$DEFINE PHYSICA$LIB dsk1:[dir1],dsk2:[dir2]
```

before running the PHYSICA program.
When you execute a script, the current disk and directory is searched for the file. If it cannot be found there, the logical search list is used to find the file. If the logical name is not defined, only the current disk and directory will be searched.

UNIX: The environment variable PHYSICA_LIB can be used to point to the disk and directory for script files. For example, suppose you define:

```
%setenv PHYSICA_LIB disk/directory
```

before running the PHYSICA program.
When you execute a script, the current disk and directory is searched for the file. If it cannot be found there, the environment variable is used to find the file. If the environment variable is not defined, only the current disk and directory will be searched.

Comments

A comment line is any line that begins with an exclamation mark, !. These lines are simply ignored, but can be useful for documentation of files. Comments can also be appended to the end of any line. Just start the comment with an exclamation mark. For example,

```
READ FILE.DAT X Y Z  ! This is a comment
```

Echoing

If the ENABLE ECHO command, page 73, is entered, the commands that are read from the file will be displayed on the monitor screen. This is useful for following the progress of a command file. If ECHO is disabled globally, but is enabled within a script file, it will be enabled only while that file is executing.

Temporarily passing control to the keyboard

If the TERMINAL command, page 256, is encountered in a command file, control passes back to the terminal keyboard. The user interactively enters commands at this point, until a null line is entered. The command file then recommences execution with the command immediately after the TERMINAL command.

By default, the message 'type ¡RETURN¿ to continue' will be displayed when the TERMINAL command is encountered. You can specify the message by entering a string with the TERMINAL command.

Returning from a script

If the RETURN command, page 224, is encountered in a command file, control passes back to the calling script, if there is one, or to the keyboard, if that script was the top level script.

Aborting a script

If control-c is typed while a script is executing, the entire script stack will be aborted. That is, no matter how deeply the scripts are nested, program flow control is passed back to the keyboard.

If you type the RETURN command from the keyboard after a TERMINAL command has been encountered in a command file, execution of that command file is aborted.

Passing parameters to a script

Parameters that are entered after the file name are used in two ways.

sequential parameters	the n^{th} parameter will replace the n^{th} ? that is found in the file
numbered parameters	the n^{th} parameter in the list will replace all ?n's found in the file

Sequential parameters must be in a one-to-one correspondence with the ?'s, and in the correct order. It is possible to mix sequential and numbered parameters in the same file, but it is not recommended as this can be very confusing.

Prompting

By default, if an incorrect parameter of a valid command is read from the file or is substituted from the parameter list, the user will be prompted to enter the correct information from the terminal keyboard, and the command will then be executed. Use the `DISABLE PROMPTING` command if a script is to abort when invalid command parameters are encountered.

Labels and GOTOs

A label is a string terminated with a colon, `:`, and with no embedded blanks. A label must be on a line by itself.

Use a `GOTO` to branch to a label. Do *not* include the colon with the label after a `GOTO`.

Example 1

```
 ...
GOTO A_LABEL    ! branch to the label (note there is no :)
 ...
A_LABEL:        ! this is a label (note the :)
 ...
```

Example 2

```
 ...
START:
IF (J>8) THEN GOTO END
 ...
J=J+1
GOTO START
END:
 ...
```


DO loops

`DO` loops in PHYSICA are similar to FORTRAN do loops, but *must* be closed off with an `ENDDO` statement. The basic form of the `DO` statement is:

```
DO j = x
```

where the looping variable, `j`, will be made into a scalar variable, and the range of the loop, `x`, can be any expression resulting in a vector.

Nested loops are allowed. The maximum number of `DO` loops in a file is fifty (50).

Example 1

```
 ...
DO J = x        ! x must be a vector, to loop will execute len(x) times
 ...            ! with J taking on the value of each element of x
ENDDO           ! end of loop
 ...
```

Example 2

```
 ...
DO I = [2:20:4] ! the loop will execute 5 times, with I taking on the
 ...            ! values [2;6;10;14;18]
ENDDO           ! end of loop
 ...
```

Conditional statements

The general form of a conditional statement is:

```
IF (boolean) THEN
```

The `boolean` can take any form, but must be either a simple function or it must be enclosed in parentheses, and it must have a scalar result. A result of 1 is true, while anything else is false.

An `IF` statement can precede a single command or it can precede a block of commands. If an `IF` statement precedes a block of commands, it *must* be closed off with an `ENDIF` statement. Nested `IF` blocks are allowed. The maximum number of `IF` blocks in a file is fifty (50).

An IF statement can also precede a single command, in which case do not use the ENDIF.

Example 1

```
 ...
IF (A>B) THEN DISPLAY 'A > B'
IF (A=B) THEN DISPLAY 'A = B'
IF (A<B) THEN DISPLAY 'A < B'
 ...
```

Example 2

```
 ...
IF (A>B) THEN
 ...
ENDIF
 ...
```

Example 3

```
 ...
START2:
IF (J<=8) THEN
 ...
 J=J+1
 GOTO START2
ENDIF
 ...
```

EXTENSION

Syntax	EXTENSION { 'ext' }
Examples	EXTENSION EXTENSION 'PHYSICA'

The EXTENSION command is used to redefine the default file extension for executable script files. The original PHYSICA default file extension is PCM. The default file extension applies to the EXECUTE command only. If you give a file name without a file extension, the default extension is automatically appended to the file name.

If the EXTENSION command is entered without a parameter, the current default extension is displayed.

Example

If you have a script file named MACRO_FILE.PCM you can execute this file with the command: @MACRO_FILE

If you have a script file named MACRO_FILE.PHYSICA you can execute this file with the command: @MACRO_FILE.PHYSICA, or with

```
EXTENSION 'PHYSICA'
@MACRO_FILE
```

FIGURE

Syntax	FIGURE BOX { lowx lowy hix hiy } FIGURE POLYGON nvert { cx cy sx sy } FIGURE CIRCLE radius { cx cy } FIGURE ARC { cx cy sx sy ex ey } FIGURE WEDGE { cx cy sx sy ex ey } FIGURE ELLIPSE a b { cx cy } angle FIGURE ARROW { sx sy ex ey }
Qualifiers	\CONFIRM, \GRAPH, \PERCENT, \WORLD
Defaults	\NOCONFIRM, \PERCENT
Examples	FIGURE BOX 10 10 90 90 FIGURE\NOCONFIRM\GRAPH POLY 6 -1 2 .01 .03 FIGURE\WORLD CIRC 2 FIGURE\WORLD WEDGE FIGURE\NOCONFIRM\GRAPH ARC -10.2 4.7 FIGURE\GRAPH ELLIPSE 5.3 1.4 -2.3 3.5 45

The FIGURE command is used to draw geometric figures. The figure type is chosen with a keyword. See Table 2.30.

Line types

The line type used for drawing figures will be the current value of LINTYP, which can be changed with the SET LINTYP command The default value for LINTYP is 1. The line type definitions can be changed with the SET LINES command. Line type 1 defaults to a normal line. See the DISPLAY LINES command for information on viewing the line types.

Fillable figures

Most of the figures are fillable: BOX, POLYGON, WEDGE, CIRCLE, ELLIPSE, and ARROWs with

keyword	*figure type*
`ARC`	an arc of a circle
`WEDGE`	a sector of a circle (fillable)
`CIRCLE`	a circle (fillable)
`BOX`	a rectangle (fillable)
`POLYGON`	a regular polygon (fillable)
`ELLIPSE`	an ellipse (fillable)
`ARROW`	an arrow (closed heads are fillable)

Table 2.30: Geometric figures that can be drawn with the `FIGURE` command

closed heads. Use the `SET FILL` command to change the fill type and pattern. See Table 2.55 on page 233 for a description of the interpretations of the `FILL` keyword.

X Windows

When running under X Windows, mouse button two toggles the continuous display of the graphics cursor location. The PHYSICA keyword `CUNITS` is the units type for these numbers. If `CUNITS` = `WORLD`, the numbers depend on the current units type, either `CM` or `IN`, as chosen with `SET UNITS`. If `CUNITS` = `GRAPH`, the numbers displayed depend on the current graph axis scales. If `CUNITS` = `PERCENT`, the numbers depend on the current window.

Units

The numeric parameters may be expressed in three types of units, which are chosen by command qualifier. The default is `\PERCENT`. See Table 2.31 for a listing of the qualifiers and their interpretations.

qualifier	*interpretation of the coordinates*
`\PERCENT`	percentages of the current window, as chosen with the `WINDOW` command. Lengths are in terms of the horizontal dimension.
`\GRAPH`	graph units, that is, the units defined by the minimum and maximum values for the last graph drawn. If no graph has been drawn yet, the defaults are $-1 \leq x \leq 1$ and $-1 \leq y \leq 1$
`\WORLD`	centimeters or inches, as chosen with the `SET UNITS` command

Table 2.31: Types of units recognized by the `FIGURE` command

For example, if the `\PERCENT` qualifier is used, then a location of $(50, 50)$ represents the centre of the current window. If the `\WORLD` qualifier is used, the coordinates are in units of the

world coordinate system, the plotting units. The default world coordinate system units are centimeters. See the `DEVICE` command for tables showing the dependence of plotting units on the graphics hardcopy output device.

Confirmation

If the `FIGURE` command is entered interactively, the default is that confirmation that the figure is acceptable as drawn will be requested. The figure will not be be entered into a hardcopy plot file, or into an open EDGR file, unless it is accepted. However, if `CONFIRM` has been disabled, with the `DISABLE CONFIRM` command, the default will be that no such confirmation will be requested. If the `\CONFIRM` qualifier is used, confirmation will be forced. If the `\NOCONFIRM` qualifier is used, confirmation will be suppressed.

Stack file

If a stack file is open, via the `STACK` command, then the (x, y) coordinates will be written to the stack file, even if they are chosen by the graphics cursor. Thus, when this stack file is executed, using the `EXECUTE` command, the graphics cursor will not be used. If confirmation is requested and the figure is not `OK`, then the command is *not* written to the stack file.

Circle

Syntax `FIGURE CIRCLE r { cx cy }`

The `FIGURE CIRCLE` command draws a circle, centred at the point `(cx,cy)`, with radius `r`. The parameter `r` is not optional, and is in terms of the horizontal dimension. The graphics cursor is used if either `cx`, or `cy` is not entered.

Arc

Syntax `FIGURE ARC { cx cy sx sy ex ey }`

The `FIGURE ARC` command draws an arc of a circle, centred at the point `(cx,cy)`, starting at the point `(sx,sy)` and finishing on the line through the points `(ex,ey)` and `(cx,cy)`. The graphics cursor is used if any of `cx`, `cy`, `sx`, `sy`, `ex`, or `ey` are not entered.

The first two points determine the radius and the starting azimuth, while the final point determines the final azimuth only. If the final point is the same as the first point, the centre, then a complete circle will be drawn.

Wedge

Syntax `FIGURE WEDGE { cx cy sx sy ex ey }`

The `FIGURE WEDGE` command draws a sector of a circle, centred at the point `(cx,cy)`, starting at the point `(sx,sy)` and finishing on the line through the points `(ex,ey)` and `(cx,cy)`. The endpoints of the arc are joined to the centre of arc. The graphics cursor is used if any of `cx`, `cy`, `sx`, `sy`, `ex`, or `ey` are not entered.

The first two points determine the radius and the starting azimuth, while the final point determines the final azimuth only. If the final point is the same as the first point, the centre, then a complete circle will be drawn.

Box

Syntax `FIGURE BOX { lowx lowy hix hiy }`

The `FIGURE WEDGE` command draws a box, or rectangle, with lower left hand corner at `(lowx,lowy)`, and upper right hand corner at `(hix,hiy)`. The graphics cursor is used if any of `lowx`, `lowy`, `hix`, or `hiy` are not entered.

Polygon

Syntax `FIGURE POLYGON n { cx cy sx xy }`

The `FIGURE POLYGON` command draws a regular polygon, with `nvert` vertices, centred at `(cx,cy)`, and with the first vertex at `(sx,sy)`. The graphics cursor is used if any of `cx`, `cy`, `sx`, or `sy` are not entered.

Ellipse

Syntax `FIGURE ELLIPSE a b { cx cy } ang`

The `FIGURE ELLIPSE` command draws an ellipse, with `a` being the major axis radius, `b` the minor axis radius, centred at point `(cx,cy)`, and with the major axis at an angle of `ang` degrees, measured counter-clockwise from the horizontal. The graphics cursor is used if either `cx` or `cy` are not entered.

Arrow

Syntax `FIGURE ARROW { sx sy ex ey }`

The `FIGURE ARROW` command draws an arrow with base at `(sx,sy)` and end point at `(ex,ey)`. The graphics cursor is used if any of `sx`, `sy`, `ex`, or `ey` are not entered.

Styles

The arrow style is chosen with the `SET ARROTYP` command. The default is value of `ARROTYP` is `0`. See Table 2.54 on page 231 and Figure 2.24 on page 230.

Head width

The width of the arrowhead is chosen with the `SET ARROWID` command. `ARROWID` is the arrow head width as a fraction of the total arrow length. The default value of `ARROWID` is `0.15`.

Length

The length of the arrowhead is chosen with the `SET ARROLEN` command. `ARROLEN` is the arrow head length as a fraction of the total arrow length. The default value of `ARROLEN` is `0.20`.

FILTER

Syntax	`FILTER\MEDIAN x f npt` `FILTER\MEAN x f npt` `FILTER\NONRECURSIVE x f c` `FILTER\RECURSIVE x f c d`
Qualifiers	`\MEDIAN, \MEAN, \NONRECURSIVE, \RECURSIVE`
Default	`\MEDIAN`
Examples	`FILTER\MEDIAN X XF 5` `FILTER\NONRECURSIVE X XF [1;-2;1]` `FILTER\MEAN X XF -5` `FILTER\RECURSIVE X XF [.3584;1.2832;.3584;0;0] [0;1]`

A digital filter is a linear combination of the input data, x, and possibly the output data, f. The input data is assumed to be *equally spaced* samples of some continuously varying quantity; and any error or noise is in the measurements. In the PHYSICA implementation of filters, the input data is assumed to have unit spacing, so a scale factor may have to be applied to produce the correctly scaled output data.

The simplest kinds of filters are the nonrecursive filters defined by the convolution formula:

$$f_n = \sum_{k=-N}^{N} c_k x_{n-k}$$

The coefficients c_k are the constants of the filter, the x_{n-k} are the input data, and the f_n are the outputs. When values of the output as well as the data values are used to compute the output values, the filter is called a recursive filter. It is usual to limit the range of nonzero coefficients to current and past values of the data x_n and to only past values of the output f_n. This type of filter is called causal recursive and can be defined by the convolution formula:

$$f_n = \sum_{k=0}^{N} c_k x_{n-k} + \sum_{k=1}^{M} d_k f_{n-k}$$

Nonrecursive or recursive filters using constant coefficients c_k and d_k are called time-invariant filters. Users are urged to refer to textbooks dealing with digital filters, such as Digital Filters by R.W. Hamming, Prentice-Hall 1977, or Digital Signal Analysis by Samual D. Stearns, Hayden Book Co. Inc.

Noise amplification caused by filtering

It can be shown (Hamming, p.17) that the sum of the squares of the filter coefficients measures the noise amplification of the filtering process. Thus, the variance, σ^2, will be amplified by $\sum \sigma c_i^2$.

Median filter

Syntax `FILTER x f npt`
`FILTER\MEDIAN x f npt`

The default is `\MEDIAN`, that is, to use a running median filter. The data array, `x`, is filtered through a window `npt` points in width. `npt` must be ≥ 2.

The median filter is particularly good at removing 'spikes' from data. The median filter moves a window over the data and outputs the median value of the data points within each window placement. The window butts up against the ends. When `npt` is even, the filter is applied twice, first skewed left then skewed right, and the results are averaged.

Mean filter

Syntax `FILTER\MEAN xin xout npt`

If the `\MEAN` qualifier is used, the filter will be the running mean, or average, filter. This filter method is sensitive to large spikes in the data. Any large spikes, for example, $> \sim 1000$ times normal value, should first be removed, by, for example, the median filter.

There are two versions of the running mean filter, which are chosen by whether `npt` is positive or negative. The window width is always `|npt|`.

If `npt` > 0, the average value of each window placement is calculated by summing the relevant points and dividing by `npt`. The averaging window butts up against the end.

If `npt` < 0, a much faster method is used which adds a new point to the right and drops an old one from the left. The window runs off half way from each end, but pseudo points outside the range are set to the appropriate end point values.

Nonrecursive filters

Syntax `FILTER\NONRECURSIVE x f c`

If the `\NONRECURSIVE` qualifier is used, the third parameter, `c`, must be a vector. The data array, `x`, is processed through a nonrecursive filter using the values of `c` as the data coefficients:

`f[n]` $= \sum_{k=1}^{N}$ `c`$[k]$ `x`$[n+k-(\frac{N}{2}+1)]$

where N is the length of vector `c`. Note that when `c` has an even number of elements, the filter will be applied to the n^{th} point by application to points from $n-N/2$ to $n-1+N/2$. For example, when N is two, the weightings will be applied to the previous point and to the current point.

Differentiating nonrecursive filters

Remember, that the `x`'s must be equally spaced, and are actually assumed by the `FILTER` command to have unit spacing. Thus, to obtain the correct output scaling, multiply `f` by $k!/(N-1)!h^k$, where k is the order of the derivative, N is the length of vector `c`, and h is the spacing of `x`, that is, $h =$ `x[i+1]-x[i]`. For example:

```
FILTER\NONRECURSIVE X XOUT [2;-16;0;16;-2]  ! 1st deriv. nonrecursive filter
XOUT=XOUT/(24*(X[2]-X[1]))     ! use scale factor 1/(h*4!)
```

See Table 2.32 for various first derivative nonrecursive filter data coefficients. See Table 2.33 for various second derivative nonrecursive filter data coefficients. See Table 2.34 for various third derivative nonrecursive filter data coefficients.

type	*data coefficients*	*scale factor*
3 point	[1; 0; -1]	$1/2h$
4 point	[1; -6; 3; 2]	$1/6h$
5 point	[2; -16; 0; 16; -2]	$1/24h$
6 point	[-4; 30; -120; 40; 60; -6]	$1/120h$

Table 2.32: Various 1^{st} derivative nonrecursive filters

type	data coefficients	scale factor
3 point	[1; -2; 1]	$1/h^2$
4 point	[0; 3; -6; 3]	$1/3h^2$
5 point	[-1; 16; -30; 16; -1]	$1/12h^2$
6 point	[0; 5; 80; -150; 80; -5]	$1/60h^2$

Table 2.33: Various 2^{nd} derivative nonrecursive filters

type	data coefficients	scale factor
4 point	[-1; 3; -3; 1]	$1/h^3$
5 point	[-2; 4; 0; -4; 2]	$1/4h^3$
6 point	[5; -35; 70; -50; 5; 5]	$1/20h^3$

Table 2.34: Various 3^{rd} derivative nonrecursive filters

Smoothing nonrecursive filters

See Table 2.35 for various quadratic smoothing nonrecursive filter data coefficients. See Table 2.36 for various quartic smoothing nonrecursive filter data coefficients. See Table 2.37 for Spencer's formulae smoothing nonrecursive filter data coefficients.

type	data coefficients	scale factor
5 point	[-3; 12; 17; 12; -3]	$1/35$
7 point	[-2; 3; 6; 7; 6; 3; -2]	$1/21$
9 point	[-21; 14; 39; 54; 59; 54; 39; 14; -21]	$1/231$
11 point	[-36; 9; 44; 69; 84; 89; 84; 69; 44; 9; -36]	$1/429$

Table 2.35: Smoothing nonrecursive filters (quadratic)

Interpolating nonrecursive filters

Suppose we have points in a vector which are "bad" and need to be replaced. Assuming one can fit the data with an odd degree polynomial. The next higher order difference equation, when set to zero, can be used to give the desired filter coefficients. For example, if the data can be fit with a 5^{th} order polynomial, the fourth difference set to zero gives:

$$y_{n-2} - 4y_{n-1} + 6y_n - 4y_{n+1} + y_{n+2} = 0$$

and solving for y_n gives:

$$y_n = \frac{1}{6}(-y_{n-2} + 4y_{n-1} + 4y_{n+1} - y_{n+1})$$

so the data filter coefficients are [$-\frac{1}{6}$; $\frac{2}{3}$; 0; $\frac{2}{3}$; $-\frac{1}{6}$].

type	data coefficients	scale factor
7 point	[5; -30; 75; 131; 75; -30; 5]	$1/231$
9 point	[15; -55; 30; 135; 179; 135; 30; -55; 15]	$1/429$
11 point	[18; -45; -10; 60; 120; 143; 120; 60; -10; -45; 18]	$1/429$
13 point	[110; -198; -135; 110; 390; 600; 677; 600; 390; 110; -135; -198; 110]	$1/2431$

Table 2.36: Smoothing nonrecursive filters (quartic)

type	data coefficients	scale factor
15 point	[-3; -6; -5; 3; 21; 46; 67; 74; 67; 46; 21; 3; -5; -6; -3]	$1/320$
21 point	[-1; -3; -5; -5; -2; 6; 18; 33; 47; 57; 60; 57; 47; 33; 18; 6; -2; -5; -5; -3; -1]	$1/350$

Table 2.37: Smoothing nonrecursive filters (Spencer's formulae)

Recursive filters

Syntax `FILTER\RECURSIVE x f c d`

If the `\RECURSIVE` qualifier is used, the third parameter, `c`, must be a vector, and the fourth parameter, `d`, must also be a vector. The data array, `x`, is processed through a recursive filter. This allows for the specification of a completely general recursive filter of arbitrary length. The values of `c` are the filter coefficients which operate on the data. The values of `d` are the filter coefficients which operate on the previously made output.

$$\mathtt{f}[n] = \sum_{k=1}^{N} \mathtt{c}[k]\ \mathtt{x}[n + k - (\frac{N}{2} + 1)] + \sum_{k=1}^{M} \mathtt{d}[k]\ \mathtt{f}[n - k]$$

where N is the length of vector `c` and M is the length of vector `d`.

Integrating recursive filters

The trapezoidal rule integration filter:

$$G_{n+1} = 0.5(x_{n+1} + x_n) + G_n$$

The Leo Tick formula for integration:

$$G_{n+1} = h(0.3584x_{n+1} + 1.2832x_n + 0.3584x_{n-1}) + G_{n-1}$$

See Table 2.38 for the trapezoidal rule and the Leo Tick formula integrating recursive filter coefficients.

	data coefficients	*output coefficients*
Trapezoidal rule	[0.5; 0.5]	[1]
Leo Tick formula	[0.3584; 1.2832; 0.3584]	[0; 1]

Table 2.38: Integrating recursive filters

Examples

The following script demonstrates how you can use the FILTER command to smooth data. See Figure 2.10.

```
X=[0:4:.05]
Y=X^2-3*X+3+SIN(X*3)*RAN(X)
WINDOW 15
SET PCHAR -1
LABEL\XAXIS ‘original data’
GRAPH X Y
WINDOW 16
LABEL\XAXIS ‘11 point quadratic formula’
FILTER\NONRECURSIVE Y YF [-36;9;44;69;84;89;84;69;44;9;-36]
SET PCHAR 0
GRAPH X YF/429
WINDOW 17
LABEL\XAXIS ‘11 point quartic formula’
FILTER\NONRECURSIVE Y YF [18;-45;-10;60;120;143;120;60;-10;-45;18]
GRAPH X YF/429
WINDOW 18
LABEL\XAXIS ‘15 point Spencer’//CHAR(39)//‘s formula’
FILTER\NONRECURSIVE Y YF [-3;-6;-5;3;21;46;67;74;67;46;21;3;-5;-6;-3]
GRAPH X YF/320
```

The following script demonstrates how you can use the FILTER command to differentiate data. See Figure 2.11.

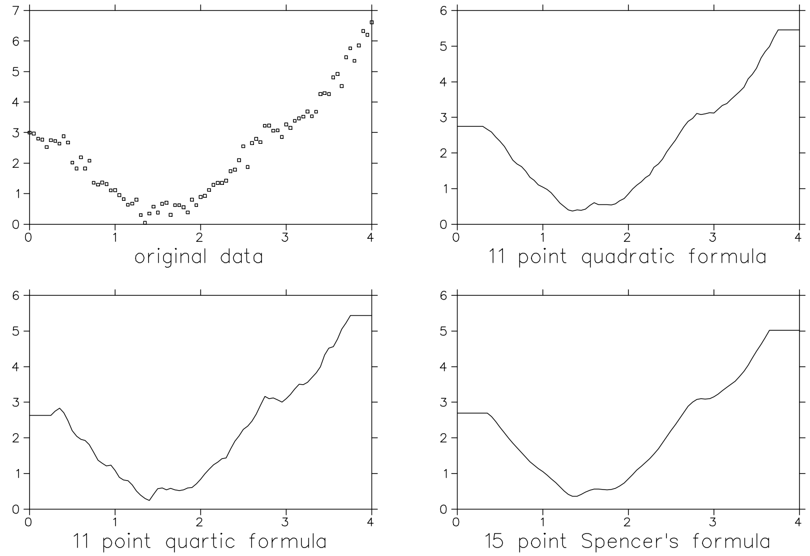


Figure 2.10: A FILTER example showing data smoothing

```
X=[0:4:.2]
H=X[2]-X[1]
Y=X^2-3*X+3
WINDOW 5
SET PCHAR -1
LABEL\XAXIS ‘original data’
SET %XLABSZ 5
GRAPH X Y
WINDOW 6
SET PCHAR 0
LABEL\XAXIS ‘1<^>st<_> derivatives’
FILTER\NONRECURSIVE Y YF [-4;30;-120;40;60;-6]
SCALE 0 4 0 -3 5 0
SET PCHAR -2
GRAPH X YF/(120*H)
SET PCHAR 0
GRAPH\NOAXES X 2*X-3
```

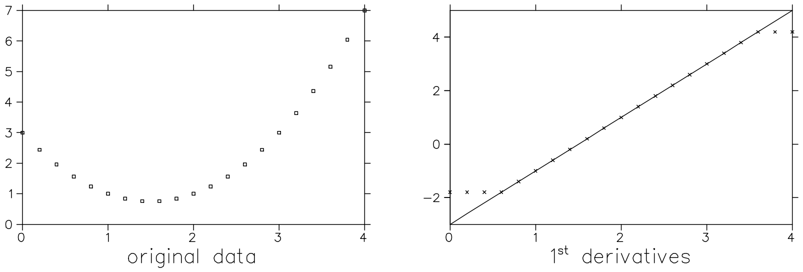


Figure 2.11: A `FILTER` example showing 1^{st} derivative

FIT

Syntax	`FIT y = expression` `FIT\UPDATE yout`
Qualifiers	`\NORMAL, \POISSON, \UPDATE, \ITMAX, \WEIGHTS, \ZEROS, \TOLERANCE, \CHISQ, \CL, \CORRMAT, \COVMAT, \E1, \E2, \VARNAMES, \FREE, \RESET, \MESSAGES`
Defaults	`\NORMAL, \-UPDATE, \-ITMAX, \-WEIGHTS, \ZEROS, \-TOLERANCE, \-CHISQ, \-CL, \-CORRMAT, \-COVMAT, \-E1, \-E2, \-VARNAMES, \-FREE, \-RESET, \MESSAGES`
Examples	`FIT Y=A*X+B` `FIT\WEIGHTS\CHISQ\CL\ITMAX W 3 Y=A*EXP(-B*X)+C` `FIT\CORR\NOMESS Y=A*X+B` `FIT\POISSON Y=A*EXP(-B*X)+C` `FIT\UPDATE YF`

By default, or if the `\NORMAL` qualifier is used, it is assumed that each data point has an error that is distributed as a normal distribution,

$$N(x;\mu,\sigma) \equiv \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \quad \text{for} \quad -\infty < x < \infty$$
$$= \sqrt{w/(2\pi)}e^{-\frac{w}{2}(x-\mu)^2}$$

where μ is the mean and σ is the standard deviation of the distribution. The weight w is defined as: $w = \frac{1}{\sigma^2}$.

If the `\POISSON` qualifier is used, the data errors are assumed to be distributed as a Poisson distribution,

$$P(x;\lambda) \equiv \frac{\lambda^x e^{-\lambda}}{x!} \quad \text{for} \quad x = 0, 1, 2, \ldots$$

where λ is the mean and the variance of the distribution.

Expression and parameters

The `expression` must result in a vector with the same length as the data vector, `y`. A maximum of twenty-five (25) fitting parameters are allowed in the `expression`. The fitting parameter values are altered during the fit. Fit parameters are created with the `SCALAR\VARY` command, and can be converted to fixed value scalars with the `SCALAR` command. If you use the `\RESET` qualifier, the fitting parameters will be reset to their original values after an unsuccessful fit, or a `control-c` abort.

If the `\VARNAMES` qualifier is used with the `FIT` command, a string array variable named

FIT$VAR will be made which will contain the names of the fitting parameter variables. The array length of FIT$VAR will be equal to the number of fit parameters.

Method

Suppose that you have N data points, y_k, for $k = 1, \ldots, N$, and the function to be fitted is $f(x, \boldsymbol{p})$, where $\boldsymbol{p}$ represents the M parameters $< p_1, p_2, \ldots, p_M >$. Define the likelihood of the parameters, given the data, as the probability of the data, given the parameters. We fit for the parameters, $\boldsymbol{p}$, by finding those values, $\boldsymbol{p}_{\text{min}}$ that maximize this likelihood. This form of parameter estimation is known as maximum likelihood estimation.

Some good references are:

- Practical Methods of Optimization, by R. Fletcher, 1980;
- Methods for Unconstrained Optimization Problems by J. Kowalik and M.R. Osborne, 1968;
- Statistical Methods in Experimental Physics, by W.T. Eadie, et.al., 1971;
- Mathematical Statistics, by John E. Freund, 1971;
- Formulae and Methods in Experimental Data Evaluation, Volume 3, Elements of Probability and Statistics, by Siegmund Brandt, 1984;
- Numerical Recipes – The Art of Scientific Computing, by W.H. Press, et.al. 1986.

Consider the likelihood function $\mathcal{L}(\boldsymbol{p}) \equiv \prod_{k=1}^{N} P(x_k, \boldsymbol{p})$ where P is the probability density, which depends on the random variable x and the parameters $\boldsymbol{p}$. $\mathcal{L}$ is a measure for the probability to observe just the particular sample we have, and is called an *a-posteriori probability* since it is computed after the sampling is done. The best estimates for $\boldsymbol{p}$ are the values which maximize $\mathcal{L}$. But maximizing the logarithm of $\mathcal{L}$ also maximizes $\mathcal{L}$, and maximizing $\ln(\mathcal{L})$ is equivalent to minimizing $-\ln(\mathcal{L})$. So, the goal becomes minimizing the log likelihood function:

$$-L(\boldsymbol{p}) \equiv -\ln \mathcal{L}(\boldsymbol{p}) = -\sum_{k=1}^{N} \ln P(x_k, \boldsymbol{p})$$

Let $\boldsymbol{p}^0$ be the initial values given for $\boldsymbol{p}$. The goal is to find a $\nabla \boldsymbol{p}$ so that $\boldsymbol{p}^1 = \boldsymbol{p}^0 + \nabla \boldsymbol{p}$ is a better approximation to the data. We use the iterative Gauss-Newton method, and the series $\boldsymbol{p}^1, \boldsymbol{p}^2, \boldsymbol{p}^3, \ldots$ will hopefully converge to the minimum, $\boldsymbol{p}_{\text{min}}$.

Generally, the Gauss-Newton method is locally convergent when χ^2 is zero at the minimum. Serious difficulties arise when f is sufficiently nonlinear and χ^2 is large at the minimum.

The Gauss-Newton method has the advantage that linear least squares problems are solved in one iteration.

Consider the Taylor expansion of $L(\boldsymbol{p})$:

$$L(\boldsymbol{p}) = L(\boldsymbol{p}^0) + \sum_{j=1}^{M} \left.\frac{\partial L}{\partial p_j}\right|_{\boldsymbol{p}^0} p_j + \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} \left.\frac{\partial^2 L}{\partial p_i \partial p_j}\right|_{\boldsymbol{p}^0} p_i p_j + \cdots$$

Define the arrays $\boldsymbol{b}$, $\boldsymbol{B}$ and $\boldsymbol{c}$:

$$[\boldsymbol{b}]_i \equiv -\left.\frac{\partial L}{\partial p_i}\right|_{\boldsymbol{p}^0} \quad \text{for} \quad i = 1, 2, \ldots, M$$

$$[\boldsymbol{B}]_{ij} \equiv \left.\frac{\partial^2 L}{\partial p_i \partial p_j}\right|_{\boldsymbol{p}^0} \quad \text{for} \quad i, j = 1, 2, \ldots, M$$

$$\boldsymbol{c} \equiv L(\boldsymbol{p}^0)$$

If we linearize, that is, assume that $\frac{\partial^2 \ln P}{\partial p_i \partial p_j} \to 0$, then $L(\boldsymbol{p}) \approx \boldsymbol{c} - \boldsymbol{b} \cdot \boldsymbol{p} + \frac{1}{2} \boldsymbol{p} \cdot \boldsymbol{B} \cdot \boldsymbol{p}$, and so $\nabla L = \boldsymbol{B} \cdot \boldsymbol{p} - \boldsymbol{b}$. The problem has reduced to solving the matrix equation $\boldsymbol{B} \cdot \nabla \boldsymbol{p} = \boldsymbol{b}$.

Note: The partial derivatives are approximated numerically using a central difference approximation:

$$\frac{\partial f(x_k, \boldsymbol{p})}{\partial p_i} = \frac{f(x_k, \boldsymbol{p} + \nabla \boldsymbol{p}_i) - f(x_k, \vec{p} - \nabla \boldsymbol{p}_i)}{2 \nabla \boldsymbol{p}_i}$$

Tolerance

Syntax

```
FIT\TOLERANCE eps y=expression
FIT\WEIGHTS\TOLERANCE w eps y=expression
FIT\ITMAX\TOLERANCE n eps y=expression
FIT\WEIGHTS\ITMAX\TOLERANCE w n eps y=expression
```

The \TOLERANCE qualifier allows the user to specify the fitting tolerance, which has a default value of 0.00001. This value is used in calculating the central difference formula for the partial derivatives, that is, $\frac{\partial f(x,p)}{\partial p}$ is approximated by

$$\frac{f(x, p(1 + \text{eps})) - f(x, p(1 - \text{eps}))}{(2p \times \text{eps})}$$

This value is also used to determine when the fit is successful.

Normal distribution

Assume that each data point, y_k, has an error that is independently random and distributed as a normal distribution, that is,

$$P(x_k,\boldsymbol{p}) = \frac{1}{\sigma_k\sqrt{2\pi}} e^{-\frac{1}{2}\left[\frac{y_k - f(x_k,\boldsymbol{p})}{\sigma_k}\right]^2}$$

where σ^2 is the variance, and $f(x_k,\boldsymbol{p})$ is the expression that we want to fit.

$$L(\boldsymbol{p}) = \sum_{k=1}^{N} \ln P(x_k,\boldsymbol{p}) = -\frac{1}{2}\sum_{k=1}^{N}\left[\frac{y_k - f(x_k,\boldsymbol{p})}{\sigma_k}\right]^2 + \text{constant}$$

The goal is to minimize the χ^2 function:

$$\chi^2(\boldsymbol{p}) \equiv \sum_{k=1}^{N}\left[\frac{y_k - f(x_k,\boldsymbol{p})}{\sigma_k}\right]^2 = \sum_{k=1}^{N} w_k\left[y_k - f(x_k,\boldsymbol{p})\right]^2$$

where the weights, w_k, are defined as: $w_k \equiv 1/\sigma_k^2$. Consider the Taylor expansion of χ^2:

$$\chi^2(\boldsymbol{p}) = \chi^2(\boldsymbol{p}^0) + \sum_{j=1}^{M}\left.\frac{\partial\chi^2}{\partial p_j}\right|_{\boldsymbol{p}^0} p_j + \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\left.\frac{\partial^2\chi^2}{\partial p_i\partial p_j}\right|_{\boldsymbol{p}^0} p_i p_j + \cdots$$

Define the arrays $\boldsymbol{b}$, $\boldsymbol{B}$ and $\boldsymbol{c}$:

$$\begin{aligned}
[\boldsymbol{b}]_i &\equiv -\left.\frac{\partial\chi^2}{\partial p_i}\right|_{\boldsymbol{p}^0} = 2\sum_{k=1}^{N} w_k\left[y_k - f(x_k,\boldsymbol{p})\right]\left.\frac{\partial f(x_k,\boldsymbol{p})}{\partial p_i}\right|_{\boldsymbol{p}^0} \quad \text{for} \quad i = 1,2,\ldots,M \\
[\boldsymbol{B}]_{ij} &\equiv \left.\frac{\partial^2\chi^2}{\partial p_i\partial p_j}\right|_{\boldsymbol{p}^0} = 2\sum_{k=1}^{N} w_k \left.\frac{\partial f(x_k,\boldsymbol{p})}{\partial p_i}\right|_{\boldsymbol{p}^0}\left.\frac{\partial f(x_k,\boldsymbol{p})}{\partial p_j}\right|_{\boldsymbol{p}^0} \quad \text{for} \quad i,j = 1,2,\ldots,M \\
\boldsymbol{c} &\equiv \chi^2(\boldsymbol{p}^0)
\end{aligned}$$

Linearize and the problem reduces to solving the matrix equation $\boldsymbol{B}\cdot\nabla\boldsymbol{p} = \boldsymbol{b}$.

χ^2 and weights

Syntax
```
FIT\WEIGHTS w y=expression
FIT\WEIGHTS\ITMAX w n y =expression
FIT\WEIGHTS\ITMAX\TOLERANCE w n eps y=expression
```

The weight at each point defaults to one (1), if a weight vector is not entered. Weights only make sense with a normal distribution, and are ignored when used with the `\POISSON` qualifier.

To make use of a weight array, the `\WEIGHTS` qualifier *must* be entered. If the `\WEIGHTS` qualifier is used, the weight vector, `w`, will then be expected. The weights are assigned to

the dependent variable in a one-to-one fashion, that is, the weight vector must be the same length as the data vector, `y`. If the `\ITMAX` qualifier is used, the weight comes before the iteration maximum in the command parameter list. If the `\TOLERANCE` qualifier is used, the iteration maximum comes before the tolerance in the command parameter list.

By default, the zero elements of the weight vector are used when calculating the number of degrees of freedom. If the `\-ZEROS` qualifier is used with the `\WEIGHTS` qualifier, then the zero elements of the weight vector will not be used when calculating the number of degrees of freedom. This could have an affect on the calculation of the confidence level, the χ^2 per degrees of freedom, and $\boldsymbol{E2}$, the root mean square total errors of estimate.

If the `\CHISQ` qualifier is used, a new scalar, named `FIT$CHISQ`, will be made with value equal to the total $\chi^2 = \sum w_k\left[y_k - f(x_k,\boldsymbol{p}_{\text{min}})\right]^2$ where w_k represents the optional weight at each data point y_k, f is the expression to be fitted, and $\boldsymbol{p}_{\text{min}}$ are the best values of the $\boldsymbol{p}$ parameters.

Hint for physicists

Very often, the data to be fitted is a histogram of physical events. In that case, since each bin would follow a multinomial distribution, the error is equal to $\sqrt{f}$, where f is the expression you are trying to fit. Of course, since you don't know the parameter values yet, you don't actually know f, so you approximate by using the y data values. In the limit, these results are the same. In the case of a large number of bins, the variance can be approximated by $\sqrt{y}$. Hence, the correct weighting factor that will give properly normalized errors is $w = 1/y$, and the corresponding one standard deviation error, $\sigma = \boldsymbol{E2}/\sqrt{\chi^2/n}$, where $\boldsymbol{E2}$ is the standard error and n is the number of degrees of freedom, usually equal to the number of data points minus the number of parameters, $(N - M)$.

Degrees of freedom

If the `\FREE` qualifier is used, then the number of degrees of freedom for the fit is output into an automatically created scalar named `FIT$FREE`. The number of degrees of freedom is either the number of data values minus the number of parameters, or, if the `\-ZEROS` qualifier is also used, the number of non-zero weights minus the number of parameters.

Poisson distribution

Assume that each data point has an error that is independently random and distributed as a Poisson distribution. The log likelihood function, $L(\boldsymbol{p})$, as a function of the fit parameters, $\boldsymbol{p}$, is minimized using a Gauss-Newton method. Since logarithms are involved, a good first approximation is required before starting the Poisson fit, so try a normal fit first, and use the resultant parameter values to start off the Poisson fit.

Weights do not have meaning, and so are not used, in a Poisson fit.

Assume that each data point, y_k, has an error that is independently random and distributed as a Poisson distribution, that is, $P(x_k,\boldsymbol{p}) = f(x_k,\boldsymbol{p})^{y_k} e^{-f(x_k,\boldsymbol{p})}/y_k!$. We want to minimize:

$$-L = \sum_{k=1}^{N} \ln\left[\frac{f(x_k,\boldsymbol{p})^{y_k} e^{-f(x_k,\boldsymbol{p})}}{y_k!}\right] = \sum_{k=1}^{N}\left[y_k \ln\left(f(x_k,\boldsymbol{p})\right) - f(x_k,\boldsymbol{p}) - \ln\left(y_k!\right)\right]$$

but $\sum \ln(y_k!)$ is a constant. So, the goal is to minimize

$$\mathcal{P}(\boldsymbol{p}) \equiv \sum_{k=1}^{N}\left[y_k \ln\left(f(x_k,\boldsymbol{p})\right) - f(x_k,\boldsymbol{p})\right]$$

Consider the Taylor expansion of $\mathcal{P}$:

$$\mathcal{P}(\boldsymbol{p}) = \mathcal{P}(\boldsymbol{p}^0) + \sum_{j=1}^{M} \left.\frac{\partial \mathcal{P}}{\partial p_j}\right|_{\boldsymbol{p}^0} p_j + \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M} \left.\frac{\partial^2 \mathcal{P}}{\partial p_i \partial p_j}\right|_{\boldsymbol{p}^0} p_i p_j + \cdots$$

Define:

$$[\boldsymbol{b}]_i \equiv -\left.\frac{\partial \mathcal{P}}{\partial p_i}\right|_{\boldsymbol{p}^0} = -\sum_{k=1}^{N}\left[\frac{y_k}{f(x_k,\boldsymbol{p})}\frac{\partial f(x_k,\boldsymbol{p})}{\partial p_i} - \frac{\partial f(x_k,\boldsymbol{p})}{\partial p_i}\right] \quad \text{for} \quad i = 1,2,\ldots,M$$

$$[\boldsymbol{B}]_{ij} \equiv \left.\frac{\partial^2 \mathcal{P}}{\partial p_i \partial p_j}\right|_{\boldsymbol{p}^0} = -\sum_{k=1}^{N}\frac{y_k}{f^2(x_k,\boldsymbol{p})}\frac{\partial f(x_k,\boldsymbol{p})}{\partial p_i}\frac{\partial f(x_k,\boldsymbol{p})}{\partial p_j} \quad \text{for} \quad i,j = 1,2,\ldots,M$$

$$\boldsymbol{c} \equiv \mathcal{P}(\boldsymbol{p}^0)$$

Then: $\mathcal{P}(\boldsymbol{p}) \approx \boldsymbol{c} - \boldsymbol{b}\cdot\boldsymbol{p} + \frac{1}{2}\boldsymbol{p}\cdot\boldsymbol{B}\cdot\boldsymbol{p}$. Linearize, and the problem reduces to solving the matrix equation $\boldsymbol{B}\cdot\nabla\boldsymbol{p} = \boldsymbol{b}$.

χ^2 of the fit

If the `\CHISQ` qualifier is used, a new scalar, named `FIT$CHISQ`, will be made with value equal to the total $\chi^2 = 2\sum\left[f(x_k,\boldsymbol{p}_{\text{min}}) - y_k + y_k \ln\frac{y_k}{f(x_k,\boldsymbol{p}_{\text{min}})}\right]$ where f is the expression to be fitted, and $\boldsymbol{p}_{\text{min}}$ are the best values of the parameters $\boldsymbol{p}$. This assumes that y_k is the outcome of a Poisson process.

Correlation and covariance

An indication of the accuracy of the fit is displayed in the output under the names $\boldsymbol{E1}$ and $\boldsymbol{E2}$.

$$[\boldsymbol{E1}]_i \equiv \sqrt{\left[\boldsymbol{B}^{-1}\right]_{ii}} \quad \text{for} \quad i = 1,\ldots,M$$

$$[\boldsymbol{E2}]_i \equiv [\boldsymbol{E1}]_i \sqrt{\frac{1}{n}\sum_{k=1}^{N} w_k [y_k - f(x_k,\boldsymbol{p})]^2} \quad \text{for} \quad i = 1,\ldots,M$$

where n is the number of degrees of freedom, and where $\left[\boldsymbol{B}^{-1}\right]_{ii}$ are the diagonal elements of the inverse of the matrix $\boldsymbol{B}$. $\boldsymbol{B}^{-1}$ is called the covariance matrix. The $[\boldsymbol{E1}]_i$ are called the root mean square statistical errors of estimate, while the $[\boldsymbol{E2}]_i$ are called the root mean square total errors of estimate, or standard errors.

The accuracy of the parameters in a linear fit is $p_i \pm [\boldsymbol{E2}]_i$ for $i = 1,\ldots,M$. In the linear case, for the standard error $\boldsymbol{E2}$ to be correct, the weights w_k must be proportional to $1/\sigma_k^2$, where σ_k is the standard deviation of the probability distribution of y_k. In the nonlinear case, $\boldsymbol{E2}$ does not have the same statistical significance.

If the `\COVMAT` qualifier is used, a matrix called `FIT$COVM` will be created which will contain $\boldsymbol{B}^{-1}$. If the `\CORRMAT` qualifier is used, a matrix with the name `FIT$CORR` will be created which will contain the correlation matrix for the fit. The size of these matrices will be M by M. If the `\E1` qualifier is used, then the root mean square statistical error for each fit parameter are output into an automatically created vector named `FIT$E1`. If the `\E2` qualifier is used, then the root mean square total error of estimate for each parameter are output into an automatically created vector named `FIT$E2`. The values are stored in these vectors in the order corresponding to the order in which the parameters appeared in the expression. The length of these vectors will be equal to the number of parameters in the fit expression.

Confidence level of the fit

If the `\CL` qualifier is used, a new scalar, named `FIT$CL`, will be made with value equal to the confidence level:

$$CL(\chi^2) = \frac{1}{2^{n/2}\Gamma(n/2)}\int_{\chi^2}^{\infty} t^{n/2-1}e^{-t/2}dt$$

where n is the degrees of freedom, usually equal to the number of data points minus the number of parameters, $(N-M)$. The confidence level is the probability that a random repeat of the given experiment would observe a worse χ^2, assuming the correctness of the model.

Number of iterations

Syntax

```
FIT\WEIGHTS\ITMAX w n y =expression
FIT\ITMAX n y =expression
FIT\ITMAX\POISSON n y =expression
```

The `\ITMAX` qualifier allows the user to specify the maximum number of iteration steps for the fit. When this maximum number is reached, the fit will stop, and the variable parameters will be updated to their last values. The fit will also stop if the fit is successful before this maximum iteration number is reached. If the `\WEIGHT` qualifier is also used, the weight array comes before the iteration number in the command parameter list.

Informational messages

By default, information on the progress of the fit, as well as the results, are displayed on the monitor screen. If the `\NOMESSAGES` qualifier is used, these informational messages will be suppressed.

Update after a fit

Syntax	`FIT\UPDATE yout`

The `FIT\UPDATE` command evaluates the previously fitted expression, that is, the expression in the last `FIT` command, for the current parameter values, and stores this result in the vector `yout`.

This is exactly equivalent to entering: `yout`=previously_fitted_expression
and is provided only to obviate the necessity of re-entering a complicated expression.

Note: The vector name `yout` usually differs from the name of the vector being fitted to avoid destroying the original data.

FMIN

Syntax	`FMIN x y xlo xhi expression`
Qualifier	`\MESSAGES`
Defaults	`\NOMESSAGES`
Example	`FMIN\-MESSAGES X Y -10 10 2*X^2-10*X+5`

The `FMIN` command returns the location and value of the local minimum in the range `xlo` to `xhi` of the specified `expression`. This expression must be a function of the independent variable `x`, which must be a scalar.

The value of `x` is interpreted as the user's initial guess for the location of the local minimum. If the value of `x` is outside the range `xlo` to `xhi`, then the midpoint, `(xlo+xhi)/2` is chosen as this initial guess.

On output, the scalar `x` will contain the location of the local minimum, and scalar `y` will contain the value of this minimum.

The `expression` must contain the variable `x`, and may contain other scalars, but must *not* contain any other non-scalar variables.

Informational messages

By default, the value of the local minimum, its location, and the upper limit on the error are displayed on the monitor screen. If the `\NOMESSAGES` qualifier is used, this informational message will be suppressed.

Example

The following script demonstrates how you can use the `FMIN` command to find a local minimum. See Figure 2.12.

```
XMIN=-10
XMAX=10
XD=[XMIN:XMAX:.1]
GRAPH XD 2*XD^2-10*XD+5
GET
 YMIN YMIN
 YMAX YMAX

X=0
FMIN X Y -10 10 2*X^2-10*X+5
GRAPH\NOAXES [XMIN;XMAX] [Y;Y]
GRAPH\NOAXES [X;X] [YMIN;YMAX]
TEXT 'y=2x<^>2<_>-10x+5'
TEXT 'minimum value= '//rchar(y)//' at x='//rchar(x)
```

FZERO

Syntax	`FZERO x expression`
Qualifier	`\MESSAGES`
Defaults	`\NOMESSAGES`
Example	`FZERO\-MESSAGES X SIN(X)/X`

The `FZERO` command returns the zeros, or roots, of `expression`. This expression must be a function of the independent variable `x`, which must be a vector.

If the length of vector `x` is N, then a maximum of N roots of the `expression` will be found. On output, the vector `x` will contain these N roots.

The `expression` must contain the variable `x`, and may contain scalars, but must *not* contain any other non-scalar variables.

Muller's method

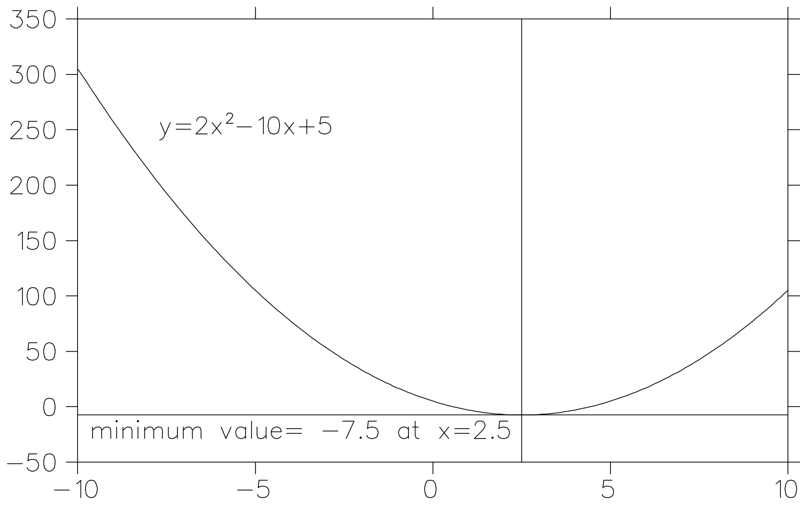


Figure 2.12: Finding a local minimum with the FMIN command

If z_{i-2}, z_{i-1} and z_i are three approximations to a root, the next approximation to the root, z_{i+1}, is taken as a zero of the quadratic that passes through $f(z_{i-2})$, $f(z_{i-1})$ and $f(z_i)$. The iteration continues by dropping z_{i-2} and repeating the quadratic fit for z_{i-1}, z_i and z_{i+1} and associated function values.

If $(r-1)$ roots have been found, the r^{th} root is found by deflating $f(z)$ and solving the equation $f_r(z)$ where

$$f_r(z) = f(z) / \prod_{i=1}^{r-1} (z - z_i)$$

where the z_i are the previously found roots. The roots are found one at a time in approximately increasing order.

The iteration stops when either

$$|(z_{i+1} - z_i)/z_{i+1}| < e_1 \quad \text{or} \quad f_r(z_{i+1}) \quad \text{and} \quad f(z_{i+1}) < e_2$$

where e_1 and e_2 are error tolerances. For more information refer to
A Method for Solving Algebraic Equations Using an Automatic Computer
by D.E. Muller, M.T.A.C. 10, 1956, pages 208-215.

In the PHYSICA implementation of Muller's method, $e_1 = 10^{-7}$ and $e_2 = 10^{-20}$. The values

in vector x are interpreted as the user's initial guesses for the location of the roots. If x[i] $= 0$, the starting approximations for the ith root are taken as: -1, 1, and 0. If x[i] $\neq 0$, the starting approximations for the ith root are taken as: 0.9x[i], 1.1x[i], and x[i]. On each iteration, if $|r_t - r_p| < e_3$, where r_t is the approximation to a root, and r_p is a previously found root, then r_t is replaced by $r_t + e_4$. In PHYSICA, $e_3 = 10^{-20}$ and $e_4 = 10^{-4}$. If a root is not found in 60 iterations, the search is terminated.

Informational messages

By default, the values of the roots are displayed on the monitor screen. If the \NOMESSAGES qualifier is used, informational messages will be suppressed.

Example

The following script demonstrates how you can use the FZERO command to find roots. See Figure 2.13.

```
XMIN=-30
XMAX= 10
XD=[XMIN:XMAX:.1]
SET %XLABSZ 5
LABEL\XAXIS ‘y=e<^>0.4x<_>-0.4x-9’
GRAPH XD EXP(0.4*XD)-0.4*XD-9
GET
 YMIN YMIN
 YMAX YMAX

NROOTS=2
X[1:NROOTS]=0
FZERO X EXP(0.4*X)-0.4*X-9
DO J = [1:NROOTS]
  GRAPH\NOAXES [X[J];X[J]] [YMIN;YMAX]
  DISPLAY ‘please position the string:  root at ’//rchar(x[j])
  TEXT ‘root at ’//rchar(x[j])
ENDDO
ZEROLINE\HORIZONTAL
```

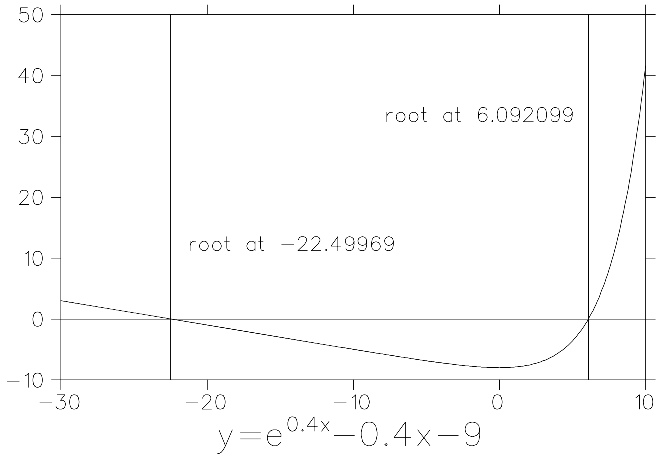


Figure 2.13: Finding roots with the FZERO command

GENERATE

Syntax	`GENERATE x min inc ,, npts` `GENERATE x min ,, max npts` `GENERATE x min inc max` `GENERATE\RANDOM x min max npts`
Qualifier	`\RANDOM`
Default	`\-RANDOM`

The `GENERATE` command creates a new vector, `x`.

By default, the new vector, `x`, will be generated according to the formula: $\texttt{x[i]} = \texttt{min} + (i-1)$ `inc` for $1 \le \texttt{i} \le \texttt{npts}$.

The minimum value, `min`, must be given. Two other values are also required: the increment and the number of points, or the maximum and the number of points, or the increment and the maximum value.

Increment and number of points given

Syntax `GENERATE x min inc ,, npts`

If the increment and the number of points are given, the above formula is applied directly.

Example

After the command: `GENERATE X -1 .5 ,, 4`
the vector `X = [-1; -0.5; 0; 0.5]`

Maximum and number of points given

Syntax `GENERATE x min ,, max npts`

If maximum value and the number of points are given, the increment is calculated:
$\texttt{inc} = (\texttt{max - min})/(\texttt{npts} - 1)$ and then used in the usual formula.

Example

After the command: `GENERATE X -1 ,, 2 4`
the vector `X = [-1; 0; 1; 2]`

Increment and maximum given

Syntax `GENERATE x min inc max`

If the increment and the maximum value are given, the number of points will be ignored if entered. The usual formula will be applied until the next value would be greater than `max`, the maximum. If the calculated maximum is different than the given maximum, a warning message will be displayed on the monitor screen. The calculated maximum will be the last value stored in the vector.

Example

After the command `GENERATE X -1 .41 1`
the warning message:

```
GENERATE warning: calculated maximum = 0.63999999E+00
                       given maximum = 0.10000000E+01
```

will be displayed on the monitor screen and the vector `X = [-1; -0.59; -0.18; 0.23; 0.64]`

Random numbers

Syntax `GENERATE\RANDOM x min max npts`

If the `\RANDOM` qualifier is used, the vector `x` will be filled with `npts` random numbers that fall between min and max. No increment should be given.

The initial value for the random number seed is 12345. Every time a random number is requested, either from the GENERATE\RANDOM command or from the RAN function, the seed is updated. You can change the seed value with the SET SEED command.

Example

After the command GENERATE\RANDOM X 1 2 5
the vector X = [1.198525; 1.897874; 1.238289; 1.367985; 1.381705]

GET

Syntax	GET { keyword { value }}
Examples	GET %XLAXIS XLX GET GET NSXINC

The GET command gets the values of the GPLOT plot characteristic keywords as well as the PHYSICA specific keywords. Use the SET command to change the values of these keywords.

If the GET command is entered with no parameters, more than one keyword value can be obtained without re-entering the GET command. Other keywords and values will be requested, until a blank line is entered, at which time the user is put back into command line entry mode. If the GET command is used in this way in a script file, the blank line is *necessary* to indicate that the GET command is finished.

If a keyword is entered with the GET command, then only that one keyword's value can be obtained with that command.

If an output variable is not entered after the keyword, the current value of that keyword will be displayed on the terminal screen. If an output variable is entered, a variable will be created and the current value of that keyword assigned to that variable.

Note: The keywords FONT, CUNITS, UNITS, VERSION, VERSIONDATE, and AUTOSCALE return a string instead of a numeric value.

Examples

To display the current value of XMIN, enter: GET XMIN

To obtain the current value of XMIN and then change it to XMIN -10, and to set the value of XMAX to XMIN+100, enter:


```
...
GET XMIN A    ! makes scalar A
SET
 XMIN A-10
 XMAX A+100   ! don't forget the blank line

...
```

To display the current graphics font name, enter GET FONT

The command: GET FONT TXT[3]
places the current font name into the 3^{rd} element of the array text variable TXT.

The GPLOT keywords

See **Appendix A** for descriptions of all the GPLOT plotting characteristic keywords. The tables produced by the DISPLAY MENU contain most of the keywords that can be accessed with the SET and GET commands, along with their current values.

The GPLOT keywords: MASK, ALIAS, PMODE, PTYPE, and ERRBAR, should *not* be changed in PHYSICA, as these are internally adjusted and used by various commands.

The PHYSICA keywords

ARROLEN

ARROLEN is the arrow head length as a fraction of the total arrow shaft length. It is used for arrows drawn with the FIGURE command.

ARROTYP

ARROTYP controls the type of arrow drawn with the FIGURE command. See Table 2.54 on page 231 and Figure 2.24 on page 230.

ARROWID

ARROWID is the arrow head width as a fraction of the total arrow shaft length. It is used for arrows drawn with the FIGURE command.

PCHAR

Optional parameters: symbol { size { colour { angle }}}

PCHAR controls the plotting symbols, or the appearance of the histogram bars, when the GRAPH command is entered. GET PCHAR obtains the plotting symbol values or arrays, as set with the SET PCHAR command. It is not necessary to set the size, colour or angle with the SET PCHAR command, but if these are not set, you cannot request their values. The type of output variable that will be made for each parameter depends on the type of variable used in the SET PCHAR command.

AUTOSCALE

Note: The value of the AUTOSCALE keyword is a string instead of a numeric value. So, if you enter: GET AUTOSCALE X, the variable X will be a string variable.

The AUTOSCALE keyword controls autoscaling for graph axes. Autoscaling remains in effect until either the command SET AUTOSCALE OFF is entered, or the SCALES command is entered. Autoscaling affects commands that draw axes, for example, the commands GRAPH, CONTOUR, DENSITY, REPLOT, and SLICES.

Autoscaling means to automatically choose the minimum and maximum values for the axes, as well as the number of large, numbered, tic marks for the axes. The type of autoscaling that is done depends on the keyword that is used with the command.

keyword	*result*
ON	Autoscale the horizontal and the vertical axes
OFF	turn off all autoscaling, the axes will appear as they are currently set
COMMENSURATE	Autoscale the horizontal and vertical axes and change the lengths of the axes so that they will be commensurate
XAXIS	Autoscale the horizontal axis only, the vertical axis will remain as currently set
YAXIS	Autoscale the vertical axis only, the horizontal axis will remain as currently set

When the \VIRTUAL qualifier is used, the virtual minima and maxima for the axes will be determined, so that the axes may not begin or end at a large tic mark. If the keyword ON is used, both x- and y-axes will have virtual minima and maxima. If the keyword XAXIS is used, only the x-axis will have virtual minimum and maximum. If the keyword YAXIS is used, only the y-axis will have virtual minimum and maximum.

CNTSEP

CNTSEP, or %CNTSEP, is the separation between contour labels in the CONTOUR command. %CNTSEP is the separation as a percentage of the height of the window, that is, YUWIND-YLWIND, while CNTSEP is the separation expressed in centimeters or inches, depending on the units type as set with the SET UNITS command.

LABSIZ

LABSIZ, or %LABSIZ, is the size of the contour labels in the CONTOUR command. %LABSIZ is the size as a percentage of the height of the window, that is, YUWIND-YLWIND, while LABSIZ is the size expressed in centimeters or inches, depending on the units type as set with the SET UNITS command.

LEGSIZ

LEGSIZ, or %LEGSIZ, is the size of the contour plot and density plot legend entries. %LEGSIZ is the size as a percentage of the height of the window, that is, YUWIND-YLWIND, while LEGSIZ is the size expressed in centimeters or inches, depending on the units type as set with the SET UNITS command.

LEGFRMT

Note: The value of the LEGFRMT keyword is a string instead of a numeric value. So, if you enter: GET LEGFRMT X, the variable X will be a string variable.

The numeric legend entries drawn by the DENSITY and CONTOUR commands are written using the LEGFRMT format.

ERRFILL

If the \ERRFILL qualifier is used with the READ\VECTOR command, an invalid field in the data file causes either the entire record to be filled with the value of ERRFILL if a format is used, or only that invalid field will be filled with ERRFILL if no format is used.

FILL

FILL is used in the FIGURE command, with the fillable figures: BOX, POLYGON, WEDGE, CIRCLE, ELLIPSE, and ARROWs with closed heads. It is also used for filling the boxes with the DENSITY\BOXES command. See Table 2.55 on page 233 for a description of the interpretations of the FILL keyword.

See the SET HATCH command for information on changing the hatch pattern definitions. See the DISPLAY HATCH command for information on how to display examples of the hatch patterns.

HATCH

Optional parameters: `n { v a }`

The `GET HATCH` command is used for obtaining the hatch pattern definitions that are used for text bolding, for filling areas under curves or histograms, and for use by the `TILE`, `PIEGRAPH`, and `FIGURE` commands.

The `SET HATCH` command is used for changing the hatch pattern definitions. The `SET HATCH` command does *not* choose the hatch pattern to be used by other commands. It only alters the definition of a hatch pattern.

If just the keyword `HATCH` is entered, a table of the spacings and angles for all ten hatch patterns is displayed. If the hatch pattern number, `n`, is entered, then only pattern `n` will be displayed. If the hatch pattern number, an output vector, `v`, and an output scalar, `a`, are entered, then `v` will contain the spacings and `a` will contain the angle for that pattern number.

The hatch pattern number, `n`, should be between one and ten. A hatch pattern is composed of an angle and from one to ten spacings. The default spacings and angles are listed in Table 2.56 on page 235. The angles are in degrees and the spacing lengths, by default, are expressed in centimeters, but if the units are changed to inches, with the `SET UNITS` command, the lengths will be converted to inches. See Figure 2.7 on page 61 for examples of the hatch patterns. See the `DISPLAY` command, page 58, for information on how to display examples of the hatch patterns.

When an object is being filled, a line is drawn inside the object at the specified angle, then a parallel line is drawn at the first spacing, and so on for the number of spacings in that pattern. This process is repeated until the object is filled.

LINE

Optional parameters: `n { v }`

The `GET LINE` command is used for obtaining the definition of the line types that are used by the commands: `GRAPH`, `LINE`, `PICK`, `ELLIPSE`, `FIGURE`, and `ZEROLINES`.

The `SET LINE` command is used for changing the definition of the line types. This command does *not* choose the line type to be used by other commands. It only alters the definition of a line type. To choose a line type, use the `SET LINTYP` command.

If just the keyword `LINE` is entered, a table of the spacings for all ten line types is displayed. If the line type number, `n`, and an output vector, `v`, are entered, then nothing will be displayed. For example, to get line type 2 into vector `X2`, enter:

```
GET LINE 2 X2
```

See the `SET LINE` command for information on how the line types are defined. There are ten line types available. The defaults are listed in Table 2.58 on page 236. The lengths are expressed in centimeters, the default, but if the units are changed to inches, with the `SET UNITS` command, the lengths will be converted to inches. See Figure 2.8 on page 62 for examples of the default line types. See the `DISPLAY` command, page 58, for information on how to display examples of the line types.

TENSION

`TENSION` controls the spline tension for the functions using cubic splines:

`DERIV`, `INTEGRAL`, `INTERP`, `SMOOTH`, `SPLINTERP`, and `SPLSMOOTH`.

SEED

`SEED` is the random number seed value. This seed is updated whenever the `GENERATE\RANDOM` command is entered, or the `RAN` is used.

POSTRES

`POSTRES` controls the PostScript graphics output resolution, in dots per inch. This applies to dot filled text characters and dot types of `DENSITY` plots. The resolution can be changed at any time, so different parts of a single drawing can be drawn with different resolutions.

SPEED

`SPEED` controls the pen plotter speed. This applies to Hewlett-Packard, Houston, and Roland RDGL II pen plotters. The speed can be changed at any time, so different parts of a single drawing can be drawn at different speeds.

WIDTH

`WIDTH` controls the character width of the alphanumeric monitor screen. The value for `WIDTH` should be between 2 and 132.

XPREV

XPREV is the last world x-coordinate that was drawn by any graphics command. The value of this keyword is automatically updated.

YPREV

YPREV is the last world y-coordinate that was drawn by any graphics command. The value of this keyword is automatically updated.

NCURVES

NCURVES is the total number of data curves that have been drawn, using the GRAPH command, since the last CLEAR command. The value of this keyword is automatically updated.

UNITS

Note: The value of the UNITS keyword is a string instead of a numeric value. So, if you enter: GET UNITS X, the variable X will be a string variable.

UNITS controls the plotting units type, either centimeters, CM, the default, or inches, IN.

CUNITS

Note: The value of the CUNITS keyword is a string instead of a numeric value. So, if you enter: GET CUNITS X, the variable X will be a string variable.

CUNITS is the units type for the graphics cursor readout when the graphics cursor is invoked by the PICK, PEAK, LINE, or FIGURE command when running under X Windows and mouse button two is pressed. If WORLD is chosen, the numbers displayed depend on the current units type, either centimeters or inches, as chosen with SET UNITS. If GRAPH is chosen, the numbers displayed depend on the current graph axis scales.

FONT

Note: The value of the FONT keyword is a string instead of a numeric value. So, if you enter: GET FONT X, the variable X will be a string variable.

FONT controls the graphics font. For a list of the font names, see Table 2.60 on page 243.

The DISPLAY FONT command will draw a font table for any font.

VERSION

Note: The value of the VERSION keyword is a string instead of a numeric value. So, if you enter: GET VERSION X, the variable X will be a string variable.

This is the current program's version number. It is character valued, with a length of 5.

VERSIONDATE

Note: The value of the VERSIONDATE keyword is a string instead of a numeric value. So, if you enter: GET VERSIONDATE X, the variable X will be a string variable.

This is the current program's version date. It is character valued, with a length of 20.

SHOWHISTORY

SHOWHISTORY controls how many lines of history to display for each numeric variable as a result of the SHOW command.

SHOWHISTORY		
$n < 0$	$\rightarrow$	all stored history lines will be displayed
$n = 0, 1$	$\rightarrow$	only the latest history line will be displayed
$n > 0$	$\rightarrow$	a maximum of n lines of history will be displayed for each variable

MAXHISTORY

MAXHISTORY is the maximum number of history lines to store for each numeric variable. MAXHISTORY was added because if a variable had its value changed within a large DO loop, a new history line was added each time the loop was processed, which could lead to virtual memory problems.

WRAP

If WRAP $= 0$, history lines and string variable contents lines are not wrapped when displayed with the SHOW command. If WRAP is non-zero, these lines are wrapped.

DEVICE

Note: The value of the DEVICE keyword is a string instead of a numeric value. So, if you enter: GET DEVICE X, the variable X will be a string variable.

This is the current hardcopy device as chosen with the DEVICE command.

GLOBALS

Syntax `GLOBALS`

This command only works under VAX/VMS.

The `GLOBALS` command displays the names of global sections to which you have access. This command is meant to be used in conjunction with the `MAP\FIOWA` or the `MAP\FIOWABIG` commands.

GRAPH

Syntax	`GRAPH { 'legendtext' } x y { ye1 { xe1 { ye2 { xe2 }}}}`
Qualifiers	`\AXESONLY`, `\NOAXES`, `\POLAR`, `\REPLOT`, `\HISTOGRAM`
Defaults	axes drawn, `\REPLOT`, `\NOPOLAR`, `\NOHISTOGRAM`, `legendtext` ignored
Examples	`GRAPH X Y` `GRAPH 'legend entry' X Y YERR XERR` `GRAPH\NOAXES X Y` `GRAPH\HISTOGRAM X Y` `GRAPH\POLAR RAD THETA`

		command
The `GRAPH` command draws:	data with axes	`GRAPH`
	just the data	`GRAPH\NOAXES`
	just the axes	`GRAPH\AXESONLY`

The data curve may be a histogram. The parameters must be vectors, but can be vectors of length one. The input vectors can have different lengths as the minimum length of all of the input vectors will be used.

Plotting symbols

The `SET PCHAR` command controls the plotting symbols, or the appearance of the histogram bars. For information on how to set the plotting symbol type, size, colour, and angle; as well as how to set hatch fill patterns, colours, and relative bar size for histograms, refer to the `SET PCHAR` command section.

Axis scaling

Autoscaling of the axes may apply if the two vectors, `x` and `y`, are entered. Refer to the `SET AUTOSCALE` command for information on how to set up autoscaling for the axes.

Use the `SCALES` command, page 226, to manually set the scales for the axes.

Graph legend

If `LEGEND` is `ON`, a legend entry is drawn into a legend frame box. A legend entry consists of a short line segment, with plotting symbol(s), and a string. The legend entry is drawn when the `GRAPH` command is entered. The string portion of the legend entry is expected as the first parameter of the `GRAPH` command, for example:

```
GRAPH 'legend entry' X Y
```

If `LEGEND` is `OFF`, a string entered as a first parameter with the `GRAPH` command is ignored. Refer to the `LEGEND` command, page 137, for more information on a graph legend.

Plotting data and axes

By default, if the `GRAPH` command is entered with *neither* the `\AXESONLY` qualifier *nor* the `\NOAXES` qualifier is used, then axes will be drawn as well as the data curve or histogram. Autoscaling will apply if it is on. For example:

```
GRAPH X Y                  ! plots axes and data curve
GRAPH\HISTOGRAM X Y        ! plots axes and histogram
```

Plotting axes only

If the `GRAPH\AXESONLY` command is entered with no parameters, autoscaling does not apply. The minima, maxima, and number of increments will be the same as the last set of axes drawn. The axes scales can be set up before entering the `GRAPH\AXESONLY` command, using the `SCALES` command. Autoscaling may apply if two parameters, `x` and `y`, are entered. For example:

```
SET AUTOSCALE ON       ! turn on autoscaling
GRAPH\AXESONLY X Y     ! plot axes autoscaled to the data


SCALES -5 5 0 10 20 0      ! -5 <= x <= 5  and  10 <= y <= 20
GRAPH\AXESONLY             ! plot the axes only
```

Plotting data only

The `GRAPH\NOAXES` command plots the data, but does not draw axes. Autoscaling will not apply. The `GRAPH\NOAXES` command overlays on an existing set of axes. For example:

```
GRAPH\NOAXES X Y       ! overlay data curve on current axes
GRAPH\HIST\NOAX X Y    ! overlay histogram on current axes
```

If the LEGEND is ON then the data variables are not necessary. That is, you can enter:

```
GRAPH\NOAXES 'legendentry'
```

and an entry is made to the legend, but no curves are plotted. The plotting character, as set by the SET PCHAR command, will be used in the legend.

Replotting data on a common scale

By default, the graph will be stored for replotting. If the \NOREPLOT qualifier is used, the graph will not be saved for replotting. The default is \REPLOT.

To redraw a graph with multiple data sets so that all the data will appear within the axis boundaries, use the REPLOT command in association with the SET AUTOSCALE command. For example:

```
SET AUTOSCALE ON   ! turn on autoscaling
SET PCHAR -1       ! set plotting character to 'box'
GRAPH X Y          ! plot data with axes
SET PCHAR -2       ! set plotting character to 'cross'
GRAPH\NOAXES U V   ! overlay another data curve
CLEAR\NOREPLOT     ! clears graphics but not the replot buffers
REPLOT             ! replot both data sets on common scale
```

Refer to the REPLOT command, page 208, for more information.

Histograms

It is possible to draw four types of histograms using the SET HISTYP approach, or you can use the \HISTOGRAM qualifier to plot a histogram with tails to $y = 0$ and profile along the x-axis.

Using the HISTYP keyword

Table 2.39 on page 118 shows the histogram type that will be produced depending on the value of HISTYP.

Using the \HISTOGRAM qualifier

HISTYP	*Result*
0	(default value) line graph, not a histogram
1	histogram with no tails and profile along the x-axis. You may control the width and colour of each individual bar.
2	histogram with tails to $y = 0$ and profile along the x-axis. You may control the filling pattern, width and colour of each individual bar
3	histogram without tails and profile along the y-axis. You may control the height and colour of each individual bar.
4	histogram with tails to $x = 0$ and profile along the y-axis. You may control the filling pattern, height and colour of each individual bar

Table 2.39: The HISTYP keyword

The \HISTOGRAM qualifier is inconsistent when used in conjunction with the \AXESONLY qualifier.

Using the \HISTOGRAM qualifier is equivalent to using a HISTYP setting of 2. A histogram *with tails* to $y = 0$ and profile along the x-axis will be plotted. The following three commands:

```
SET HISTYP 2      ! force histogram plotting
GRAPH X Y         ! plot axes and histogram
SET HISTYP 0      ! reset to default value
```

are equivalent to the single command:

```
GRAPH\HIST X Y
```

Filling

To fill the area under a histogram, you can use the SET PCHAR command. The SET PCHAR command allows you to fill each histogram bar with a different fill pattern. This only applies to histograms with tails, HISTYP $= 2$ or 4.

See the SET PCHAR command for more information on filling, colours, and relative bar size for histograms.

See the SET HATCH command, page 228, for information on changing the hatch pattern definitions.

See the DISPLAY command for information on how to display examples of the hatch patterns.

Polar coordinates

If the \POLAR qualifier is used, the input vectors, x and y, are assumed to represent polar coordinates, where x contains the radial components and y contains the angular components, in degrees. The polar coordinates are transformed to rectangular coordinates before plotting, but the vectors x and y are returned unchanged.

Error bars

Syntax `GRAPH { 'legendtext' } x y { ye1 { xe1 { ye2 { xe2 }}}}`

The optional vectors ye1, xe1, ye2, and xe2 are interpreted as errors for drawing error bars. You can have symmetric or asymmetric error bars.

Symmetric error bars

For symmetric error bars, the error variable should contain one half of the total error. See Table 2.40.

parameters present	*Result*
ye1 but not ye2	symmetric vertical error bars will be drawn at the point (x[j],y[j]), the error bar is drawn from y[j]-ye1[j] to y[j]+ye1[j]
xe1 but not xe2	symmetric horizontal error bars will be drawn at the point (x[j],y[j]), the error bar is drawn from x[j]-xe1[j] to x[j]+xe1[j]

Table 2.40: Symmetric error bars

Asymmetric error bars

For asymmetric error bars, the first error variable contains the lower error and the second error variable contains the upper error. See Table 2.41.

Error bar shape

The error bars will have "feet", that is, short line segments, one at each end of the error bar, which are perpendicular to the error bar. The size of the foot is the same as the size of the plotting symbol, which can be changed using the SET %CHARSZ command or by

parameters present	*Result*
ye1 and ye2	asymmetric vertical error bars are drawn at the point (x[j],y[j]), the error bar is drawn from y[j]-ye1[j] to y[j]+ye2[j]
xe1 and xe2	asymmetric horizontal error bars are drawn at the point (x[j],y[j]), the error bar is drawn from x[j]-xe1[j] to x[j]+xe2[j]

Table 2.41: Asymmetric error bars

entering a relative size vector with the SET PCHAR command. The error bar will be clipped at the boundaries of the plotting symbol if the symbol is symmetric under 90° rotations, for example, a box (symbol number 1).

Filling

To fill the area under a curve, use the SET LINTYP command. See also the histogram discussion above, for specifics on filling histograms. If $101 \leq$ LINTYP ≤ 110, then the hatch pattern LINTYP-100 is chosen. If $211 \leq$ LINTYP ≤ 299, then the dot fill pattern LINTYP-200 is chosen. The polygonal region defined by the (x, y) coordinate pairs, with the last point connected to the first, will be filled with the chosen hatch or dot pattern.

Hatch patterns

$101 \leq |$LINTYP$| \leq 110$ means to fill using a hatch pattern. The filling will be done with hatch pattern |LINTYP|-100. For example, if LINTYP $= 108$, then hatch pattern number 8 will be used.

A hatch pattern is composed of an angle and one to ten spacings. These spacings are simply cycled through as the region is being filled, that is, a line is drawn inside the region at the specified angle, then a parallel line is drawn at the first spacing, then another parallel line is drawn at the second spacing, and so on for the number of spacings in that pattern. This process is repeated until the region is filled. The hatch patterns can be redefined with the SET HATCH command and displayed with the DISPLAY FILL command. There are ten hatch patterns available.

Dot fill patterns

$211 \leq |$LINTYP$| \leq 299$ means to fill using a dot pattern. The filling will be done with dot pattern |LINTYP|-200. For example, if LINTYP $= 234$, then dot pattern 34 will be used. If LINTYP < 0, then the dots are erased instead of turned on.

A dot pattern is of the form: uv, where the digit u is the increment number of dots to light up horizontally, $1 \leq u \leq 9$, and the digit v is the increment number of dots to light up vertically, $1 \leq v \leq 9$. For example, a dot pattern of 34 means to light up every third dot horizontally and every fourth dot vertically. If uv is negative, then the dots are erased instead of turned on. Note that 200 is interpreted the same as 211, that is, every dot is lit.

PostScript output

For PostScript output, set the POSTRES keyword to the appropriate resolution for your hardcopy device, using the SET command. Use a combination of line thickness and resolution for a quicker resulting picture. For example, the Lexmark inkjet printer has a resolution of 360 dpi. A "good" picture can be obtained with $\texttt{POSTRES} = 180$ and $\texttt{LINTHK} = 2$.

Examples

The following script demonstrates the plotting of error bars. See Figure 2.14.

```
X=[0:20]              ! generate some "data"
Y=X^2-20*X+50         !
YEL=10*RAN(X)         ! lower error
YEU=10*RAN(X)         ! upper error
YES=20*RAN(X)         ! symmetric error
SET PCHAR -1          ! set plotting symbol
WINDOW 5              ! set window
LABEL\X ‘Asymmetric errors’
GRAPH X Y YEL,,YEU    ! plot with asymmetric errors
WINDOW 6              ! change windows
LABEL\X ‘Symmetric errors’
GRAPH X Y YES         ! plot with symmetric errors
```

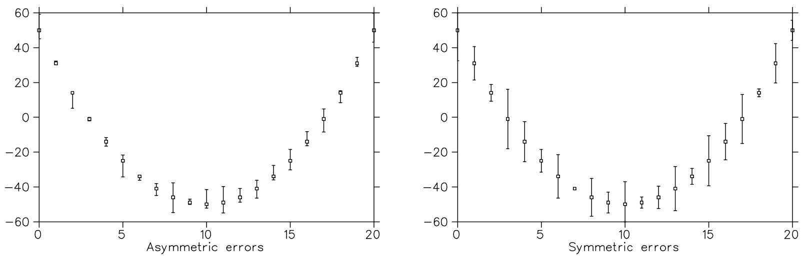


Figure 2.14: Plotting error bars with the GRAPH command

The following script demonstrates filling under area under a curve. See Figure 2.15.

```
X=[0:20]              ! generate some "data"
Y=X^2-20*X+50         !
XX[1]=X[1]            ! fix up the data so it starts at (x[1],0)
XX[2:LEN(X)+1]=X      ! and ends at (x[#],0)
XX[LEN(X)+2:LEN(X)+2]=X[#]
YY[1]=0
YY[2:LEN(Y)+1]=Y
YY[LEN(Y)+2:LEN(Y)+2]=0
SET PCHAR 0           ! no plotting symbol
SET LINTYP 233        ! dot pattern (every 3rd dot both directions)
WINDOW 15             ! set window
LABEL\X ‘Dot fill’    ! label the x axis
GRAPH XX YY           !
ZEROLINE\HORIZONTAL   ! draw horizontal line through (0,0)
SET LINTYP 108        ! hatch pattern number 6
WINDOW 16             ! change windows
LABEL\X ‘Hatch fill’  ! label the x axis
GRAPH XX YY           !
ZEROLINE\HORIZONTAL   ! draw horizontal line through (0,0)
```

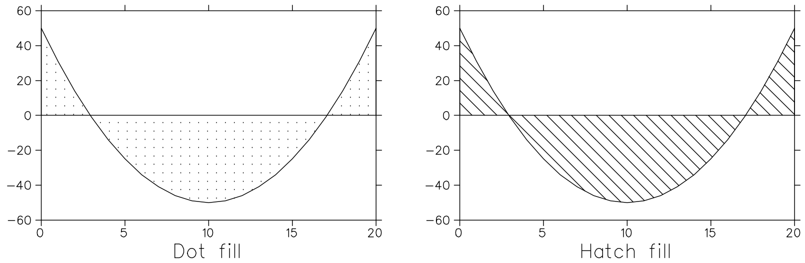


Figure 2.15: Filling the area under a curve drawn with the GRAPH command

The following script demonstrates plotting histograms and filled histograms. See Figure 2.16.

```
X=[0:24:3]               ! generate some "data"
Y=X^2-20*X+50            !
SET HISTYP 1             ! histogram with no tails
WINDOW 5                 ! set window
LABEL\X ‘HISTYP = 1’     ! label the x axis
GRAPH X Y                ! plot the histogram
WINDOW 6                 ! set window
LABEL\X ‘Narrow bars’    ! label the x axis
SET PCHAR 0 .8
GRAPH X Y                ! plot the histogram
WINDOW 7                 ! set window
SET HISTYP 2             ! histogram with tails
LABEL\X ‘Hatch pattern #8’
SET PCHAR 8 .8           ! hatch pattern #8
GRAPH X Y                ! plot the histogram
ZEROLINES\HORIZONTAL     ! draw horizontal line thru (0,0)
WINDOW 8                 ! set window
LABEL\X ‘Individual bar filling’
SET PCHAR [11:99:11] .8 ! each bar filled
GRAPH X Y                ! plot the histogram
ZEROLINES\HORIZONTAL     ! draw horizontal line thru (0,0)
```

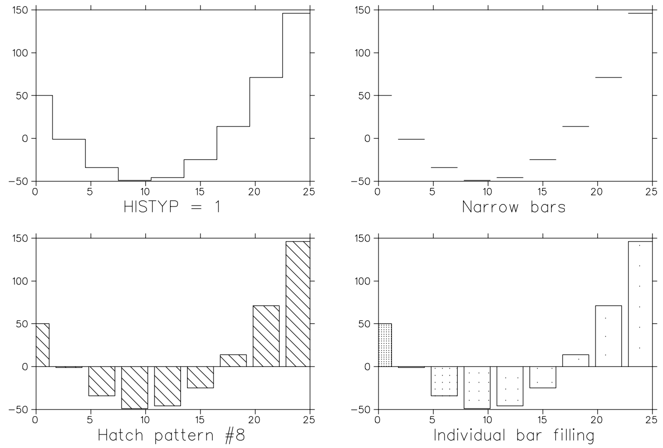


Figure 2.16: Histogram examples drawn with the GRAPH command

GRID

Syntax	`GRID x y z m`
Qualifiers	`\POLAR, \INTERPOLATE, \INDICES, \PATTERN, \SIZE, \XYOUT, \BOUNDS, \CHECKDUP`
Defaults	`\NOPOLAR, \INTERPOLATE, \NOINDICES, \NOSIZE, \NOXYOUT, \NOBOUNDS, \NOCHECKDUP`
Examples	`GRID X Y Z M` `GRID\POLAR\XYOUT R T Z M RO TO` `GRID\NOINTERP\XYOUT X Y Z M XOUT YOUT` `GRID\SIZE\BOUNDS 50 X Y Z M .1 .5 3 6`

The `GRID` command creates a regular matrix from scattered data points. The three vectors, `x`, `y` and `z`, are assumed to represent scattered points, where `z[i]` is the altitude corresponding to the coordinates `(x[i],y[i])`. Suppose $l = min(\texttt{len(x)},\texttt{len(y)},\texttt{len(z)})$. By default, a square matrix, `m`, is interpolated, with row and column dimensions equal to $5 \times \sqrt{l}$.

The coordinates of element `m[i,j]` of the output matrix will be `(xout[j],yout[i])`. The columns of `m` are of constant x, and the rows are of constant y.

Polar coordinates

The `\NOINTERPOLATE` qualifier cannot be used with the `\POLAR` qualifier.

If the `\POLAR` qualifier is used, `x` is assumed to contain the radial components and `y` is assumed to contain the angular components, in degrees. The output matrix, `m`, will be regular in polar coordinates, with the columns of constant radius and the rows of constant angle.

Duplicate points

By default, duplicate `(x,y)` locations are not checked for before the matrix is made. If you want duplicate points to be ignored, use the `\CHECKDUP` qualifier.

Interpolated grid

Syntax	`GRID x y z m`
Qualifiers	`\POLAR, \SIZE, \XYOUT, \BOUNDS, \CHECKDUP`
Defaults	`\NOPOLAR, \NOSIZE, \NOXYOUT, \NOBOUNDS, \NOCHECKDUP`

By default, the set of scattered data points is used to construct a Thiessen triangulation of the plane and a regular matrix, `m`, is interpolated.

Output matrix size

Syntax `GRID\SIZE s x y z m`

Suppose $l = min($`len(x)`,`len(y)`,`len(z)`$)$. By default, the interpolated matrix will be square, with row and column dimensions both equal to $5 \times \sqrt{l}$. If another size, `s`, is desired, you must use the `\SIZE` qualifier, and the row and column dimensions will be both equal to `s`.

Output vectors

Syntax `GRID\XYOUT x y z m xout yout`

If output vectors, `xout` and `yout`, are desired, you must use the `\XYOUT` qualifier. The coordinates of output matrix element `m[i,j]` will be `(xout[j],yout[i])`, where `xout` contains the x-coordinates of each column and `yout` contains the y-coordinates of each row.

Range of interpolation

Syntax `GRID\BOUNDS x y z m minx maxx miny maxy`
`GRID\BOUNDS\XYOUT x y z m xout yout minx maxx miny maxy`
Defaults `\NOBOUNDS`, interpolation range = range of `x` and `y`

By default, the range of the grid interpolation is the range of values of the vectors `x` and `y`. If the `\BOUNDS` qualifier is used, this range is specified by the final four numbers, `minx`, `maxx`, `miny`, and `maxy`.

Non-interpolated grid

Syntax `GRID\PATTERN x y z m`
Qualifiers `\XYOUT`, `\CHECKDUP`
Defaults `\NOXYOUT`, `\NOCHECKDUP`

Suppose the vectors x and y have length h, and suppose that for some $n1$ and $n2$, x and y have the following pattern:

$$\begin{array}{llll} x_1 & = x_2 & = \cdots = & x_{n2} \\ x_{n2+1} & = x_{n2+2} & = \cdots = & x_{n2+n2} \\ \vdots & & & \vdots \\ x_{(n1-1)n2+1} & = x_{(n1-1)n2+2} & = \cdots = & x_{n1\cdot n2} \end{array}$$

$$\begin{array}{llll} y_1 & = y_{n2+1} & = \cdots = & y_{(n1-1)n2+1} \\ y_2 & = y_{n2+2} & = \cdots = & y_{(n1-1)n2+2} \\ \vdots & & & \vdots \\ y_{n2} & = y_{n2+n2} & = \cdots = & y_{n1\cdot n2} \end{array}$$

where $h = n1 \cdot n2$. If the x and y vectors have this form, it is possible to construct a matrix, without interpolation, with $n2$ rows and $n1$ columns, that is, $m_{i,j} = z_k$ where $k = j + (i-1)n1$ for $i = 1, 2, \ldots, n2$ and for $j = 1, 2, \ldots, n1$.

Output vectors

Syntax `GRID\PATTERN\XYOUT x y z m xout yout`

If output vectors, `xout` and `yout`, are desired, you must use the `\XYOUT` qualifier. The coordinates of output matrix element `m[i,j]` will be `(xout[j],yout[i])`, where `xout` contains the x-coordinates of each column and `yout` contains the y-coordinates of each row. If the output matrix has $n1$ columns and $n2$ rows, then the length of `xout` will be $n1$ and the length of `yout` will be $n2$.

`xout` $= [x_1; x_{n2+1}; \cdots; x_{(n1-1)n2+1}$
`yout` $= [y_1; y_2; \cdots; y_{n2}$

Example

The vectors:

```
X = [ 1; 1; 1; 1; 2; 2; 2; 2; 3;  3;  3;  3 ]
Y = [ 1; 2; 3; 4; 1; 2; 3; 4; 1;  2;  3;  4 ]
Z = [ 1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12 ]
```

have the proper form, with $n2 = 4$ and $n1 = 3$.
If you entered the command `GRID\PATTERN\XYOUT X Y Z M XO YO`
the resultant variables would be:

$$\mathtt{M} = \begin{pmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{pmatrix} \mathtt{XO} = [1;2;3],\ \mathtt{YO} = [1;2;3;4]$$

Matrix from sparse data

Syntax	`GRID\INDICES x y z m`
Qualifiers	`\XYOUT, \CHECKDUP`
Defaults	`\NOXYOUT, \NOCHECKDUP`

The vectors x and y are assumed to contain index locations for the z data values.

Suppose that h is the minimum length of x, y, and z; and $nc = max(\texttt{x[i]})$, $nr = max(\texttt{y[i]})$ for $\texttt{i} = 1, \ldots, h$.

Then $\texttt{m[i,j]} = 0$ for $\texttt{i} = 1, \ldots, nr$; $\texttt{j} = 1, \ldots, nc$ except m[y[i],x[i]] = z[i] for $\texttt{i} = 1, \ldots, h$. m will have nr rows and nc columns.

Output vectors

Syntax `GRID\INDICES\XYOUT x y z m xout yout`

If output vectors, xout and yout, are desired, you must use the \XYOUT qualifier. The coordinates of output matrix element m[i,j] will be (xout[j],yout[i]), where xout contains the x-coordinates of each column and yout contains the y-coordinates of each row. If the output matrix has nc columns and nr rows, then xout = [1:nc] and yout = [1:nr].

Example

Suppose

```
X = [  1;  4;  1;  3;  5 ]
Y = [  2;  1;  6;  4;  6 ]
Z = [ 10; 15; 20; 25; 30 ]
```

after the command: `GRID\INDICES\XYOUT X Y Z M XO YO`

the resultant variables will be:

$$\texttt{M} = \begin{pmatrix} 0 & 0 & 0 & 15 & 0 \\ 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 25 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 20 & 0 & 0 & 0 & 30 \end{pmatrix} \quad \texttt{XO} = [1:5],\ \texttt{YO} = [1:6]$$

HARDCOPY

Syntax	`HARDCOPY keyword { queue }`
	`HARDCOPY keyword { file }`

The HARDCOPY command is used for obtaining graphics hardcopies for the currently active device type. The initial default graphics hardcopy type is an HP LaserJet bitmap at 150 dpi. Use the DEVICE command to choose a different graphics hardcopy device type.

Printing and saving

The first command parameter is a keyword which refers to the action of printing or saving the graphics. This depends on which graphics hardcopy device is currently active. Refer to Table 2.42 for a listing of the appropriate hardcopy print and save codes for the bitmap device types. Refer to Table 2.43 for a listing of the appropriate hardcopy print and save codes for the non-bitmap device types.

device keyword	*action code*	*parameter*	*description*	*default*
HPLASER	P	{ queue }	print on a queue	HP$LASER
	PC	{ queue }	print on a queue -compressed-	HP$LASER
	S	{ file }	save in a file	HPLASER.PLT
	A		auxiliary port output	
	T	{ file }	TeXoutput file file	HPTEX.PLT
	TC	{ file }	TeXoutput file -compressed-	HPTEX.PLT
	TJ	{ file }	TeXoutput file -justified-	HPTEX.PLT
	TJC	{ file }	TeXoutput file -compressed-justified-	HPTEX.PLT
INKJET	P	{ queue }	print on a queue	INK_JET
	PT	{ queue }	print on a queue -transparency-	INK_JET
	S	{ file }	save in a file	INKJET.PLT
	A		auxiliary port output	
HPTHINKJET	P	{ queue }	print on a queue	HP$THINKJET
	S	{ file }	save in a file	HPTHINK.PLT
	A		auxiliary port output	
PRINTRONIX	P	{ queue }	print on a queue	LNPTR
	S	{ file }	save in a file	PX.PLT
	A		auxiliary port output	

Table 2.42: HARDCOPY command print and save codes for bitmap devices

Note: Compressed format can speed up the printing of a drawing by as much as a factor of 5. Compressed format is recognized *only* by the LaserJet IIP, the LaserJet III, and later model

device keyword	*action code*	*parameter*	*description*	*default*
POSTSCRIPT	P	{ queue }	print on a queue	POST$SCRIPT
	S	{ file }	save in a file	POSTSCRIPT.PLT
	A		auxiliary port output	
HPPLOTTER	P	{ queue }	print on a queue	HPLTR
	S	{ file }	save in a file	HPP.PLT
	A		auxiliary port output	
GKS	S	{ file }	save file	GKS.PLT
HOUSTON	P	{ queue }	print on a queue	HOUSTON
	S	{ file }	save in a file	HOUSTON.PLT
	A		auxiliary port output	
IMAGEN	P	{ queue }	print on a queue	IMAGEN
	S	{ file }	save in a file	IMAGEN.PLT
	A		auxiliary port output	
LA100	P	{ queue }	print on a queue	LA100
	S	{ file }	save in a file	LA100.PLT
	A		auxiliary port output	
LN03+	P	{ queue }	print on a queue	LN03
	S	{ file }	save in a file	LN03.PLT
	A		auxiliary port output	
ROLAND	P	{ queue }	print on a queue	RDGL
	S	{ file }	save in a file	RDGL.PLT
	A		auxiliary port output	

Table 2.43: HARDCOPY command print and save codes for non-bitmap devices

printers. Do *NOT* use compressed format output on other than these devices.

Examples

Suppose that the Hewlett-Packard pen plotter output has been enabled, with the command DEVICE HPPLOTTER. To print the graphics directly on queue QUENAME, enter:

```
HARDCOPY P QUENAME
```

Suppose that the HP LaserJet bitmap has been enabled with the DEVICE HPLASERJET command. To save the graphics in a file, FILE.PLT, for inclusion in a TEXor LATEXdocument in justified format, enter:

```
HARDCOPY TJ FILE.PLT
```

Suppose a PostScript printer has been enabled with the DEVICE POSTSCRIPT command. To save the graphics in a PostScript file, FILE.PSC, to be printed later, enter:

```
HARDCOPY S FILE.PSC
```

HELP

Syntax	HELP { string ... }
	HELP\LIBRARY libname { string ... }
Qualifiers	\PAGE, \LIBRARY
Defaults	\NOPAGE, \NOLIBRARY

For VMS users:

The HELP command invokes the on-line help facility. To get information about a specific topic, include it as the string with the HELP command. To leave help quickly, type control-z.

Paging the output

By default, output to the screen is not displayed by pages, but output to the screen continues until the information display ends. If the \PAGE qualifier is used, output to the screen is paged, that is, the output stops after each screen full of information is displayed.

User defined library

The HELP\LIBRARY command allows the user to specify a help library other than the default PHYSICA help library. The full filename specification is required, that is:

disk:[directory]libname.hlb

The \LIBRARY qualifier can be used with the \PAGE qualifier.

For UNIX users:

The HELP command invokes an on-line help facility that mimics the built-in VMS help facility. To browse the help information, enter just the HELP command. To get information about a specific topic, include it as the string with the HELP command, but then no subtopics will be displayed. To leave help quickly, type control-d.

INPUT

Syntax	INPUT x1 { x2 ... x8 } INPUT\MATRIX m nr nc
Qualifier	\MATRIX
Default	create vectors

The INPUT command is used to interactively enter data, from the terminal keyboard, into vector(s) or into a matrix. The default is to create vectors.

If the INPUT command is used in a script file, input will still be expected from the terminal keyboard.

Vectors

Syntax INPUT x1 { x2 ... x8 }

Enter one set of numbers per line, that is, enter x1[i] ... x8[i] all on one line. The maximum number of vectors that can be input with one command is 8.

VMS: Input is terminated by typing control-z.

UNIX: Input is terminated by typing control-d.

Making corrections

If you want to change a previous entry, say the n^{th} entry, enter In before the new entries. All entries must be present. For example, suppose you are creating three vectors, X, Y, and Z, and when you are entering the fourth set of numbers, you realize you have made a mistake in the second entry. The following example shows how you could correct the mistake.


```
PHYSICA: INPUT X Y Z
Enter 3 numbers, ( control-Z ends )
(  1) >> 10 .1 1000
(  2) >> 20 2 2000
(  3) >> 30 .3 3000
(  4) >> I2 20 .2 2000
(  4) >> 40 .4 4000
(  5) >> 50 .5 5000 <control-z>
PHYSICA:
```

and then you will have the following vectors:

```
X = [ 10; 20; 30; 40; 50 ]
Y = [ 0.1; 0.2; 0.3; 0.4; 0.5 ]
Z = [ 1000; 2000; 3000; 4000; 5000 ]
```

Matrix

Syntax INPUT\MATRIX m nr nc

The INPUT\MATRIX command is used to interactively enter data, from the terminal keyboard, into a matrix. A new matrix will be created.

The number of rows, nr, and the number of columns, nc, must be entered. Enter one row of the matrix, that is, nc numbers, per line. Only nr lines will be requested.

Making corrections

If you want to change a previous row, say the n^{th} row, enter In before the new entries. All entries must be present. For example, suppose you are creating a matrix, M, with 5 columns and 3 rows. When you are entering the third set of numbers, you realize you have made a mistake in the second set. The following example shows how you could correct the mistake.

```
PHYSICA: INPUT\MATRIX M 3 5
Enter 5 numbers
(row  1) >> 1 2 3 4 5
(row  2) >> 6 7 8 9 10
(row  3) >> I2 6 7 -8 9 10
(row  3) >> 11 12 13 14 15
PHYSICA:
```

and you will have the following matrix:

$$M = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & -8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix}$$

INQUIRE

Syntax	`INQUIRE 'prompt string' v1 { v2 ... }`
Defaults	if `vI` does not exist, it is assumed to be a scalar
Examples	`INQUIRE 'Enter a value >>' A` `INQUIRE 'Enter YES or NO >>' TXT`

The `INQUIRE` command is intended for use in script files. The prompt string is written to the monitor screen and you are expected to enter the correct number and type of values, corresponding to the variable names following the prompt string. You can inquire for scalars, vectors, matrices, or string variables.

If *just* a carriage return is typed in response to an `INQUIRE` prompt, the variable(s) that are being requested will keep their current value(s). This allows you to have default values for inquired variables.

If variable `vI` does not exist, it is assumed to be a scalar. If `vI` is a scalar, the user is expected to enter a literal constant or a scalar. For example,

```
S=3  ! default value is 3
INQUIRE 'Enter scalar >> ' S
```

If `vI` is a vector, the user is expected to enter a set of values or a vector. For example,

```
V=[1:10]  ! default values
INQUIRE 'Enter vector >> ' V
```

If `vI` is a matrix, the user is expected to enter a matrix. For example,

```
MATRIX M 5 5
INQUIRE 'Enter matrix >> ' M
```

If `vI` is a string variable, the user is expected to enter a literal string or a string variable. For example,

```
A='yes'  ! default is yes
```



```
INQUIRE 'Enter Yes or No >> ' A
```

Examples

Suppose you want a script that asks the user a question requiring a yes or no response, with the script branching depending on the response. Following is an example of this procedure. Note that the response is converted to uppercase, since it must be in uppercase for the checks to find a YES or NO.

```
DUM='yes'                            ! make a dummy string variable
ASK:                                 ! a label
INQUIRE 'Enter YES or NO >>' DUM     ! get a response from the keyboard
IF EQS(UCASE(DUM),'YES') THEN               ! check if YES
  ...
  ...
  GOTO NEXT                          ! it was YES, so continue
ENDIF
IF EQS(UCASE(DUM),'NO') THEN                ! check if NO
  ...
  ...
  GOTO NEXT                          ! it was NO, so continue
ENDIF
DISPLAY 'Invalid response: expecting YES or NO'
GOTO ASK:                            ! go back and ask again
NEXT:                                ! continue with the script
 ...
 ...
```

In the following example, the first command defines `X` to be a vector of length 1. The second command defines `S` to be a scalar. The third command creates a scalar `N` and assigns it the value 5.

```
VECTOR X 1
SCALAR S
N = 5
INQUIRE 'Enter a vector with '//RCHAR(N)//' entries and a scalar >>' X S
```

The `INQUIRE` command will write out the prompt and wait for you to type in a vector set and a single value. For example:

```
Enter a vector with 5 entries and a scalar >> [0;1;3.5;-4;100] 10.3
```

JOURNAL

Syntax	`JOURNAL filename`
Qualifiers	`\APPEND, \MACRO`
Defaults	`\NOAPPEND`, `\NOMACRO`, initial journal file = `PHYSICA.JOURNAL`

A journal file is a record of all user input to PHYSICA, as well as all non-graphics output, such as error messages, from PHYSICA. The initial state is to have the journaling of interactive input and output on, journaling of script file input and output off, and the initial journal file name is `PHYSICA.JOURNAL`.

Journaling can be disabled by entering the `DISABLE JOURNAL` command. This closes the file. Enter `ENABLE JOURNAL` to open the last journal file that was open, and to append subsequent journal entries to this file. To also enable the writing of script file input and output to the journal file, use the `ENABLE JOURNAL\MACRO` command.

To just disable journaling of script file input and output, use the `DISABLE JOURNAL\MACRO` command.

To open a new journal file, use the `JOURNAL` command. To open with append, use the `\APPEND` qualifier.

To find out whether a journal file is currently open, and if so, to display the name of the current journal file, enter the `STATUS` command.

Environment variables in file names

For UNIX users, it is now possible to use an environment variable in a file name, if the environment variable is preceeded by a `$`. For example,

```
setenv FILE myjournal
physica
journal $FILE
```

The environment variable can be just the first part of the filename, for example,

```
setenv FILE my
physica
journal $FILEjournal
```

KEYWORD

Syntax	`KEYWORD`

The `KEYWORD` command enters an interactive mode, where you type a keyword and the online help locations of that keyword are displayed. The help locations are separated by blanks, while vertical bars, `|`, separate the levels within each location. For example, typing the keyword `shell` will display the help locations `DISABLE|SHELL ENABLE|SHELL`. You could then find information on `shell` by typing `HELP DISABLE SHELL` or `HELP ENABLE SHELL`.

The wildcard is `*`. An initial wildcard and/or a final wildcard are allowed. For example, `*inc*` displays `inch inches nlxinc nlyinc nsxinc nsyinc` which are valid keywords; while `inc*` displays `inch inches`.

Typing a `TAB`, or `control-I`, is similar to using a final wildcard, in that all matching keywords are displayed, unless there is a unique keyword, and then the keyword is completed for you.

LABEL

Syntax	`LABEL { 'textstring' }`
Qualifiers	`\XAXIS, \YAXIS`
Defaults	no x-axis label, no y-axis label
Examples	`LABEL\XAXIS TXTVAR[5][1:10]`
	`LABEL\XAXIS`
	`LABEL\YAXIS 'This'//CHAR(39)//'s a label with a single quote'`
	`LABEL\YAXIS '<alpha,beta,^>10'`

The `LABEL` command sets the automatic x- or y-axis text label. Use the `\XAXIS` qualifier to set the x-axis label and use the `\YAXIS` qualifier to set the y-axis label.

The default font can be changed with the `SET FONT` command. To change the size of the x-axis text label use the `SET` command, with either `XLABSZ` or `%XLABSZ`. To change the size of the y-axis text label use the `SET` command, with either `YLABSZ` or `%YLABSZ`.

The x-axis automatic text label is drawn, centred, below the x-axis. The x-axis label is drawn only if and when the x-axis is drawn. The y-axis automatic text label will be drawn, centred, to the left of the y-axis. The y-axis label is drawn only if and when the y-axis is drawn. The commands that draw axes are: `GRAPH`, `CONTOUR`, `DENSITY`, and `REPLOT`.

Turning off the labels

Entering the `LABEL\XAXIS` command without a parameter indicates that no automatic x-axis text label is to be drawn.

Entering the `LABEL\YAXIS` command without a parameter indicates that no automatic y-axis text label is to be drawn.

Example

The following script demonstrates automatic axis labeling, and produced Figure 2.17 on page 137.

```
LABEL\XAXIS '<alpha,beta,gamma> X-AXIS TEST'
LABEL\YAXIS 'Y-AXIS TEST <sigma,^>2<_> > 5'
GRAPH\AXESONLY
```

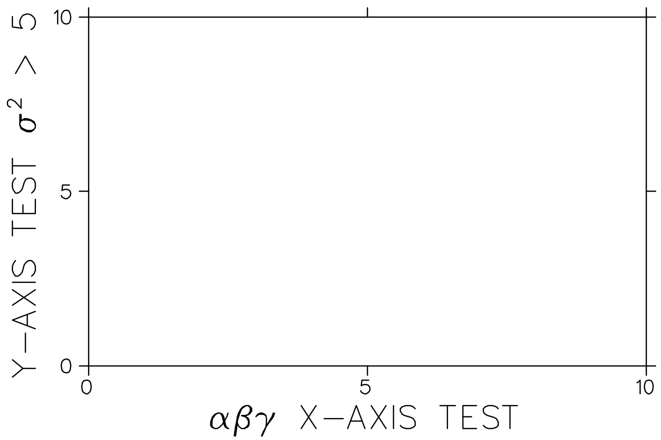


Figure 2.17: Automatic axis labels using the LABEL command

LEGEND

Syntax	`LEGEND ON	OFF` `LEGEND FRAME ON	OFF	{ xlo ylo xhi yhi }` `LEGEND TRANSPARENCY ON	OFF` `LEGEND AUTOHEIGHT ON	OFF` `LEGEND NSYMBOLS n` `LEGEND TITLE { 'title string' }` `LEGEND STATUS`
Defaults	legend off, frame on, transparency on, autoheight on, \PERCENT, (xlo,ylo) = (55,22.5), (xhi,yhi) = (78,45), number of plotting symbols = 1, no title

If LEGEND is ON, a legend entry is drawn into a legend frame box when the GRAPH command is entered. A legend entry consists of a short line segment with plotting symbols, and a string.

The string portion of a legend entry

If LEGEND is ON, the string portion of a legend entry is expected as the first parameter of the GRAPH command. For example:

```
GRAPH 'legend entry' X Y
```

If LEGEND is OFF, a string entered as a first parameter with the GRAPH command is ignored.

The string may contain formatting commands. See the PLOTTEXT command, page 165, for information on text formatting.

If LEGEND is ON then the data variables are not necessary, that is, you can enter: GRAPH\NOAXES 'Legend entry' and an entry is made to the legend but no curves are plotted.

Legend entry text height

Syntax	`LEGEND AUTOHEIGHT ON` `LEGEND AUTOHEIGHT OFF`
Default	ON

By default, the height of the string portion of the legend entry will be determined so that the string fits ”lengthwise” in the legend frame box. If you want to specify the height of the string portion of the legend entry, enter the LEGEND AUTOHEIGHT OFF command. The default is AUTOHEIGHT ON. When AUTOHEIGHT is OFF, the text height will be the current value of TXTHIT, as specified with the SET command. You can also include the text height as a text formatting command, <Hnn.n> or <Hnn.n%>, within the string.

Do not include a text height formatting command when AUTOHEIGHT is ON. This will cause infinite looping, as the program attempts to adjust the text height to fit within the frame.

The line segment portion of a legend entry

The line segment portion of a legend entry is drawn with the same line type and plotting symbol as the corresponding data curve.

Plotting symbols

Syntax	`LEGEND NYSMBOLS n`
Default	number of symbols = 1

The number of plotting symbols drawn on the line segment can be specified by using the

NSYMBOLS keyword. The default is n = 1, to plot one symbol. If one symbol is drawn, it will be in the middle of the line segment. If more one symbol is drawn, they will be equally spaced along the line segment, with one symbol at each end. If no symbols are drawn, the line segment will still be drawn. If no symbols are plotted on the data curve, no symbols will appear in the legend entry. Only the first n plotting symbols used for the data curve will be drawn in the legend entry.

Plotting symbols for the data curve are chosen with the SET PCHAR command.

The frame box

Syntax LEGEND FRAME { units } { xlo ylo xhi yhi }
LEGEND FRAME ON
LEGEND FRAME OFF

Defaults ON, \PERCENT, (xlo,ylo) = (55,22.5), (xhi,yhi) = (78,45)

Use the FRAME keyword to control the legend frame box. Legend entries are drawn starting from the top of the frame box. No clipping is done at the edges of the frame box, so entries may appear outside the frame box.

If LEGEND FRAME OFF is entered, the outline of the frame box will not be drawn. The default is ON, that is, to draw the legend frame box outline. If neither OFF nor ON is entered, it is assumed that you are entering the corner coordinates of the frame box.

Coordinates

The coordinates of the lower left corner of the legend frame box are (xlo,ylo), and the coordinates of the upper right hand corner are (xhi,yhi). The default lower left corner is (55,22.5), and the default upper right corner is (78,45), expressed as percentages of the current window.

If any of xlo, ylo, xhi, and yhi are not entered, the graphics cursor will be used to choose the missing coordinates.

Units

The frame box coordinates may be expressed in three types of units, which are chosen by command qualifier. The default is \PERCENT. See Table 2.44 for a listing of the qualifiers and their interpretations.

For example, if the \PERCENT qualifier is used, then a location of $(50, 50)$ represents the centre of the current window. If the \WORLD qualifier is used, the coordinates are in units of the

qualifier	*interpretation of the coordinates*
\PERCENT	percentages of the current window, as chosen with the WINDOW command.
\GRAPH	graph units, that is, the units defined by the minimum and maximum values for the last graph drawn. If no graph has been drawn yet, the defaults are $-1 \leq x \leq 1$ and $-1 \leq y \leq 1$
\WORLD	centimeters or inches, as chosen with the SET UNITS command

Table 2.44: Types of units recognized by the LEGEND command

world coordinate system, the plotting units. The default world coordinate system units are centimeters. See the DEVICE command for tables showing the dependence of plotting units on the graphics hardcopy output device.

Transparency

Syntax LEGEND TRANSPARENCY ON
LEGEND TRANSPARENCY OFF

Default ON

When the legend entries are drawn originally, the heights of the strings will likely all be different if AUTOHEIGHT is ON, and whatever is behind the frame box will be visible in the frame box. That is, the legend will be transparent. If the REPLOT command is entered, all the strings will be redrawn at the same height, using the minimum height of all the legend entries, and, if TRANSPARENCY is OFF, the background of the frame box will be erased. The default is TRANSPARENCY ON.

Note: This background erasing only applies to the monitor screen and to bitmap graphics hardcopy output, such as HP LaserJet output.

Remember, the legend frame box background will only be erased when a REPLOT command is entered.

Moving or resizing the legend frame box

The frame box can be moved and/or resized. First enter the LEGEND FRAME command to specify a new frame box, and then enter the REPLOT command.

The legend title

Syntax `LEGEND TITLE { ‘title string’ }`

Default no title

The default is to have no legend title. If there is a legend title, it is drawn just above the top of the frame box. The height of the legend title will be the current value of `TXTHIT`, as specified with the `SET` command. Text formatting commands may be included in the title string.

The legend title can be turned off by entering the `LEGEND TITLE` command without a title string.

Status

Syntax `LEGEND STATUS`

Use the `STATUS` keyword to display, on the monitor screen, the current status of the legend attributes.

Example

The following script produces figure 2.18 on page 143.


```
! first define the elements of a array string variable
T[1]=‘sin(<FITALIC.3>x<FTSAN>)<FITALIC.3>e<fITALIC.3,^>x<FTSAN>/5’
T[2]=‘<FITALIC.3>x<FTSAN>/2-5’
T[3]=‘(<FITALIC.3>x<FTSAN>/3.5)<^>2<_>+3<FITALIC.3>x<FTSAN>/3.5’
T[4]=‘cos(<FITALIC.3>x<FTSAN>)<FITALIC.3>e<FTSAN,^>-<FITALIC.3>x<FTSAN>/9’
X = [0:4*PI:.5]                ! generate some data
LEGEND ON                      ! turn legend on
LEGEND FRAME 20 50 50 80       ! define the legend frame
LEGEND FRAME ON                ! draw the frame box outline
LEGEND TITLE ‘<FROMAN.SERIF>Test legend’
SET LINTYP 1                   ! draw the first data curve
SET PCHAR 1                    ! plotting symbol
LEGEND NSYMBOLS 1              ! one plotting symbol in legend entry
GRAPH T[1] X SIN(X)*EXP(X/5)   ! T[1] is the legend string entry
SET LINTYP 10                  ! draw the second data curve
SET PCHAR 2                    ! plotting symbol
LEGEND NSYMBOLS 2              ! 2 symbols in legend entry
CURVE T[2] X X/2-5             ! T[2] is the legend string entry
SET LINTYP 5                   ! draw the third data curve
SET PCHAR 3                    ! plotting symbol
LEGEND NSYMBOLS 3              ! 3 symbols in legend entry
CURVE T[3] X (X/3.5)^2+3*X/3.5 ! T[3] is the legend string entry
SET LINTYP 7                   ! draw the fourth data curve
SET PCHAR 13                   ! plotting symbol
LEGEND NSYMBOLS 4              ! 4 symbols in legend entry
CURVE T[4] X COS(X)*EXP(-X/9)  ! T[4] is the legend string entry
REPLOT                         ! replot data on common scale
```

LINE

Syntax `LINE`
`LINE\XYOUT x y { pen_code { line_type { colour { thickness }}}}`
`LINE x y { pen_code { line_type { colour { thickness }}}}`

Qualifiers `\GRAPH`, `\PERCENT`, `\WORLD`, `\XYOUT`

Defaults `\PERCENT`, interactive drawing

This command draws line segments. If no parameters are entered with the `LINE` command, the graphics crosshair is used to draw interactively. If the `\XYOUT` qualifier is used, the graphics crosshair is used to draw the line segments, and the parameters are assumed to be output vectors in which to store the line segment coordinates and other attributes. If the `\XYOUT` qualifier is not used, and vectors are entered as parameters, then the input vectors

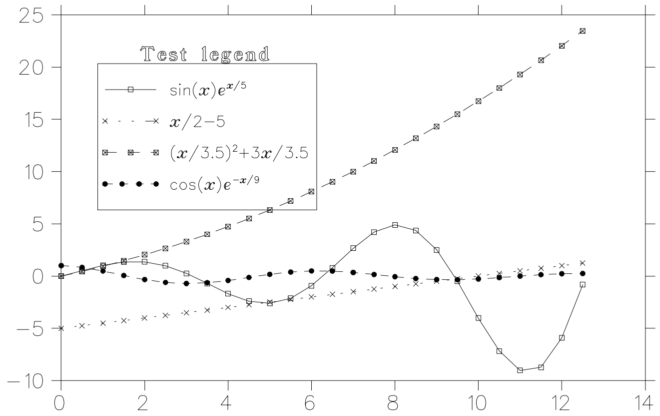


Figure 2.18: An example illustrating the graph LEGEND

are assumed to contain coordinates and other attributes for the line segments, and the graphics crosshair is not used.

Plotting units

If the `\PERCENT` qualifier is used, then the values of the coordinates will be interpreted as being in percentages of the current window, for example, $(50, 50)$ represents the centre of the current window. This is the default if no units qualifier is used.

If the `\GRAPH` qualifier is used, then the values of the coordinates will be interpreted as being in graph units, that is, the units defined by the minimum and maximum values for a graph. These units are taken from the last graph drawn, or, if no graph has been drawn yet, they are the default values: $-1 <= x <= 1$ and $-1 <= y <= 1$.

If the `\WORLD` qualifier is used, then the values of the coordinates will be interpreted as being in centimeters or inches, depending on the units type chosen with the `SET UNITS` command. The default world units are centimeters.

X Windows

When running under X Windows, mouse button two toggles the continuous display of the graphics cursor location. The PHYSICA keyword `CUNITS` is the units type for these numbers. If `CUNITS` = `WORLD`, the numbers depend on the current units type, either `CM` or `IN`, as chosen with `SET UNITS`. If `CUNITS` = `GRAPH`, the numbers displayed depend on the current graph axis

scales. If `CUNITS` = `PERCENT`, the numbers depend on the current window.

Non-interactive drawing

Syntax	`LINE x y { pen_code { line_type { colour { thickness }}}}`
Defaults	`pen_code` = $[3; 2; 2; ...]$, `line_type` = $[1; 1; 1; ...]$, `colour` = $[1; 1; 1; ...]$, `thickness` = $[1; 1; 1; ...]$

Vectors `x` and `y` should contain the x and y coordinates for the line segments' end points. Optional vector `pen_code` should contain the pen codes, either 2 (connect) or 3 (disconnect) for each point. Optional vector `line_type` should contain the line type codes (between 1 and 10) for the line segments. The line types are described in the `SET LINE` command section. Optional vector colour should contain the colour codes (between 1 and 8). Optional vector `thickness` should contain the the line thickness for bitmap and PostScript hardcopy output.

key	action
`Q`	quit the `LINE` command
`M`	display the command menu
`/`	clear the alpha-numeric terminal screen This has no affect on the graphics
`U`	start a line segment A mark will be drawn at the current location, but is not entered into the plot file, or into an open EDGR file, it is just there for reference
`D`	end a line segment A line segment is drawn from the last location that was chosen with the `U` or `D` key to the current location. This line segment will be entered into the plot file, and into an open EDGR file
`T`	try a line segment A line segment is drawn from the last location that was chosen with the `U` or `D` key to the current location. This line segment will *not* be entered into the plot file, or into an open EDGR file. If the line segment is acceptable, do not move the crosshair, and type the `D` key. If the line segment is not acceptable, simply move to another location and try again; this line segment will then be erased.
`L`	change the line type The current line type code is displayed and a new code can be entered. No carriage return is necessary after the new code is entered. The line type should be between 1 and 10, to choose number 10, enter an `A`. The line types are described in the `SET LINE` command section. The default line type is 1.
`C`	change the colour The current code is displayed and a new code can be entered. No carriage return is necessary after the new code is entered. 1=white, 2=red, 3=green, 4=blue, 5=yellow, 6=cyan, 7=magenta, 8=white
`N`	change the units The current units code is displayed and a new code can be entered. No carriage return is necessary after the new code is entered. 1 - percentages of the world coordinate system 2 - world coordinate system units (centimeters or inches) 3 - graph units 4 - percentages of the current window
`X`	toggle the display of the x and y coordinates at the bottom of the terminal screen. This is for user reference only. By default, the x and y coordinates are not displayed.

Table 2.45: The `LINE` command interactive menu

LIST

Syntax	`LIST x1 { x2 ... x5 } { n1 { n2 { n3 }}}` `LIST\MATRIX matrix` `LIST\MATRIX\FORMAT matrix (format)`
Qualifiers	`\PAGE, \MATRIX, \FORMAT, \COUNTER`
Defaults	vector(s) expected, `n1` = 1, `n3` = 1, not paged, `format` = 1PD13.4, `\COUNTER`
Examples	`LIST X Y Z` `LIST X Y Z 1 10 2` `LIST\PAGE X[10:#] Y Z` `LIST\MATRIX\FORMAT M[2:11,5:20:2] (6(F10.3,2X))`

By default, the `LIST` command displays, on the monitor screen, a listing of the specified vectors, `xI[n1:n2:n3]`, with a counter displayed on the left under the heading `#`. `n1` defaults to 1 and `n3` defaults to 1 if not entered. If `n2` is not entered, all of the elements of each vector are listed. If the counter is not desired, use the `\-COUNTER` qualifier.

Paging the output

The `\PAGE` qualifier only applies to listings of vectors.

If the `\PAGE` qualifier is used, the listing will be paged, that is, after every twenty lines have been displayed, you will be asked if you want to continue or quit. Type the `Q` key to quit. Type any other key to continue the listing. No carriage return is necessary.

Listing a matrix

Syntax	`LIST\MATRIX matrix` `LIST\MATRIX\FORMAT matrix (format)`
Qualifiers	`\MATRIX, \FORMAT`
Defaults	`format` = 1PD13.4

The `LIST\MATRIX` command displays a matrix on the monitor screen. The format defaults to `1PD13.4`. Use the `\FORMAT` qualifier to indicate that a user specified format has been entered. The format must be enclosed in parentheses, `(` and `)`. Any standard FORTRAN format is valid, but only REAL variables can be displayed. Do not use INTEGER, LOGICAL, or CHARACTER formats.

LOAD

Syntax `LOAD filename{,libraries}`

Examples `LOAD MYFILE`
`LOAD MYFILE,LIB1/LIB,LIB2/LIB`

The `LOAD` command is only available under VAX/VMS.

The `LOAD` command dynamically loads and links an object module of a user written subroutine or a user written `REAL*8` function. The `filename` is the compiled object module.

If a function is loaded, it can only be used in expressions, via the name `USERN`. If a subroutine is loaded, it can only be used with the `CALL` command, with no specified name.

If your subroutine or function requires other routines, either object modules files or object module libraries, just include the other file names and/or library names in the `LOAD` command. For example:

```
LOAD somename,OTHER1.OBJ,DISK:[DIR]LIBRARY/LIB
```

Restrictions

- The complete linked module has a size limit of 256,000 bytes (500 blocks).
- Only one object module can be loaded at any time. A subsequent `LOAD` command will simply replace the old module with the new.

Arguments

A user written function can have up to 20 `REAL*8` arguments, all of which must be scalars.

A user written subroutine can have up to 15 arguments, which can be a mixture of contstants, scalars, vectors, matrices, quote strings, and string variables. The numeric arguments must be `REAL*8`.

The user *must* call the loaded subroutine or function with the correct number and type of arguments, else an addressing exception will result, and the program may be corrupted.

Subroutines

The name of a subroutine to be loaded dynamically via the `LOAD` command is irrelevant and can be anything the user desires. A user written subroutine *must* have the following form:


```
SUBROUTINE subname(IATYPE,ICODE,IUPDATE,IER,arg1,arg2,...)
INTEGER*4 IATYPE(15),ICODE(3,15),IUPDATE(15),IER
```

Other than the required arguments, `IATYPE`, `ICODE`, `IUPDATE`, and `IER`, there may be from 1 to 15 arguments in the subroutine argument list. The user is responsible for insuring that the correct number and type of arguments are used when actually employed with the `CALL` command. The parameters used in the `CALL` command, `argI`, which are passed as arguments to the subroutine, may be constants, scalars, vectors, matrices, quote strings, or string variables. The number of arguments and the type of argument *must* agree with the actual subroutine.

All of the numeric aguments, except for the required integer arguments `IATYPE`, `ICODE`, `IUPDATE`, and `IER`, must be `REAL*8`. A string argument is passed as a `LOGICAL*1` array.

Note: The integer arguments `IATYPE`, `ICODE`, `IUPDATE`, and `IER` should *not* be mentioned as parameters with the `CALL` command.

See the file: `PHYSICA$DIR:PHYSICA_USER_FUNCTIONS.FOR` for some subroutine examples.

IATYPE

`IATYPE` is an `INTEGER*4` array, length 15, that indicates the type of each of the subroutine arguments `argI`. See Table 2.4 on page 15.

ICODE

`ICODE` is an `INTEGER*4` array, dimensioned 3 by 15, that indicates the dimension of each of the subroutine arguments `argI`. Never extend variables beyond their original size as passed to the subroutine. If a variable is shortened inside the subroutine, the subroutine must update the new dimensions in the `ICODE` array, so that PHYSICA can reduce the variable dimensions appropriately. See Table 2.5 on page 16.

The `ICODE` array will be filled by PHYSICA with the current dimensions of the arguments, so the user written subroutine can check and, if necessary, update the dimensions of any of the subroutine arguments.

Never extend vectors, matrices, or string variables beyond their original sizes as passed to the user written subroutine. If a variable's size is shortened inside the subroutine, then the subroutine must update the `ICODE` array so that these variable dimensions can be reduced internally by PHYSICA upon return from the subroutine.

IUPDATE

`IUPDATE` is an `INTEGER*4` array, length 15, that the user routine sets to indicate to PHYSICA whether one of the `argI` arguments has been modified inside that subroutine.

The default value for `IUPDATE(i)` is 0. Set `IUPDATE(i)` to 1 to indicate that the $\mathtt{i}^{th}$ argument, `argI`, has been modified. Never extend variables beyond their original size as passed to the subroutine. If a variable is shortened inside the subroutine, the subroutine must update the new dimensions in the `ICODE` array, so that PHYSICA can reduce the variable dimensions appropriately.

IER

`IER` is an `INTEGER*4` variable that defaults to the value 0. Your routine can set `IER` to indicate to PHYSICA that an error has occured in the routine. Arithmetic errors, such as division by zero, over/underflow, will be asynchronously trapped. If other error tests are to be done inside the subroutine, the user flags the error by setting `IER = -1` before the `RETURN`. If the `CALL` command was executed from within a script, this error flag causes PHYSICA to abort that script and control is passed back to the keyboard.

Numeric arguments

All the numeric arguments of your subroutine, except for the integer arguments `IATYPE`, `ICODE`, `IUPDATE`, and `IER`, must be `REAL*8`. A string argument is passed as a `LOGICAL*1` array. Dimension numeric array arguments with length 1, for example:

```
REAL*8 X(1), Y(1), Z(1)
```

String arguments

All the string arguments of your subroutine must be `LOGICAL*1`, and should be dimensioned 1, for example:

```
LOGICAL*1 LFILE(1)
```

You can convert this to a string, say, `CHARACTER*80 CFILE`, using the following method:

```
      LENF = ICODE(1,i)
      DO I = 1, LENF
        CFILE(I:I) = CHAR(LFILE(I))
      END DO
```


where `LFILE` is the $\mathtt{i}^{th}$ argument.

Accessing matrix data

If a matrix is passed as an argument to a user written subroutine, the elements of the matrix can only be accessed using a calculated index. To access element `m[i,j]` of the matrix `m`, use `m[i+(j-1)*nrows]` for `i=1, ..., nrows` and `j=1, ..., ncols`.

Subroutine example

Consider the following subroutine, in file `LOAD1.FOR`, for calculating the cumulative product of the elements of a vector. Note the use of the mandatory integer arrays for checking input parameter types and sizes, for indicating which variables have changed, and for flagging errors.

```
      SUBROUTINE LOAD1(IATYPE,ICODE,IUPDATE,IER,X,Y)
      INTEGER*4 IATYPE(15),ICODE(3,15),IUPDATE(15),IER
      REAL*8    X(1), Y(1)

C   This subroutine requires two vector arguments, with the length of Y
C   greater than or equal to the length of X, and calculates the cumulative
C   product of X and stores the results in Y.

C   First check for input errors

      IF( IATYPE(3) .NE. -99 )THEN
        WRITE(*,*)' ERROR: too many arguments for loaded subroutine'
        IER = -1
        RETURN
      END IF
      IF( IATYPE(2) .EQ. -99 )THEN
        WRITE(*,*)' ERROR: not enough arguments for loaded subroutine'
        IER = -1
        RETURN
      END IF
      IF( IATYPE(1) .EQ. -99 )THEN
        WRITE(*,*)' ERROR: not enough arguments for loaded subroutine'
        IER = -1
        RETURN
      END IF
      IF( IATYPE(2) .NE. 1 )THEN
        WRITE(*,*)' ERROR: second argument is not a vector'
        IER = -1
        RETURN
      END IF
      IF( IATYPE(1) .NE. 1 )THEN
        WRITE(*,*)' ERROR: first argument is not a vector'
        IER = -1
        RETURN
      END IF
      IF( ICODE(1,2) .LT. ICODE(1,1) )THEN
        WRITE(*,*)' ERROR: second vector length < first vector length'
        IER = -1
        RETURN
      END IF
      IUPDATE(2) = 1    ! indicates the second argument is changed
      Y(1) = X(1)
      DO I = 2, ICODE(1,1)
        Y(I) = Y(I-1)*X(I)
      END DO
      RETURN
      END
```


To compile this source code, from within PHYSICA, creating an object module, load it, and then to make use of it, enter:

```
$FORTRAN LOAD1
LOAD LOAD1
X=[1:20]
VECTOR XPROD 20
CALL X XPROD
SCALAR\DUMMY J
LIST X XPROD RPROD(X[J],J,1:20)   ! RPROD function is the same as LOAD1
```

Functions

A user written *function* should have the following form:

```
REAL*8 FUNCTION funcname(ARG1,ARG2,...,ARGM)
REAL*8 ARG2,ARG2,...,ARGM
```

The name of the function, `funcname`, is irrelevant, since, when the function is used in an expression, the name that is used *must* be `USERN`.

Note: There is no `ICODE` array for a function.

There may be from 1 to 20 arguments. All arguments *must* be `REAL*8` scalars.

All arguments of the function are scalars. When the function is used in an expression, you can use vectors or matrices as arguments since the expression evaluator will process the function one element at a time.

Functions are *not* used with the `CALL` command. A user written function can be used wherever a function can be used in an expression.

Arithmetic errors, such as division by zero, overflow, or underflow, will be asynchronously trapped.

Note: There are also eight user written functions which are loaded at run time via a shareable image. These eight functions must be called `USER1`, `USER2`, ..., `USER8`.

Function example

Suppose you want to use a function, ANYNAME, which has three arguments, x_1, x_2, and x_3, and returns the sum of x_1 and x_2 when x_3 is within $\frac{1}{2}$ of $\sqrt{x_1^2 + x_2^2}$, else it returns 0.

```
REAL*8 FUNCTION ANYNAME(X1,X2,X3)
IMPLICIT REAL*8 (A-H,O-Z)
ANYNAME = 0.0D0
IF( ABS( SQRT(X1**2+X2**2)-X3 ) .LT. 0.5D0 )ANYNAME = X1+X2
RETURN
END
```

To compile this source code, from within PHYSICA, creating an object module, load it, and then to make use of it, enter:

```
$FORTRAN ANYNAME
LOAD ANYNAME
=USERN(A,B,C)
```

A simple way to test the above function:

```
X=RAN([1:100])
Y=RAN([1:100])
Z=RAN([1:100])
U1=USERN(X,Y,Z)
U2=(X+Y)*(ABS(SQRT(X^2+Y^2)-Z)<.5)
```

and compare U1 with U2.

MAP

Syntax `MAP\FIOWA global`
`MAP\FIOWABIG global`
`MAP\YBOS`
`MAP\HBOOK name`

The MAP command is used to map onto shared memory or a global section. The FIOWA, FIOWABIG, and NOVA options are only available under VAX/VMS. The YBOS option is only available under DEC AXP OSF/1. The HBOOK option is available for global sections under VAX/VMS and for shared memory under all UNIX systems supported in the CERN libraries.

HBOOK

Syntax `MAP\HBOOK name`

A "snapshot" is taken of the data in the named global section or shared memory. To access the most current data, re-enter the MAP command. The variables created are the same as with the RESTORE\HBOOK command. Use the GLOBALS command under VAX/VMS to see the global section names to which you have access. Refer to the RESTORE\HBOOK command for information on what variables are created and their definitions.

FIOWA

Syntax `MAP\FIOWA globalname`
`MAP\FIOWABIG globalname`

The MAP\FIOWA and MAP\FIOWABIG commands only work under VAX/VMS.

The MAP command is used to map onto a global section. A "snapshot" is taken of the data in the global section. To access the most current data, re-enter the MAP command. The variables created are the same as with the RESTORE\FIOWA command. Use the GLOBALS command to see the global section names to which you have access. Refer to the RESTORE command for information on what variables are created and their definitions.

Use the \FIOWABIG qualifier to access data sets made with the "big" version of FIOWA.

YBOS

Syntax `MAP\YBOS`

The MAP\YBOS command only works under Digital Unix.

The MAP command is used to map onto shared memory. A "snapshot" is taken of the data in the shared memory. To access the most current data, re-enter the MAP\YBOS command. The variables created are the same as with the RESTORE\YBOS and with the RESTORE\YBOS\DOTPLOT commands. Refer to the RESTORE command for information on what variables are created and their definitions.

MATRIX

Syntax `MATRIX m1 { m2 ... } nrow ncol`

Examples `MATRIX M 10 20`
`MATRIX M1 M2 M3 10 20`

The MATRIX command creates new matrices or changes the dimensions of existing matrices. Each mI will be a matrix with nrow rows and ncol columns.

If mI exists and is a matrix, then it will be either trimmed down to the specified dimensions or zero filled to expand it.

If mI exists but is not a matrix, it will be destroyed first and then re-created as a zero filled matrix with nrow rows and ncol columns. If mI does not exist, it will be created as a zero filled matrix with nrow rows and ncol columns.

MONITOR

Syntax `MONITOR { keyword }`

The MONITOR command is used to select a graphics output monitoring device. Table 2.46 lists the currently recognized graphics display device types.

Selecting a monitor type necessitates clearing the graphics.

The MONITOR command should be entered before opening an EDGR file.

device	*keyword*
Digital VT640	VT640
Digital VT241	VT241
Citoh CIT-467	CIT467
Tektronix 4010/12	TK4010
Tektronix 4107	TK4107
Plessey PT-100G	PT100G
Seiko GR-1105	GR1105
X Window system	X
Generic terminal	GENERIC

Table 2.46: Monitor device types and corresponding keywords

Disabling/enabling graphics monitor output

Use the MONITOR OFF command to disable graphics output to the monitor. This can be useful when executing command files which the user is confident of and does not need to monitor. The hardcopy output is not affected, and is much quicker to obtain. To make the previous monitor type again available, enter the MONITOR ON command. When ON is used, the monitor type that was last in use will again be available for monitoring graphics.

The generic terminal driver

The generic terminal driver is a driver that is controlled by a data file that can be created and modified by the user. The file contains lists of escape sequences for performing the

various terminal functions. Creating such a file for a given terminal may allow it to be used with PHYSICA even though it is not explicitly supported.

This generic terminal data file name is passed to the program by a logical name, on VMS systems, or by an environment variable, on UNIX systems. This name must be assigned before you run the PHYSICA program.

VMS: `$ DEFINE TRIUMF$GENTERM disk:[directory]filename`

UNIX: `% setenv TRIUMF_GENTERM <directory>filename`

NEWS

Syntax `NEWS`

Use the NEWS command to display the latest information about changes in the program and new features pertaining to the PHYSICA program.

ORIENTATION

Syntax `ORIENTATION keyword`

Default initial orientation = LANDSCAPE

The ORIENTATION command sets the graphics orientation. This requires that the graphics be cleared, which is done automatically.

Note: The ORIENTATION command should be entered before opening an EDGR file with the EDGR OPEN command.

There are two basic graphics orientations available.

keyword	*result*
LANDSCAPE	the world coordinate system large dimension is horizontal
PORTRAIT	the world coordinate system large dimension is vertical

The initial orientation is landscape.

Landscape orientation means, for example, that text, with a text angle of zero, will be drawn across the wide dimension of the plotter paper, or that a graph will appear with the x-axis drawn across the wide dimension of the plotter paper.

Portrait orientation means, for example, that text, with a text angle of zero, will be drawn across the narrow dimension of the plotter paper, or that a graph will appear with the x-axis drawn across the narrow dimension of the plotter paper.

The world coordinate system

The world coordinate system plotting units depend four things:

- the graphics orientation;
- the graphics hardcopy device type;
- the graphics hardcopy device size;
- the type of graphics units.

Refer to the `DEVICE` command, page 47, for tables showing the world coordinate system plotting units for the various devices.

The type of graphics units are chosen with the `SET UNITS` command.

PEAK

Syntax	`PEAK xout yout` `PEAK\PNUM xout yout num`
Qualifiers	`\PNUM`
Defaults	`num` = number of last drawn data curve, `\NOPNUM`

The `PEAK` command brings up the graphics cursor, and interacts with the user through a keystroke menu, to find the minima or maxima of the current data curve drawn on the screen. The coordinates of the peaks so found are stored in the specified variables `xout` and `yout`. By default, the data curve referenced is the last one drawn. When the graphics cursor is brought up, the program is waiting for input of a one or two character keycode.

Choosing the data curve

Syntax `PEAK\PNUM xout yout num`

Any one of the data curves currently on the screen may be specified explicitly by using the `\PNUM` qualifier and including the scalar `num`. This scalar specifies which data curve, in the order they were drawn, is to be selected.

X Windows

When running under X Windows, mouse button two toggles the continuous display of the graphics cursor location. The PHYSICA keyword `CUNITS` is the units type for these numbers. If `CUNITS` = `WORLD`, the numbers depend on the current units type, either `CM` or `IN`, as chosen

with `SET UNITS`. If `CUNITS` = `GRAPH`, the numbers displayed depend on the current graph axis scales. If `CUNITS` = `PERCENT`, the numbers depend on the current window.

Code keys

Remember that this command assumes that a graph is present on the screen and none of the plot characteristics have been changed after the graph was done. After entering the `PEAK` command, the graphics cursor will appear. Typing a special code key results in a specific action. See Table 2.47 on page 158.

key	*action*
`<`	moves the graphics cursor to the nearest peak (or dip) to the left
`>`	moves the graphics cursor to the nearest peak (or dip) to the right
`Pn`	defines criterion for peak finding to be `n` points monotonically increasing, followed by `n` points monotonically decreasing. (default `n`= 1) *Note*: works only with > when enabled, `n`= 0 disables this function
`Fn`	fits least-squares parabola to `2*n+1` nearby points (`n`= $1, \ldots, 9$) and moves the graphics cursor to the fitted peak location, e.g., `F3` would peak fit the surrounding 7 pts
`Cn`	use the $\mathtt{n}^{th}$ curve on the screen for subsequent codes
`+`	search for peaks (maxima)
`-`	search for dips (minima). This is the default.
`A`	amplitude required for peak (or dip). The user is prompted to enter a limiting amplitude by returning two cursor y-coordinates (abort with `Z`)
`Y`	set the y tolerance parameter. Move the cursor to the desired y value and enter any character (abort with `Z`)
`N`	disable the y tolerance flag
`Z`	aborts entry of cursor coordinates for `A,Y` codes
`D`	display the last peak position in graph coordinates
`R`	record and save the last peak position
`S`	save the last peak position and mark with a symbol
`/`	alphanumeric clear
`Q`	quit
All other characters ignored	

Table 2.47: Key codes for the `PEAK` command

PICK

Syntax	`PICK { xout yout { y1 { n1 } y2 { n2 } ... }}` `PICK\NPTS num { xout yout { y1 { n1 } y2 { n2 } ... }}`
Qualifiers	`\NPTS, \POLYGON, \MATRIX, \COUNTS, \MIN, \MAX, \DISPLAY`
Defaults	`nI = I, \NONPTS, \NOPOLYGON, \NOMATRIX, \NOCOUNTS, \NOMIN, \NOMAX, \DISPLAY`

The default action for the `PICK` command is to pick points off a graph which is currently displayed on the monitor screen, and to optionally save the values in output vectors, `xout` and `yout`. It is assumed that a graph is present on the screen and none of the plot characteristics have been changed after the plot was done. The graphics cursor is used, and various actions depend on which key is typed from the keyboard. If `xout` and `yout` are not entered, then no points will be saved.

X Windows

When running under X Windows, mouse button two toggles the continuous display of the graphics cursor location. The PHYSICA keyword `CUNITS` is the units type for these numbers. If `CUNITS` = `WORLD`, the numbers depend on the current units type, either `CM` or `IN`, as chosen with `SET UNITS`. If `CUNITS` = `GRAPH`, the numbers displayed depend on the current graph axis scales. If `CUNITS` = `PERCENT`, the numbers depend on the current window.

Specifying the number of points

Syntax `PICK\NPTS num { xout yout { y1 { n1 } y2 { n2 } ... }}`

When the `\NPTS` qualifier is used, a scalar, `num`, is expected which will be the maximum number of points that can be recorded with that command. When you have recorded `num` points, the command automatically stops.

Code keys

Remember that this command assumes that a graph is present on the screen and none of the plot characteristics have been changed after the graph was done. After entering the `PICK` command, the graphics cursor will appear. Typing a special code key results in a specific action. See Table 2.48.

If the `\-DISPLAY` qualifier is used, it turns off the message display for the default action, which is picking points off a graph. This was added to clean up the terminal window for those using the `PICK` command who do not desire to see the (x, y) locations or the menu. These can be turned on inside the `PICK` command by typing the `O` key.

key	*action*
`D`	Digitize the current crosshair position and display the coordinates corresponding to that location
`R`	same as `D`, but also record the coordinates, in graph units, of that location in the optional vectors `xout` and `yout`, record the interpolated value of curve `nI` in vector `yI`. The number of recorded points is displayed
`M`	same as `R`, but place a marker at the recorded location
`C`	same as `M`, but also connect that point to the last recorded point with a line segment
`O`	turn on message display
`Q`	quit picking points

Table 2.48: Key codes for the `PICK` command

Automatic digitizing of previously graphed data

It is possible to automatically digitize points off of previously drawn data curves by making use of the optional parameters `yI` (vector name) and `nI` (number). Assuming that the total number of curves that have been plotted on the currently displayed graph is at least `nI`, then the coordinate corresponding to the horizontal (independent) location of the crosshair will be interpolated on the `nI`th curve and saved in the vector `yI`. Linear interpolation is used, and extrapolation is allowed. If the number `nI` is not present following the vector `yI`, then `nI` defaults to `I`.

Examples

Suppose you have entered the following commands to draw three data curves:

```
GRAPH X1 Y1
GRAPH\NOAXES X2 Y2
GRAPH\NOAXES X3 Y3
```

To pick points off the three data sets, you could enter: `PICK XP YP YP1 YP2 YP3` Move the crosshair horizontally across the screen and type the `R` key at each location where you desire a value. The vector `YP1` will the the interpolated values from curve `(X1,Y1)`, vector `YP2` will contain those from `(X2,Y2)`, and vector `YP3` will contain those from `(X3,Y3)`. The horizontal (independent) coordinates will be contained in the vector `XP`.

Suppose you have drawn the same three data sets as above, but you only want to pick points off of the third curve drawn, that is, (X3,Y3). Enter: `PICK XP YP YP3 3`
Move the crosshairs horizontally across the screen and type the R key at the locations where you desire values off the third curve. The vector YP3 will contain the interpolated values from curve (X3,Y3), while the horizontal (independent) coordinates will be contained in vector XP.

Choosing the vertices of a polygon

Syntax	`PICK\POLYGON xp yp`

The \POLYGON qualifier indicates that you are picking vertex points for a polygon. The only difference between \POLYGON and the default, \NOPOLYGON, is that a final point is added when you quit picking points. This final point will be the same as the first point picked, thus closing the polygon.

Matrices

Syntax	`PICK\MATRIX nrow ncol mxout myout` `PICK\MATRIX\MIN { xin yin } m mxout myout` `PICK\MATRIX\MAX { xin yin } m mxout myout`
Defaults	xin = [1;2;3;...], yin = [1;2;3;...]

The PICK\MATRIX command allows you to pick points off the screen using the graphics cursor. The x coordinates of the points are placed in matrix mxout, while the y coordinates are placed in matrix myout. The user must specify the size of these output matrices in nrow, the number of rows, and ncol, the number of columns. Both mxout and myout will be created with the same dimensions.

This feature was included to be used in conjunction with the BIN2D\MATRIX command, page 9.

Finding regional minima for a matrix

The PICK\MATRIX\MIN command allows you to choose circular regions on the screen, by choosing centres and radii. The minimum value of the matrix, m, contained within this circle will be found. The x coordinate of these minima will be output into matrix mxout, the y coordinates will be output into matrix myout.

Finding regional maxima for a matrix

The PICK\MATRIX\MAX command allows the user to choose circular regions on the screen, by choosing centres and radii. The maximum value of the matrix, m, contained within this circle will be found. The x coordinate of these maxima will be output into matrix mxout, the y coordinates will be output into matrix myout.

Determining regional counts for data sets

Syntax	`PICK\COUNTS { xin } ydata counts` `PICK\COUNTS\MATRIX { xin yin } m counts`
Defaults	xin = [1;2;3;...], yin = [1;2;3;...]

The PICK\COUNTS command allows you to pick pairs of points off the screen. The number of counts of the vector ydata inside that region will be displayed on the screen and stored in the output vector, counts. The input vector, xin, should contain the x coordinates for the data ydata. xin defaults to [1;2;3;...] if not entered.

Matrices

The PICK\COUNTS\MATRIX command allows you to pick rectangular regions off the screen. The number of counts of the matrix inside each such region will be displayed on the screen and stored in the output vector, counts. The input vectors, xin and yin, should contain the x and y coordinates for the matrix, where xin corresponds to the columns and yin corresponds to the rows, that is, m[i,j] has coordinates (xin[j],yin[i]). xin and yin both default to [1;2;3;...] if not entered.

PIEGRAPH

Syntax	`PIEGRAPH w e f c o r { cx cy { a }}`
Qualifiers	\GRAPH, \PERCENT, \WORLD
Defaults	\PERCENT, a = 0

The PIEGRAPH command draws piecharts. The first five parameters are vectors which define each wedge of the pie. See Table 2.49.

Coordinates

The parameters r, cx, cy, and a should be literal constants or scalars. The radius of the pie chart is r. The parameters cx and cy represent the x and y location of the centre of the pie chart. The starting angle, in degrees, for the first wedge can be optionally specified with the parameter a, which defaults to zero.

If either cx or cy is not entered, the graphics cursor will be used to choose that coordinate. If both cx and cy are present, the graphics cursor will not be used.

w	wedge values, in whatever units are applicable the sum of all the wedge values is 100% of the pie chart
e	explode values, as a percent of the radius
f	filling codes $\texttt{f} = 0$ then no filling $1 \le \|\texttt{f}\| \le 10$ then fill with hatch pattern $11 \le \|\texttt{f}\| \le 99$ then fill with dot pattern $\texttt{f} < 0$ then the outline of the wedge is not drawn, but hatch pattern $\|\texttt{f}\|$ is drawn
c	colour numbers for the wedge fill pattern
o	colour numbers for the wedge outline

Table 2.49: The pie wedge defining characteristics

Units

The pie chart coordinates may be expressed in three types of units, which are chosen by command qualifier. The default is `\PERCENT`. See Table 2.50 for a listing of the qualifiers and their interpretations.

qualifier	*interpretation of the coordinates*
`\PERCENT`	percentages of the current window, as chosen with the `WINDOW` command.
`\GRAPH`	graph units, that is, the units defined by the minimum and maximum values for the last graph drawn. If no graph has been drawn yet, the defaults are $-1 \le x \le 1$ and $-1 \le y \le 1$
`\WORLD`	centimeters or inches, as chosen with the `SET UNITS` command

Table 2.50: Types of units recognized by the `PIEGRAPH` command

For example, if the `\PERCENT` qualifier is used, then a location of $(50, 50)$ represents the centre of the current window. If the `\WORLD` qualifier is used, the coordinates are in units of the world coordinate system, the plotting units. The default world coordinate system units are centimeters. See the `DEVICE` command for tables showing the dependence of plotting units on the graphics hardcopy output device.

Figure 2.19 illustrates how each individual pie wedge is drawn.

Pie wedge filling

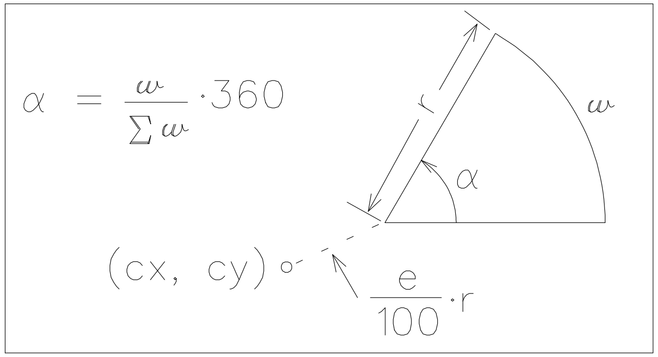


Figure 2.19: Pie chart wedge definition

Each wedge of the pie can be filled with either a grey scale dot fill pattern or a hatch pattern.

Hatch patterns

A hatch pattern is composed of an angle and one to ten spacings. These spacings are simply cycled through as the region is being filled, that is, a line is drawn inside the region at the specified angle, then a parallel line is drawn at the first spacing, then another parallel line is drawn at the second spacing, and so on for the number of spacings in that pattern. This process is repeated until the region is filled. The hatch patterns can be redefined with the `SET HATCH` command and displayed with the `DISPLAY FILL` command. There are ten hatch patterns available.

Dot fill patterns

A dot pattern is of the form: uv, where the digit u is the increment number of dots to light up horizontally, $1 \le u \le 9$, and the digit v is the increment number of dots to light up vertically, $1 \le v \le 9$. For example, a dot pattern of 34 means to light up every third dot horizontally and every fourth dot vertically. If uv is negative, then the dots are erased instead of turned on.

PostScript output

For PostScript output, set the `POSTRES` keyword to the appropriate resolution for your hardcopy device, using the `SET` command. Use a combination of line thickness and resolution for a quicker resulting picture. For example, the Lexmark inkjet printer has a resolution of 360 dpi. A “good” picture can be obtained with $\texttt{POSTRES} = 180$ and $\texttt{LINTHK} = 2$.

Example

The following script produces Figure 2.20.

```
PIE_VALUES=[12.1;18.2;24.2;15.2;30.3]
EXPLODE=[10;10;0;25;0]
FILL=[8;33;44;7;22]
WCOLOUR=[1;2;3;4;5]
OCOLOUR=[5;4;3;2;1]
PIEGRAPH PIE_VALUES EXPLODE FILL WCOLOUR OCOLOUR 40 50 50
```

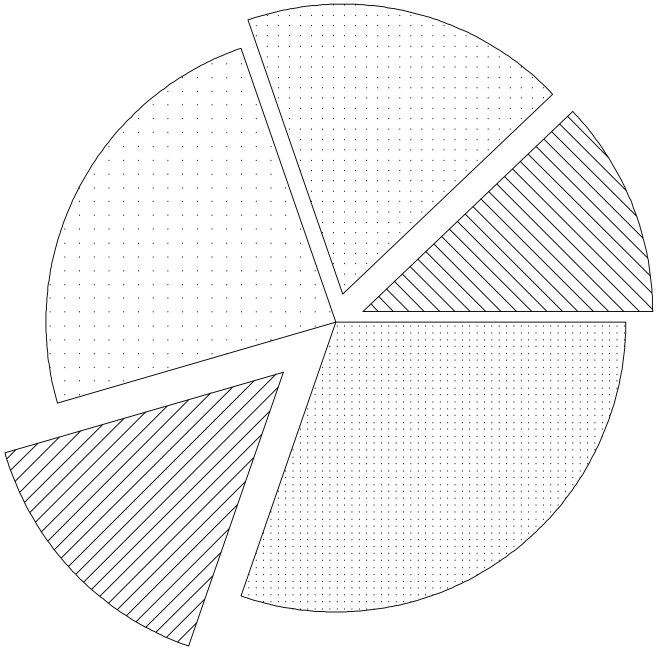


Figure 2.20: An example of a pie chart

PLOTTEXT

Syntax PLOTTEXT filename

The PLOTTEXT command is used to draw lines of formatted text that are read from a file. Text formatting commands can be interspersed in the lines of text. The text formatting

commands are summarized in Table 2.51.

Every text formatting command *must* be bracketed by the command delimiters, except for comment lines, which do not require the trailing command delimiter.

There is also a set of special reserved character names, see Figure 2.6 on page 61. These special names must be enclosed by the command delimiters, for example, <alpha><beta>. See the DISPLAY command on page 58 for information on how to display these special reserved names.

Environment variables in file names

For UNIX users, it is now possible to use an environment variable in a file name, if the environment variable is preceeded by a $. For example,

```
setenv FILE mytext
physica
plottext $FILE
```

The environment variable can be just the first part of the filename, for example,

```
setenv FILE my
physica
plottext $FILEtext
```

Comments

Command form: <! comment...

To insert comment lines in a formatted text file, that is, lines that will not be drawn on the plot, include <! in positions one and two of a line. The rest of the line will be considered to be a comment and will be ignored. This is useful for documenting formatted text files. For example, this input:

```
<!
<! Use PORTRAIT orientation
<!
<H3%><S5%><FROMAN.SWISSL>This is an example of a formatted
<! text file with comments
```

produces this output:

This is an example of a formatted
text file with comments

Command delimiters

`<!`	comment line
`<Qx>,<Qxy>`	set command delimiters
`<->`	line continuation
`< >`	insert blank line
`<Hxx.x>`, `<Hxx.x%>`	text height
`<Sxx.x>`, `<Sxx.x%>`	line spacing
`<Mxx.x>`, `<Mxx.x%>`	left margin
`<B>`, `<Bn>`, `<Bn:m>`	grey scale or hatch fill
`<Cn>`	colour
`<Ffontname>`	text font
`<JC>`, `<JCxx.x>`, `<JCxx.x%>`	centre
`<JL>`, `<JLxx.x>`, `<JLxx.x%>`	left justify
`<JR>`, `<JRxx.x>`, `<JRxx.x%>`	right justify
`<Zxx.x>`, `<Zxx.x%>`	insert horizontal space
`<_>`	sub-script mode
`<^>`	super-script mode
`<em>`	italics mode
`<X>`	hexadecimal mode
`<\b>`	macron (bar) under previous character
`<\d>`	dot under previous character
`<\^>`	circumflex over previous character
`<\'>`	acute over previous character
`<\‘>`	grave over previous character
`<\">`	umlaut over previous character
`<\~>`	tilde over previous character
`<\=>`	macron (bar) over previous character
`<\.>`	dot over previous character
`<\u>`	breve over previous character
`<\v>`	v over previous character
`<\H>`	dieresis (double quote) over previous character

Table 2.51: `PLOTTEXT` command text formatting commands

Command form: `<Qx>`, `<Qxy>`
Default delimiters: < and >

To set the leading command delimiter to `x` and the trailing command delimiter to `y`, include `<Qxy>` in the formatted text. If a single character, `x`, is to be the leading *and* the trailing command delimiter, simply include `<Qx>` in the formatted text. This is useful when the current command delimiters are required as characters to be drawn.

Within a file, the command delimiters will remain as set until changed with another `<Qx>` or `<Qxy>` command, but the command delimiters always default to < and > when the `PLOTTEXT` command is entered.

For example, this input:

```
<FTSAN><H.6><S1.2>Change the delimiters from <Q\> < and >
to \Q<>\ \ and back to <Q\> < >\Q<>\
```

produces this output:

```
Change the delimiters from  < and >
to  \ and back to  < >
```

Continuation Lines

Command form: `<->`

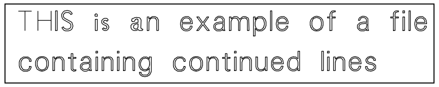
To continue a line of formatted text onto the next line of a file, include `<->` at the end of the line. The next line of text will be drawn at the end of the line with the continuation. Blanks at the end of the line with the continuation command, but before the `<->` will be included, as well as blanks at the beginning of the next line.

This is useful when you have so many text formatting commands in a line that the line becomes very long. The maximum input line length is 255 characters, so it is necessary to continue the line onto two or more input lines.

For example, this input:

```
<H0.5><S1>TH<FROMAN.FUTURA>IS <FROMAN.FASHON>is <->
<FROMAN.SERIF>a<FROMAN.SWISSL>n example <->
of a file
containing <->
continued lines
```

produces this output:



Inserting a blank line

Command form: < >

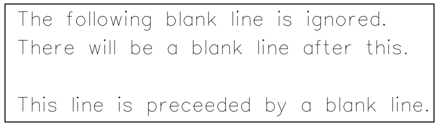
To insert a blank line in the plotted text use < >. Blank lines that are encountered in a formatted text file are simply ignored.

For example, this input:

```
<H.6><S1.2>The following blank line is ignored.

There will be a blank line after this.< >
This line is preceded by a blank line.
```

produces this output:



Character height

Command form: <Hxx.x>, <Hxx.x%>
Default height: the current value of TXTHIT

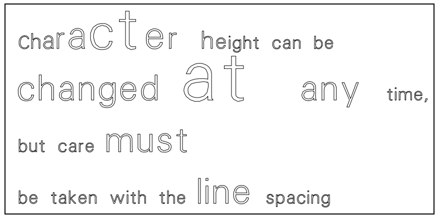
To set the character height to xx.x units, include <Hxx.x> in the formatted text. This applies to subsequent text. The units, either centimeters or inches, are defined by the SET UNITS command. To set the height as a percentage of the height of the current window, that is, YUWIND - YLWIND, use <Hxx.x%>.

Within a file, the text height will remain as set until changed with another <Hxx.x> or <Hxx.x%> command, but the text height always defaults to the current value of TXTHIT when the PLOTTEXT command is entered.

For example, this input:


```
<FROMAN.SWISSL><H0.5><S2>C<H.6>h<H.7>a<H.9>r<->
<H1.1>a<H1.4>c<H1.7>t<H1.4><->
<H1.1>e<H.9>r <H.7>h<H.6>e<H.5>ight can be
<H1>changed <H2>at <H1>any <H.5>time,
but care <H1>must<H.5>
<S2>be taken with the <H1>line<H.5> spacing
```

produces this output:



Line spacing

Command form: <Sxx.x>, <Sxx.x%>
Default line spacing: 1.5× TXTHIT

To set the line spacing to xx.x units, include <Sxx.x> in the formatted text. The units, either centimeters or inches, are defined by the SET UNITS command. To set the line spacing as a percentage of the height of the current window, that is, YUWIND - YLWIND, use <Sxx.x%>.

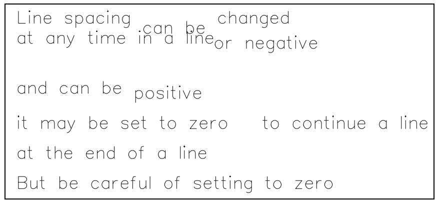
The line spacing is the distance from the bottom of the previous text line to the bottom of the current text line. This spacing is the automatic vertical spacing to be used between text lines, but line spacing takes place *immediately*, so that each character in a text line may be drawn at any vertical distance, always measured from the bottom of the previous text line.

Within a file, the line spacing will remain as set until changed with another <Sxx.x> or <Sxx.x%> command, but the line spacing always defaults to 1.5× the current value of TXTHIT when the PLOTTEXT command is entered.

For example, this input:

```
<H0.5><S.8>Line spacing <S1.2>can be <S.8>changed
at any time in a line<S1.4>
and can be <S1.6>positive <S-.4>or negative<S1.4>
it may be set to zero
<S0><JL10>to continue a line<S1.2>
But be careful of setting to zero<S0>
at the end of a line
```

produces this output:



Left margin

Command form: <Mxx.x>, <Mxx.x%>
Default left margin: $0.01 \times$ (XUWIND − XLWIND)

To set the left margin to xx.x units, include <Mxx.x> in the formatted text. The units, either centimeters or inches, are defined by the SET UNITS command. To set the left margin as a percentage of the width of the current window, that is, XUWIND - XLWIND, use <Mxx.x%>.

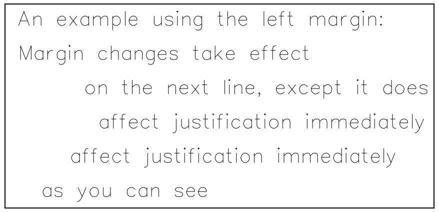
The left margin applies to the subsequent text lines. It does *not* apply to the current line.

Within a file, the left margin will remain as set until changed with another <Mxx.x> or <Mxx.x%> command, but the left margin always defaults to $0.01\times$ the width of the current window.

For example, this input:

```
<H0.5><S1.2>An example using the left margin:
<M2.5>Margin changes take effect
on the next line, except it does
<JL3>affect justification immediately
<M3.5><JL2>affect justification immediately<M1>
as you can see
```


produces this output:



Bolding

Command form: <Bn>, <Bn:m>, <B>
Default: no bolding

Bolding means that the text characters will be filled with a dot pattern or a hatch pattern. Bolding should only be used with the fonts: ROMAN.SERIF, ROMAN.FUTURA, ROMAN.FASHON, ROMAN.LOGO1, ROMAN.SWISSL, ROMAN.SWISSM, ROMAN.SWISSB, or TRIUMF.OUTLINE.

When <Bn> is encountered, subsequent characters will be filled. Within a file, bolding will remain as set until different fill pattern(s) are selected with another <Bn> or <Bn:m> command. Bolding is turned off by entering the <B> command.

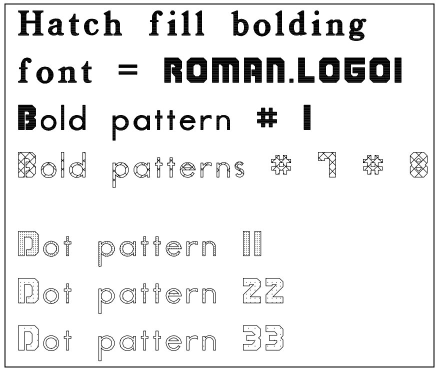
		n	=	0	then no filling
1	≤	|n|	≤	10	then fill with hatch pattern
11	≤	|n|	≤	99	then fill with dot pattern
		n	<	0	then the outline of the character is not drawn, but hatch pattern |n| is drawn

When <Bn:m> is encountered, the two *hatch* patterns |n| and |m| are used to fill the characters. This can be used to create a cross-hatching pattern. This feature *cannot* be used with two dot patterns.

For example, this input:

```
<H.8><S1.5><B1><FROMAN.SERIF>Hatch fill bolding
font = <FROMAN.LOGO1>ROMAN.LOGO1
Bold pattern # 1
<B7:8>Bold patterns # 7 # 8<B>
<H.8><S2.5><B11>Dot pattern 11
<S1.5><B22>Dot pattern 22
<B33>Dot pattern 33<B>
```

produces this output:



Dot fill patterns

A dot pattern is of the form: uv, where the digit u is the increment number of dots to light up horizontally, $1 \leq u \leq 9$, and the digit v is the increment number of dots to light up vertically, $1 \leq v \leq 9$. For example, a dot pattern of 34 means to light up every third dot horizontally and every fourth dot vertically. If uv is negative, then the dots are erased instead of turned on. Note that 200 is interpreted the same as 211, that is, every dot is lit.

PostScript output

For PostScript output, set the POSTRES keyword to the appropriate resolution for your hard-copy device, using the SET command. Use a combination of line thickness and resolution for a quicker resulting picture. For example, the Lexmark inkjet printer has a resolution of 360 dpi. A “good” picture can be obtained with POSTRES $= 180$ and LINTHK $= 2$.

Hatch patterns

A hatch pattern is composed of an angle and one to ten spacings. These spacings are simply cycled through as the region is being filled, that is, a line is drawn inside the region at the specified angle, then a parallel line is drawn at the first spacing, then another parallel line is drawn at the second spacing, and so on for the number of spacings in that pattern. This process is repeated until the region is filled. The hatch patterns can be redefined with the SET HATCH command and displayed with the DISPLAY FILL command. There are ten hatch patterns available.

Colour

Command form: <Cn>
Default colour: the current colour as set by the COLOUR command

To set the colour, include <Cn> in the text. This applies to subsequent text.

Within a file, the colour will remain as set until changed with another <Cn> command, but the colour always defaults to the current colour as set by the COLOUR command. See Table 2.6, page 20, for the colour associated with each colour code.

For example:

```
<FSCRIPT.2><S1.2><H.6><C2>Colour <C3>cannot <C4>be <C5>displayed
<C6>on <C7>photocopies, <C1>but <C2>try <C3>this
<C4>formatted <C5>text <C6>on a <C7>colour device
```

Font

Command form: <Ffontname>
Default font: the current font as selected with the SET FONT command

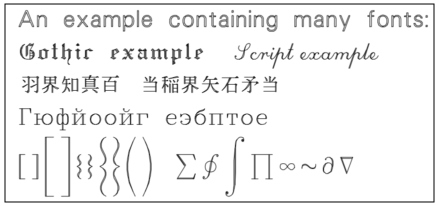
To select a font, include <Ffontname> in the formatted text. This will apply to subsequent text. Within a file, the font will remain as set until changed with another <Ffontname> command, but the font always defaults to the current font as set by the SET FONT command.

The manual TRIUMF GRAPHICS FONTS contains font tables. See Table 2.60 on page 243 for a list of the font names.

For example, this input:

```
<H0.5><S1><FROMAN.SWISSL>An example containing many fonts:
<FGOTHIC.ENGLISH>Gothic example  <FSCRIPT.2>Script example
<FKANJI4>Kanji example
<FCYRILLIC.2>Cyrillic example
<S1.5><FMATH>0123456789 SJKMOPQR
```

produces this output:



Centre justification

Command form: <JC>, <JCxx.x>, <JCxx.x%>

To centre text at the location xx.x units from the left edge of the current window, that is, XLWIND, include <JCxx.x> before the text. The units, either centimeters or inches, are defined by the SET UNITS command. To centre the text at a location defined as a percentage of the width of the current window, that is, XUWIND - XLWIND, use <JCxx.x%>.

If no number is present, that is, <JC>, the text will be centred at the location midway in the window between the right edge, XUWIND, and the left edge.

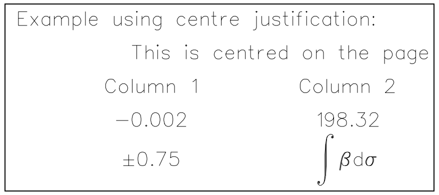
The centred text will include all text after the <JC>, <JCxx.x> or <JCxx.x%> command and up to the next justification command (justify left, right, or centre) or up to the end of the line, whichever comes first.

It is possible to centre about any position on the current line, in any order, and to mix left and right justifications with centering on the same line.

For example, this input:

```
<H0.5><S1.2>Example using centre justification:
<JC>This is centred on the page
<JC5>Column 1<JC12>Column 2
<JC5>-0.002<JC12>198.32
<S1.4><JC5><PM>0.75<JC12><Integrl><beta>d<sigma>
```

produces this output:



Left justification

Command form: <JL>, <JLxx.x>, <JLxx.x%>

To left justify text at the location xx.x units from the left edge of the current window, that is, XLWIND, include <JLxx.x> before the text. The units, either centimeters or inches, are defined by the SET UNITS command. To left justify the text at a location defined as a percentage of the width of the current window, that is, XUWIND - XLWIND, use <JLxx.x%>.

If no number is present, that is, <JL>, the text will be left justified at the left edge of the window.

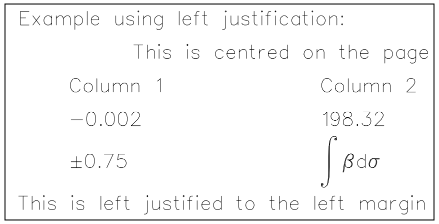
The left justified text will include all text after the <JL>, <JLxx.x> or <JLxx.x%> command and up to the next justification command (justify left, right, or centre) or up to the end of the line, whichever comes first.

It is possible to left justify about any position on the current line, in any order, and to mix left and right justifications with centering on the same line.

For example, this input:

```
<H0.5><S1.2>Example using left justification:
<JC>This is centred on the page
<JL11>Column 2<JL2>Column 1
<JL2>-0.002<JL11>198.32
<S1.5><JL2><PM>0.75<JL11><INTEGRL><beta>d<sigma>
This is left justified to the left margin
```

produces this output:



Right justification

Command form: <JR>, <JRxx.x>, <JRxx.x%>

To right justify text at the location xx.x units from the left edge of the current window, that is, XLWIND, include <JRxx.x> before the text. The units, either centimeters or inches, are defined by the SET UNITS command. To right justify the text at a location defined as a percentage of the width of the current window, that is, XUWIND - XLWIND, use <JRxx.x%>.

If no number is present, that is, <JR>, the text will be right justified at the right edge of the window, that is, XUWIND.

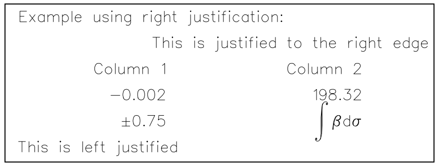
The right justified text will include all text after the <JR>, <JRxx.x> or <JRxx.x%> command and up to the next justification command (justify left, right, or centre) or up to the end of the line, whichever comes first.

It is possible to left justify about any position on the current line, in any order, and to mix left and right justifications with centering on the same line.

For example, this input:

```
<H0.5><S1.2>Example using right justification:
<JR>This is justified to the right edge
<JR16>Column 2<JR7>Column 1
<JR7>-0.002<JR16>198.32
<JR7><PM>0.75<JR16><INTEGRL><beta>d<sigma>
This is left justified
```

produces this output:



Horizontal spacing

Command form: <Zxx.x>, <Zxx.x%>

To insert a horizontal space of xx.x units, include <Zxx.x> in the formatted text. The units, either centimeters or inches, are defined by the SET UNITS command. To set the horizontal space as a percentage of the width of the current window, that is, XUWIND - XLWIND, use <Zxx.x%>.

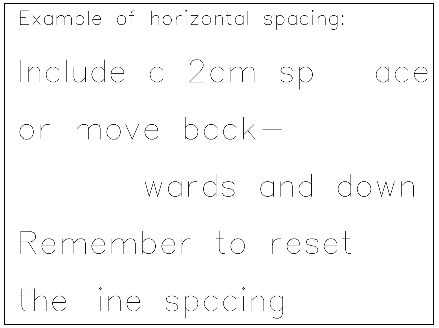
The horizontal space, which may be positive or negative, is measured from the current location.

Note: The right, centre and left justification tabs, are measured from the left edge of the window.

For example, this input:


```
<H0.5><S.8>Example of horizontal spacing:
<H.8><S2>Include a 2cm sp<Z2>ace
or move back-<Z-5><S4>wards and down
<S2>Remember to reset
the line spacing
```

produces this output:



Sub-script mode

Command form: <_>

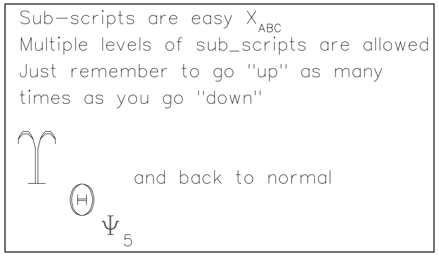
To enter sub-script mode, include <_> in the formatted text. Subsequent text will have 60% the current height and will be vertically spaced down a distance equal to 60% of the current height. This allows for multiple levels of sub-scripts, but for every level of sub-scripting, there must be a corresponding level of super-scripting to bring the text back "up".

Within a file, each level of sub-scripting remain in effect until <^>, super-script mode, is encountered.

For example, this input:

```
<H0.5><S1>Sub-scripts are easy X<_>ABC<^>
Multiple levels of sub_scripts are allowed
Just remember to go "up" as many
times as you go "down"
<S3><H2><UPSILON><_><THETA><_><PSI><_>5<^><^><^><->
<H.5>and back to normal
```

produces this output:



Super-script mode

Command form: <^>

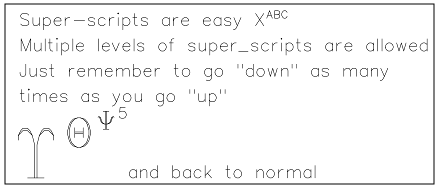
To enter super-script mode, include <^> in the formatted text. Subsequent text will have 60% the current height and will be vertically spaced up a distance equal to 60% of the current height. This allows for multiple levels of super-scripts, but for every level of super-scripting, there must be a corresponding level of sub-scripting to bring the text back "down".

Within a file, each level of super-scripting remain in effect until <_>, sub-script mode, is encountered.

For example, this input:

```
<H0.5><S1>Super-scripts are easy X<^>ABC<_>
Multiple levels of super_scripts are allowed
Just remember to go "down" as many
times as you go "up"
<S3><H2><UPSILON><^><THETA><^><PSI><^>5<_><_><_><->
<H.5>and back to normal
```

produces this output:



Slanted mode

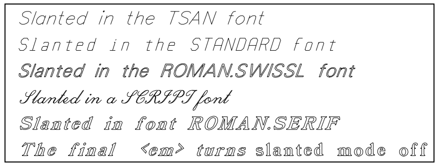
Command form: <em>

To control slanted, or emphasis, mode, include <em> in the formatted text. <em> acts like a toggle switch in that the first time <em> is encountered slanted mode will be turned on, that is, subsequent text will be slanted, and the next time it is encountered, slanted mode will be turned off. Slanted mode may be used for any character in any font, even with bolding on.

For example, this input:

```
<em><H0.6><S1.2>Slanted in the TSAN font
<FSTANDARD>Slanted in the STANDARD font
<FROMAN.SWISSL>Slanted in the ROMAN.SWISSL font
<FSCRIPT.2>Slanted in a SCRIPT font
<FROMAN.SERIF>Slanted in font ROMAN.SERIF
The final <Q\> <em>\Q<>\ turns <em>slanted mode off
```

produces this output:



Note: The graphics editor EDGR does not recognize slanted mode. So, if you open an EDGR file and include slanted text, when you edit the graphics with EDGR, the text will not be slanted.

Hexadecimal mode

Command form: <X>

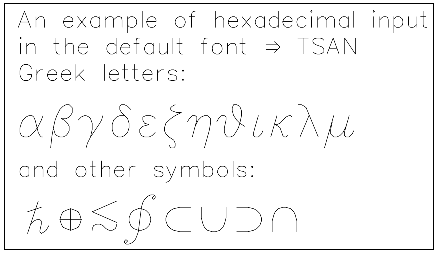
To control hexadecimal text input mode, include <X> in the formatted text. <X> acts like a toggle switch in that the first time <X> is encountered, hexadecimal mode is turned on, and the next time it is encountered, hexadecimal mode is turned off. Hexadecimal mode means that the text is be assumed to be pairs of hexadecimal digits that represent non-keyboard characters.

The hexadecimal codes for characters depend on which font is being used. Refer to the font tables in the TRIUMF GRAPHICS FONTS manual for these codes. Also, see the DISPLAY FONT command for information on how to display any font table.

For example, this input:

```
<H0.5><S.8>An example of hexadecimal input
in the default font <Rightarrow> TSAN
Greek letters:
<H1><S1.7><X>CACBCCCDCECFDADBDCDDDEDF<X>
<H.5><S1.2>and other symbols:
<H1><S1.7><X>4AAFB96954555657<X>
```

produces this output:



Accents

Command	
<\b>	macron (bar) under previous character
<\d>	dot under previous character
<\^>	circumflex over previous character
<\'>	acute over previous character
<\`>	grave over previous character
<\">	umlaut over previous character
<\~>	tilde over previous character
<\=>	macron (bar) over previous character
<\.>	dot over previous character
<\u>	breve over previous character
<\v>	v over previous character
<\H>	dieresis (double quote) over previous character

Table 2.52: Formatted text accent special characters

To place one of the special accents on a character, insert the appropriate command immediately after that character. The accent will be centred over or under that character. See Table 2.52 for a listing of the accent commands. See Figure 2.21 for examples of the accents on the letter "o" in the TRIUMF.2 font.

Not all fonts allow for accents. For example, the cyrillic font or the hiragana font. The accent

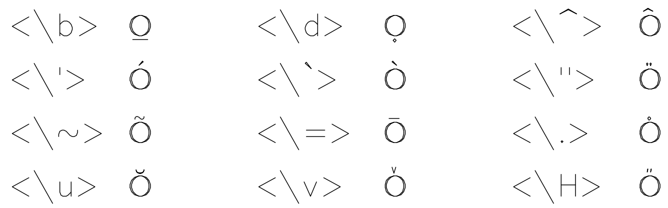


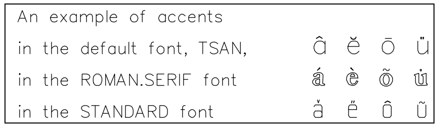
Figure 2.21: Example accents on the letter "o"

will be positioned fairly well over the lower case letters "a", "e", "o", and "u", *but not perfectly positioned*. You can use EDGR to relocate the accent if its position is not to your liking.

For example, this input:

```
<H0.3><S0.8>An example of accents
in the default font, TSAN, <->
<JL40%><H0.5>a<\^> e<\u> o<\=> u<\"><H0.3>
<FTSAN>in the ROMAN.SERIF font <->
<FROMAN.SERIF><JL40%><H0.5>a<\'> e<\`> o<\~> u<\.><H0.3>
<FTSAN>in the STANDARD font <->
<FSTANDARD><JL40%><H0.5>a<\v> e<\H> o<\^> u<\~>
```

produces this output:



POLYGON

Syntax	`POLYGON xpoly ypoly xdata ydata key`
Qualifiers	`\INSIDE`
Defaults	`\INSIDE`

The `POLYGON` command creates a vector, `key`, which will have the same length as the input vectors, `xdata` and `ydata`. By default, `key[i]` $= 1$ if the point (`xdata[i]`,`ydata[i]`) is inside the polygon defined by input vectors `xpoly` and `ypoly`, otherwise `key[i]` $= 0$. If the `\NOINSIDE` qualifier is used, `key[i]` $= 0$ if the point (`xdata[i]`,`ydata[i]`) is inside the polygon, otherwise `key[i]` $= 1$.

Example

You can use this command is conjunction with the PICK\POLYGON command to choose a polygon and the DESTROY command to eliminate data points within the chosen polygon. The following script produces Figure 2.22.

```
GEN\RANDOM X -5 5 200  ! generate some "data"
GEN\RANDOM Y 10 20 200 !
SET PCHAR -16          ! choose unjoined point plotting symbol
WINDOW 5               !
GRAPH X Y              ! display the data graphically
PICK\POLYGON XP YP     ! interactively choose a polygon around the
                       !  data you want to eliminate
POLYGON XP YP X Y K    ! create key vector, K
DESTROY X Y IFF (K=1)  ! eliminate unwanted data
WINDOW 6               !
GRAPH X Y              ! display data without unwanted points
SET PCHAR 0            ! choose no plotting symbol, joined
GRAPH\NOAXES XP YP     ! overlay the polygon you chose above
```

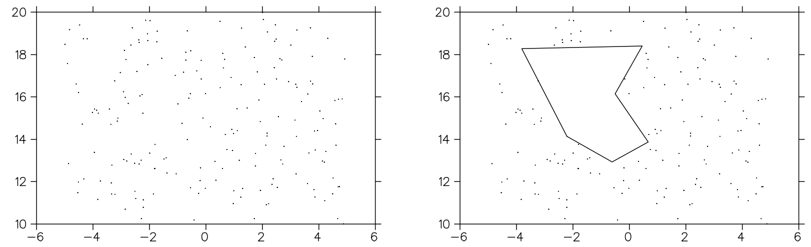


Figure 2.22: An example demonstrating the POLYGON command

QUIT

Syntax QUIT

Qualifiers \CLEAR

Defaults \NOCLEAR

The QUIT command is the cleanest way to stop the program. This does a complete FORTRAN STOP. You cannot re-enter PHYSICA after quitting, without re-running the program.

If the \CLEAR qualifier is appended to the QUIT command, the graphics is cleared before the program is stopped.

Commands

READ

Syntax

```
READ file{\line_range} x1{\c1} { x2{\c2} ... }
READ\FORMAT file{\line_range} (frmt) x1 { x2 ... }
READ\UNFORMATTED file{\line_range} (frmt) x1 { x2 ... }

READ\SCALAR file{\n} s1{\c1} { s2{\c2} ... }
READ\SCALARS\FORMAT file{\n} (frmt) s1 { s2 ... }
READ\SCALARS\UNFORMATTED file{\n} (frmt) s1 { s2 ... }

READ\MATRIX file{\n} m nrows { ncols }
READ\MATRIX\FORMAT file{\n} (frmt) m nrows { ncols }
READ\MATRIX\UNFORMATTED file\n (frmt) m nrows { ncols }

READ\TEXT file{\line_range} txtvar
READ\TEXT\FORMAT file{\line_range} (frmt) txtvar
READ\TEXT\UNFORMATTED file{\line_range} (frmt) txtvar
```

Qualifiers \VECTORS, \SCALARS, \MATRIX, \TEXT, \ASCII, \UNFORMATTED, \FORMAT, \CONTINUE, \CLOSE, \APPEND, \OVERLAY, \EXTEND, \ERRSTOP, \ERRFILL, \ERRSKIP, \FLIPPED, \MESSAGES

Defaults \VECTORS, \ASCII, \-FORMAT, \-CONTINUE, \-CLOSE, \-APPEND, \-OVERLAY, \EXTEND, \ERRSTOP, \FLIPPED, \MESSAGES

Examples

```
READ FILE.DAT X Y Z
READ\APPEND\FORMAT FILE.DAT\3 (2X,3F10.2) X Y Z
READ\FORMAT FILE.DAT\[2:100:2] (6X,F10.3,2X,F10.3) X\3 Y\1 Z\7
READ FILE.DAT 4X

READ\SCALAR FILE.DAT A B C
READ\SCALAR\FORMAT FILE.DAT\3 (2X,3F10.2) A B C
READ\SCALAR FILE.DAT\2 A\3 B\1 C\7

READ\MATRIX FILE.DAT M 10 20
READ\MATRIX\FORMAT FILE.DAT\3 (7(F10.3,2X)) M 10 20

READ\TEXT FILE.DAT T
READ\TEXT FILE.DAT\3 T
READ\TEXT\FORMAT FILE.DAT\3 (2X,A10) T[2]
READ\TEXT FILE.DAT\[2;3;15] T
```

READ is a general purpose command for reading vectors, scalars, a matrix, or string variables from a file. The maximum record length that can be read is 32768 bytes.

The variable type that will be read is determined by a command qualifier. The default, requiring no special qualifier, is to read data into vectors. Reading data into other variable types is chosen by using the appropriate qualifier. Refer to Table 2.53. The parameters that are expected depend on which of these qualifiers is used.

variable type	*qualifier*	
multiple vectors	\VECTORS	(default)
multiple scalars	\SCALARS	
one matrix	\MATRIX	
one string or one string array	\TEXT	

Table 2.53: Variables that can be read and their required qualifiers

By default, informational messages are displayed on the terminal monitor. If the \NOMESSAGES, or \-MESSAGES, qualifier is used, these informational messages will not be displayed.

Environment variables in file names

For UNIX users, it is now possible to use an environment variable in a file name, if the environment variable is preceeded by a $. For example,

```
setenv FILE dum.dat
physica
read $FILE x y z
```

The environment variable can be just the first part of the filename, for example,

```
setenv FILE dum
physica
read $FILE.dat x y z
```

Opening and closing files

By default, the file is closed after the read, so that a subsequent READ of the same file will start reading at the first record of the file. If the \CONTINUE qualifier is used, the file will not be closed after processing that READ command, and a subsequent READ of the same file will begin reading at the next record. If the \CLOSE qualifier is used, the file will be closed first and then reopened before reading, so that READ\CLOSE always begins reading with the first record in the file.

Examples

The command READ\SCALARS\CLOSE\CONTINUE DUM.DAT N first closes the file DUM.DAT, reads the first record, and leaves the file open.

If you entered: READ\SCALARS DUM.DAT N after the previous command, the second record would be read.

The commands

```
READ\CLOSE\CONTINUE DUM.DAT\[1:10] X
READ\CONTINUE DUM.DAT\[1:10] Y
```

would read the first ten records into vector X and the next ten records into vector Y.

Reading data into vectors

By default, vectors are read from columns of numbers in an ASCII file. The file is read by records, using free format. The cIth column is placed into vector xI. cI defaults to I. Every record is read, from record 1 to the end of file. If xI exists, it will be destroyed, and a new xI vector created.

ASCII files

Syntax	READ file{\line_range} x1{\c1} { x2{\c2} ... } READ\FORMAT file{\line_range} (frmt) x1 { x2 ... }
Qualifiers	\FORMAT, \CONTINUE, \CLOSE, \APPEND, \OVERLAY, \EXTEND, \ERRSTOP, \ERRFILL, \ERRSKIP, \MESSAGES
Defaults	\-FORMAT, \-CONTINUE, \-CLOSE, \-APPEND, \-OVERLAY, \EXTEND, \ERRSTOP, \MESSAGES

By default, or if the \ASCII qualifier is used, the file is assumed to be an ASCII file and is read by records, starting with the first record.

A scalar appended to the file name, file\n, specifies the starting record. The first $n-1$ records will be skipped.

A vector appended to the file name, file\x, specifies from which records to read data. The first x$[1]-1$ records will be skipped. The data will be read from records x[1], x[2], ..., x[#]. Records x$[i]+1$ to x$[i+1]-1$ will be skipped. The vector x must be monotonically increasing.

By default, the I$_{th}$ column is placed into vector xI. The column number can be specified by appending a scalar, cI, to the vector name as a qualifier. In this case, the cI$_{th}$ column can be placed into the xI vector. For example, after the command:

```
READ DUM.DAT W\2 X\4 Y Z\1
```

W would contain column 2, X would contain column 4, Y would default to column 3, and Z would contain column 1.

By default, free format is used for reading the data. Number fields can be separated by blanks or by commas. The \FORMAT qualifier must be used to indicate that a format is present. The format must be enclosed in parentheses, (and). If a format is used, column numbers *cannot* be specified. Standard FORTRAN formats are valid, but only REAL variables can be read, so do not use INTEGER, LOGICAL or CHARACTER formats. All values are converted to REAL*8 for internal storage.

To read different numbers of elements into vectors with a single READ command, use the \NOEXTEND, or \-EXTEND, qualifier. The output length of a vector will be number of values that are read, to a maximum of that vector's original length. For example, suppose that vector X has length 10 and vector Y has length 20, and suppose you enter:

```
READ\-EXTEND file\[1:20] X Y
```

If 20 records are read, vector X will be made with a length of 10 and vector Y will be made with a length of 20. If only 15 records are read, vector X will have length 10 but vector Y will only have length 15.

By default, a new vector is created to hold the newly read data. If the \OVERLAY qualifier is used, an existing vector will have the newly read data overlayed on the original data. The resultant vector length may be longer, but never shorter. Use the \APPEND qualifier to append the newly read data onto the end of existing vectors.

\-EXTEND is incompatible with \OVERLAY.
\-EXTEND is incompatible with \APPEND.

Field counts are specified by an integer preceding the vector name. Field counts must be literal integers, that is, they *cannot* be scalar variables. For example:

```
READ FILE.DAT 3X 2Y Z
```

will read 6 numbers from each record, placing the first 3 numbers into X[i], X[$i+1$], X[$i+2$], the next 2 numbers into Y[i], Y[$i+1$], and the last number into Z[i]. This command is equivalent to:

```
READ FILE.DAT X X X Y Y Z
```

Records beginning with an exclamation mark, !, are considered to be comments and are ignored.

By default, \ERRSTOP, an invalid field stops the read, but the data that has been read up to the error is saved. If the \ERRSKIP qualifier is used, an invalid field causes the entire record to be skipped. If the \ERRFILL qualifier is used, an invalid field causes the entire record to be filled with the value of ERRFILL if a format was entered, or only the invalid field will be set

to ERRFILL if no format was entered. By default, ERRFILL $= 0$, but it's value can be changed with the SET command and it's value obtained with the GET command.

Example 1

```
                        1  2  3  4  5  6
FILE.DAT                7  8  9 10 11 12
                       13 14 15 16 17 18
```

command	*result*
READ FILE.DAT 3X 2Y Z	X = [1;2;3;7;8;9;13;14;15] Y = [4;5;10;11;16;17] Z = [6;12;18]
READ FILE.DAT 3X	X = [1; 2; 3; 7; 8; 9; 13; 14; 15]
READ DUM.DAT 6X	X = [1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;16;17;18]

Example 2

```
                    1  23.7    0.1000E-5
                    2 -31.4    0.2000E-3
                    3   9.09   0.3000E-1
DUM.DAT             4  10.001  0.4000E+1
                    5  -2.0    0.5000E+2
                    6  30.2    0.6000E+3
```

command	*result*
READ DUM.DAT X Y	X = [1;2;3;4;5;6] Y = [23.7;-31.4;9.09;10.001;-2.0;30.2]
READ DUM.DAT\3 X Y	X = [3;4;5;6] Y = [9.09;10.001;-2.0;30.2]
READ DUM.DAT\[2:4] X\3 Y	X = [.0002;.03;4.0] Y = [-31.4;9.09;10.001]
READ\FORMAT DUM.DAT\[1:5:2] (2X,F8.4,E9.4) X Y	X = [23.7;9.09;-2.0] Y = [.000001;.03;50.0]

Example 3

Sample data file: DUM.DAT	This script reads data into vectors and allows the user to choose when to stop reading.

```
10 20 30
40 50 60
70 80 90
this is a test
100 200 300
400 500 600
700 800 900
test line two
-10 -20 -30
-40 -50 -60
-70 -80 -90
```

```
LNUM = 1
START:
READ DUM.DAT\LNUM X Y Z
IF ~EXIST('X') THEN GOTO DONE  ! end of file
LIST X Y Z
ANS='Y'
INQUIRE 'read again ? (Y|n)' ANS
IF NES(UCASE(ANS),'Y') THEN GOTO DONE
LNUM = LEN(X)+LNUM+1
GOTO START
DONE:
```

Unformatted binary files

Syntax	`READ\UNFORMATTED file{\line_range} (frmt) x1 { x2 ... }`
Qualifiers	`\CONTINUE, \CLOSE, \APPEND, \OVERLAY, \EXTEND, \MESSAGES`
Defaults	`\-CONTINUE, \-CLOSE, \-APPEND, \-OVERLAY, \EXTEND, \MESSAGES`

If the `\UNFORMATTED` qualifier is used, the file is assumed to be an unformatted binary file. The two methods of reading data from unformatted binary files, by record or stream, are indicated by the prescription paramter, `(frmt)`, which is required when the `\UNFORMATTED` qualifier is used. The `frmt` prescription must be enclosed in parenthesis, (and). For example:

```
real*4 x(100), y(100)
do i = 1, 100
  write(unit)x(i),y(i)          ! written by record   ( 100 records )
end do
write(unit)x,y                  ! written as a stream ( 1 record )
write(unit)(x(i),y(i),i=1,100)  ! written as a stream ( 1 record )
```

The qualifiers `\ERRSTOP`, `\ERRSKIP`, and `\ERRFILL` cannot be used with the `\UNFORMATTED` qualifier.

Reading by record

The `frmt` codes: `R4 R8 I1 I2 I4 L1 L4 Xn`
specify that the data is to be read by record, and indicate the data type and the number of bytes for each value in each record. The `Xn` code specifies skipping over `n` bytes. All values are converted to `REAL*8` for internal storage.

For example: `READ\UNFORMATTED FILE.DAT (3R4,X8,I2) X Y Z A`
indicates 22 bytes per record: `REAL*4, REAL*4, REAL*4`, skip 8 bytes, `INTEGER*2`.

A vector appended to the file name, `file\x`, specifies from which records to read data. The first $x[1]-1$ records will be skipped. The data will be read from records $x[1], x[2], \ldots, x[\#]$. Records $x[i]+1$ to $x[i+1]-1$ will be skipped. The vector `x` must be monotonically increasing.

To read different numbers of elements into vectors with a single `READ` command, use the `\NOEXTEND`, or `\-EXTEND`, qualifier. The output length of a vector will be number of values that are read, to a maximum of that vector's original length. For example, suppose that vector `X` has length 10 and vector `Y` has length 20, and suppose you enter:

```
READ\UNFORMATTED\-EXTEND file\[1:20] (2R4) X Y
```

If 20 records are read, vector `X` will be made with a length of 10 and vector `Y` will be made with a length of 20. If only 15 records are read, vector `X` will have length 10 but vector `Y` will only have length 15.

By default, a new vector is created to hold the newly read data. If the `\OVERLAY` qualifier is used, an existing vector will have the newly read data overlayed on the original data. The resultant vector length may be longer, but never shorter. Use the `\APPEND` qualifier to append the newly read data onto the end of existing vectors.

`\-EXTEND` is incompatible with `\OVERLAY`.
`\-EXTEND` is incompatible with `\APPEND`.

Field counts must be literal integers, that is, they *cannot* be scalar variables. Field counts are specified by an integer preceding the vector name.

For example: `READ\UNFORMATTED FILE.DAT (6R8) 3X 2Y Z`
will read 6 numbers from each record, placing the first 3 numbers into $X[i], X[i+1], X[i+2]$, the next 2 numbers into $Y[i], Y[i+1]$, and the last number into $Z[i]$. This command is equivalent to: `READ\UNFORMATTED FILE.DAT (6R8) X X X Y Y Z`

Stream reading

The `frmt` codes: `1B 2B 4B 8B`

specify that the data is to be read as a stream, and indicate the data type only. All values are converted to REAL*8 for internal storage. The number of values to read is indicated by creating the vectors before issuing the READ command. The number of values to read into a vector will be exactly the current length of that vector.

For example:

```
VECTOR X 100
VECTOR Y 200
READ\UNFORMATTED FILE.DAT (4B) X Y
```

indicate that 100 REAL*4 values are to be read into X and 200 REAL*4 values into Y.

A scalar appended to the file name, file\n, specifies the starting record. The first n − 1 records will be skipped.

Restrictions:
- vector line ranges *cannot* be used
- field counts *cannot* be used
- \-EXTEND is always in effect
- \OVERLAY *cannot* be used
- \APPEND *cannot* be used

Example 1

Suppose you have written some data using the code fragment:

```
REAL*8 X(10), Y(10), Z(10)
INTEGER*2 NUM
...
OPEN(UNIT=20,FILE='DUM.DAT',FORM='UNFORMATTED')
NUM = 10
WRITE(20)NUM          ! first record contains length of arrays
WRITE(20)X,Y,Z        ! stream write the arrays, this writes all of X
...                   ! followed by all of Y, then all of Z
```

You could read this data using the commands:

```
READ\UNFORMATTED\SCALARS DUM.DAT (I2) N  ! get length of vectors
VECTOR X Y Z N                           ! create vectors with length N
READ\UNFORMATTED DUM.DAT\2 (8B) X Y Z    ! stream read vectors
```

Example 2

Suppose you have written some data using the code fragment:

```
REAL*8 X(10), Y(10), Z(10)
...
OPEN(UNIT=20,FILE='DUM.DAT',FORM='UNFORMATTED')
NUM = 10
WRITE(20)NUM                     ! first record contains array length
WRITE(20)(X(I),Y(I),Z(I),I=1,10) ! stream write the arrays,
...                              ! but mix the arrays
```

You could read this data using the commands:

```
READ\UNFORMATTED\SCALARS DUM.DAT (I2) N      ! get length of vectors
READ\UNFORMATTED\MATRIX DUM.DAT\2 (8B) M 3 N ! stream read as a matrix
X = M[1,*]                                   ! extract first row
Y = M[2,*]                                   ! extract second row
Z = M[3,*]                                   ! extract third row
```

Example 3

Suppose you have written some data using the code fragment:

```
REAL*8 X(10), Y(10), Z(10)
...
OPEN(UNIT=20,FILE='DUM.DAT',FORM='UNFORMATTED')
DO I = 1, 10
  WRITE(20)X(I),Y(I),Z(I)        ! write the arrays by record
END DO
...
```

You could read this data using the command:

```
READ\UNFORMATTED DUM.DAT (3R8) X Y Z    ! read by record
```

Example 4

Suppose you have written some data using the code fragment:

```
REAL*8 X(10), Y(10), Z(10)
INTEGER*2 NUM
...
OPEN(UNIT=20,FILE='DUM.DAT',FORM='UNFORMATTED')
NUM = 10
WRITE(20)NUM          ! first record contains length of arrays
WRITE(20)X            ! stream write the X array
WRITE(20)Y            ! stream write the Y array
WRITE(20)Z            ! stream write the Z array
...
```

You could read this data using the commands:

```
READ\UNFORMATTED\SCALARS\CONTINUE DUM.DAT (I2) N ! don't close file
VECTOR X Y Z N                               ! create vectors of length N
READ\UNFORMATTED\CONTINUE DUM.DAT (8B) X  ! stream read
READ\UNFORMATTED\CONTINUE DUM.DAT (8B) Y  ! stream read
READ\UNFORMATTED\CONTINUE DUM.DAT (8B) Z  ! stream read
```

Example 5

To read the unformatted binary file DUM.DAT, bytes $5-8$ into X, bytes $9-12$ into Y, bytes $13-16$ into Z, enter the command:

```
READ\UNFORMATTED DUM.DAT (X4,3R4) X Y Z
```

To read the same file, but putting all the data into the vector Y, that is, bytes $5-8$ into Y$[i]$, bytes $9-12$ into Y$[i+1]$, bytes $13-16$ into Y$[i+2]$, where $i=[1;4;7;\ldots]$, enter the command:

```
READ\UNFORMATTED DUM.DAT (X4,3R4) 3Y
```

To start reading at the end of the second record of the file and to read THETA (INTEGER*4) from bytes $1-4$, and PHI (REAL*8) from bytes $5-12$, enter the command

```
READ\UNFORMATTED DUM.DAT\2 (I4,R8) THETA PHI
```

Reading data into scalars

The READ\SCALARS command reads scalar numbers from one record of a file. By default, the first record is read from an ASCII file, and, if no errors are encountered on the read, the I$_{th}$ number is placed into scalar sI. New scalar variables are created. By default, no scalars will

be made if an invalid field is encountered on the read. A scalar appended to the file name, file\n, specifies the starting record. The first $n-1$ records will be skipped.

ASCII files

Syntax	READ\SCALARS file{\n} s1{\c1} { s2{\c2} ... } READ\SCALARS\FORMAT file{\n} (frmt) s1 { s2 ... }
Qualifiers	\FORMAT, \CONTINUE, \CLOSE, \MESSAGES
Defaults	\-FORMAT, \-CONTINUE, \-CLOSE, \MESSAGES

By default, or if the \ASCII qualifier is used, the file is assumed to be an ASCII file and the first record is read. Specify which record to read by appending a scalar to the file name as a qualifier, file\n. By default, no scalars will be created if an invalid field is encountered while reading.

By default, the I$_{th}$ number field is placed into scalar sI. The field number can be specified by appending a scalar, cI, to the scalar name as a qualifier. In this case, the cI$_{th}$ field can be placed into the sI scalar.

For example, after the command: READ\SCALARS DUM.DAT W\2 X\4 Y Z\1
W would contain field 2, X would contain field 4, Y would default to field 3, and Z would contain field 1.

The \FORMAT qualifier must be used to indicate that a format is present. The format must be enclosed in parentheses, (and). If a format is used, field numbers *cannot* be specified.

Standard FORTRAN formats are valid, but only REAL variables can be read, so do not use INTEGER, LOGICAL or CHARACTER formats. All values are converted to REAL*8 for internal storage.

If the \ERRFILL qualifier is used, an invalid field causes the invalid field to be set to ERRFILL. By default, ERRFILL $=0$, but it's value can be changed with the SET command, and obtained with the GET command. \ERRFILL cannot be used with a format.

Example

```
DUM.DAT      1  23.7   0.1000E-5
             2 -31.4   0.2000E-3
             3   9.09  0.3000E-1
```

command	*result*	
READ\SCALARS DUM.DAT A B	A = 1	B = 23.7
READ\SCALARS DUM.DAT A\3 B	A = .000001	B = 23.7
READ\SCALARS DUM.DAT\3 A B	A = 3	B = 9.09
READ\SCALARS DUM.DAT\2 A\3 B	A = .0002	B = −31.4
READ\SCALARS\FORMAT DUM.DAT (2X,F8.4,E9.4) A B	A = 23.7	B = .000001

Unformatted binary files

Syntax	`READ\SCALARS\UNFORMATTED file{\n} (frmt) s1 { s2 ... }`
Qualifiers	`\CONTINUE, \CLOSE, \MESSAGES`
Defaults	`\-CONTINUE, \-CLOSE, \MESSAGES`

If the `\UNFORMATTED` qualifier is used, the file is assumed to be an unformatted binary file. By default, the first record is read. Specify which record to read by appending a scalar to the file name as a qualifier, `file\n`. No scalars will be created if an invalid field is encountered while reading. The `(frmt)` paramter is a prescription that specifies how the record is to be read and is required when the `\UNFORMATTED` qualifier is used. The prescription must be enclosed in parenthesis, (and).

The `frmt` codes: `R4 R8 I1 I2 I4 L1 L4 Xn` indicate the data type and the number of bytes for each value in the record. The `Xn` code is used to skip over n bytes. All values are converted to `REAL*8` for internal storage.

Examples

```
READ\SCALARS\UNFORMATTED DUM.DAT (3R4,X8,I2) A B C D
```

reads 22 bytes from the first record: `REAL*4`, `REAL*4`, `REAL*4`, skip 8 bytes, `INTEGER*2` and creates 4 scalars from these numbers.

```
READ\SCALARS\UNFORMATTED DUM.DAT (X10,R8) A
```

creates scalar `A` from the `REAL*8` number in bytes $11-18$ of the first record.

Reading data into a matrix

By default, a two dimensional array is read by records, starting with the first record, from an ASCII file, in free format, where `nrows` is the first dimension, the number of rows, and `ncols` is the second dimension, the number of columns. The first dimension *must* be entered

exactly. If the second dimension is not known, do not enter a value for `ncols`, and the read will continue until the end of file. The actual second dimension will be displayed when the read operation is done. A new matrix variable will be made. No matrix will be made if an error is encountered on the read.

A scalar appended to the file name, `file\n`, specifies the starting record. The first $n-1$ records will be skipped.

ASCII files

Syntax	`READ\MATRIX file{\n} m nrows { ncols }`
	`READ\MATRIX\FORMAT file{\n} (frmt) m nrows { ncols }`
Qualifiers	`\FORMAT, \CONTINUE, \CLOSE, \FLIPPED, \MESSAGES`
Defaults	`\-FORMAT, \-CONTINUE, \-CLOSE, \FLIPPED, \MESSAGES`

By default, or if the `\ASCII` qualifier is used, the file is assumed to be an ASCII file and is read by records. A scalar appended to the file name as a qualifier, `file\n`, specifies the starting record. The file will be read from the n_{th} record to the end of file.

The `\FORMAT` qualifier must be used to indicate that a format is present. The format must be enclosed in parentheses, (and).

Standard FORTRAN formats are valid, but only `REAL` variables can be read, so do not use `INTEGER`, `LOGICAL` or `CHARACTER` formats. All values are converted to `REAL*8` for internal storage.

Flipped

Suppose you have a matrix `M` which has 3 rows and 4 columns. When you enter `WRITE\MATRIX file M` the rows of the matrix are written into records of the file. There will be 3 records, each containing 4 numbers. But, if you then entered `READ\MATRIX file M 3 4` the input matrix would be scrambled, because `READ` puts the first record into the first column, the second record into the second column, and so on. Thus, the matrix is transposed, or flipped. To read it in properly, you would have to enter `READ\MATRIX file M 4 3` and then take the transpose, `M = <-M`. So, there is a qualifier, `\-FLIPPED`, which has the syntax:

```
READ\MATRIX\-FLIPPED file matrix ncolumns nrows
```

Note: In the default, without the `\-FLIPPED` qualifier, the syntax remains: `READ\MATRIX file matrix nrows ncol`

Example 1

the code fragment — creates the file

```
    REAL*4 M(7,5)
    DO J = 1, 5
      DO I = 1, 7
        M(I,J) = 10*I+J
      END DO
    END DO
    WRITE(1,10)M
 10 FORMAT(3F4.0)
```

```
11. 21. 31.
41. 51. 61.
71. 12. 22.
32. 42. 52.
62. 72. 13.
23. 33. 43.
53. 63. 73.
14. 24. 34.
44. 54. 64.
74. 15. 25.
35. 45. 55.
65. 75.
```

command — *result*

```
READ\MATRIX\FORMAT file (3F4.0) M 7 5
```

$$
M = \begin{pmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \\ 61 & 62 & 63 & 64 & 65 \\ 71 & 72 & 73 & 74 & 75 \end{pmatrix}
$$

Example 2

the code fragment

```
    REAL*4 M(7,5)
    DO J = 1, 5
      DO I = 1, 7
        M(I,J) = 10*I+J
      END DO
    END DO
    WRITE(1,10)M
 10 FORMAT(14F4.0)
```

creates the file

```
11. 21. 31. 41. 51. 61. 71. 12. 22. 32. 42. 52. 62. 72.
13. 23. 33. 43. 53. 63. 73. 14. 24. 34. 44. 54. 64. 74.
15. 25. 35. 45. 55. 65. 75.
```


command — *result*

```
READ\MATRIX\FORMAT file (14F4.0) M 7 5
```

$$
M = \begin{pmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \\ 61 & 62 & 63 & 64 & 65 \\ 71 & 72 & 73 & 74 & 75 \end{pmatrix}
$$

Example 3

the code fragment — creates the file

```
    REAL*4 M(7,5)
    DO J = 1, 5
      DO I = 1, 7
        M(I,J) = 10*I+J
      END DO
    END DO
    DO J = 1, 5
      WRITE(1,10)(M(I,J),I=1,7)
    END DO
 10 FORMAT(4F4.0)
```

```
11. 21. 31. 41.
51. 61. 71.
12. 22. 32. 42.
52. 62. 72.
13. 23. 33. 43.
53. 63. 73.
14. 24. 34. 44.
54. 64. 74.
15. 25. 35. 45.
55. 65. 75.
```

command — *result*

```
READ\MATRIX file M 7 5
```

$$
M = \begin{pmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \\ 61 & 62 & 63 & 64 & 65 \\ 71 & 72 & 73 & 74 & 75 \end{pmatrix}
$$

Example 4

the code fragment / creates the file

```
    REAL*8 M(7,5)
    DO J = 1, 5
      DO I = 1, 7
        M(I,J) = 10*I+J
      END DO
    END DO
    WRITE(1,10)((M(I,J),J=1,5),I=1,7)
 10 FORMAT(5F5.0)
```

```
11.  12.  13.  14.  15.
21.  22.  23.  24.  25.
31.  32.  33.  34.  35.
41.  42.  43.  44.  45.
51.  52.  53.  54.  55.
61.  62.  63.  64.  65.
71.  72.  73.  74.  75.
```

command / *result*

`READ\MATRIX file M 5`

$$M = \begin{pmatrix} 11 & 21 & 31 & 41 & 51 & 61 & 71 \\ 12 & 22 & 32 & 42 & 52 & 62 & 72 \\ 13 & 23 & 33 & 43 & 53 & 63 & 73 \\ 14 & 24 & 34 & 44 & 54 & 64 & 74 \\ 15 & 25 & 35 & 45 & 55 & 65 & 75 \end{pmatrix}$$

Example 5

the code fragment / creates the file

```
    REAL*8 M(5,8)
    DO J = 1, 8
      DO I = 1, 5
        M(I,J) = 10*I+J
      END DO
    END DO
    WRITE(1,10)((M(I,J),I=1,5),J=1,8)
 10 FORMAT(5F5.0)
```

```
11.  21.  31.  41.  51.
12.  22.  32.  42.  52.
13.  23.  33.  43.  53.
14.  24.  34.  44.  54.
15.  25.  35.  45.  55.
16.  26.  36.  46.  56.
17.  27.  37.  47.  57.
18.  28.  38.  48.  58.
```


command / *result*

`READ\MATRIX file M 5`

$$M = \begin{pmatrix} 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \\ 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 \\ 31 & 32 & 33 & 34 & 35 & 36 & 37 & 38 \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48 \\ 51 & 52 & 53 & 54 & 55 & 56 & 57 & 58 \end{pmatrix}$$

Example 6

the code fragment / creates the file

```
    REAL*8 M(5,8)
    DO J = 1, 8
      DO I = 1, 5
        M(I,J) = 10*I+J
      END DO
    END DO
    WRITE(1,10)((M(I,J),J=1,8),I=1,5)
 10 FORMAT(4F5.0)
```

```
11.  12.  13.  14.
15.  16.  17.  18.
21.  22.  23.  24.
25.  26.  27.  28.
31.  32.  33.  34.
35.  36.  37.  38.
41.  42.  43.  44.
45.  46.  47.  48.
51.  52.  53.  54.
55.  56.  57.  58.
```

command / *result*

`READ\MATRIX file M 4`

$$M = \begin{pmatrix} 11 & 15 & 21 & 25 & 31 & 35 & 41 & 45 & 51 & 55 \\ 12 & 16 & 22 & 26 & 32 & 36 & 42 & 46 & 52 & 56 \\ 13 & 17 & 23 & 27 & 33 & 37 & 43 & 47 & 53 & 57 \\ 14 & 18 & 24 & 28 & 34 & 38 & 44 & 48 & 54 & 58 \end{pmatrix}$$

command / *result*

`READ\MATRIX file M 8`

$$M = \begin{pmatrix} 11 & 21 & 31 & 41 & 51 \\ 12 & 22 & 32 & 42 & 52 \\ 13 & 23 & 33 & 43 & 53 \\ 14 & 24 & 34 & 44 & 54 \\ 15 & 25 & 35 & 45 & 55 \\ 16 & 26 & 36 & 46 & 56 \\ 17 & 27 & 37 & 47 & 57 \\ 18 & 28 & 38 & 48 & 58 \end{pmatrix}$$

Example 7

Suppose you have the file DUM.DAT

```
Header line 1
Header line 2
1.0 2.0 3.0 4.0 5.0 6.0 7.0
1.1 2.1 3.1 4.1 5.1 6.1 7.1
1.2 2.2 3.2 4.2 5.2 6.2 7.2
1.3 2.3 3.3 4.3 5.3 6.3 7.3
1.4 2.4 3.4 4.4 5.4 6.4 7.4
```

command	*result*
READ\MATRIX DUM.DAT\3 M 7	$M = \begin{pmatrix} 1.0 & 1.1 & 1.2 & 1.3 & 1.4 \\ 2.0 & 2.1 & 2.2 & 2.3 & 2.4 \\ 3.0 & 3.1 & 3.2 & 3.3 & 3.4 \\ 4.0 & 4.1 & 4.2 & 4.3 & 4.4 \\ 5.0 & 5.1 & 5.2 & 5.3 & 5.4 \\ 6.0 & 6.1 & 6.2 & 6.3 & 6.4 \\ 7.0 & 7.1 & 7.2 & 7.3 & 7.4 \end{pmatrix}$

Unformatted binary files

Syntax READ\MATRIX\UNFORMATTED file{\n} (frmt) m nrows { ncols }
Qualifiers \CONTINUE, \CLOSE, \MESSAGES
Defaults \-CONTINUE, \-CLOSE, \MESSAGES

If the \UNFORMATTED qualifier is used, the file is assumed to be an unformatted binary file. The two methods of reading data from unformatted binary files, by record or stream, are indicated by the prescription, (frmt), paramter, which is required whith the \UNFORMATTED qualifier. A scalar appended to the file name as a qualifier, file\n, specifies the starting record. The frmt prescription must be enclosed in parenthesis, (and).

For example:

```
real*8 x(100,10)
do j = 1, 10
  write(unit)(x(i,j),i=1,100)       ! write 10 records
end do
write(unit)x                        ! stream write
write(unit)(x(i,j),i=1,100),j=1,10) ! stream write
```


Reading by record

Syntax READ\MATRIX\UNFORMATTED file{\n} (frmt) m nrows { ncols }

The frmt codes: R4 R8 I1 I2 I4 L1 L4 Xn
specify that the data is to be read by record, and indicate the data type and the number of bytes for each value in each record. The Xn code specifies skipping over n bytes. For example, (20R4) indicate 80 bytes per record: 20 REAL*4 values. All values are converted to REAL*8 for internal storage. If the number of columns, ncols, is not entered, records will be read until an end of file is reached.

Stream reading

Syntax READ\MATRIX\UNFORMATTED file{\n} (frmt) m nrows ncols

The frmt codes: 1B 2B 4B 8B
specify that the data is to be read as a stream, and indicate the data type only. The number of values to read is indicated by the number of rows, nrows, and the number of columns, ncols, *both* of which are required. All values are converted to REAL*8 for internal storage.

For example: READ\MATRIX\UNFORMATTED DUM.DAT (4B) M 10 100 indicate that 1000 REAL*4 values are to be stream read into matrix M with 10 rows and 100 columns.

Example 1

Consider the code fragment:

```
REAL*8 X(7,5)
INTEGER*2 NR, NC
DATA NR /7/, NC /5/
DO J = 1, 5
  DO I = 1, 7
    X(I,J) = 10*I+J
  END DO
END DO
OPEN(UNIT=20,FILE='DUM.DAT',FORM='UNFORMATTED')
WRITE(20)NR,NC        ! first record contains dimensions
WRITE(20)X            ! stream write the array
```

commands	*result*
`READ\UNFORM\SCALARS DUM.DAT (2I2) NR NC` `READ\UNFORM\MATRIX DUM.DAT\2 (8B) M NR NC`	$M = \begin{pmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \\ 61 & 62 & 63 & 64 & 65 \\ 71 & 72 & 73 & 74 & 75 \end{pmatrix}$

Example 2

Consider the code fragment:

```
REAL*8 X(7,5)
INTEGER*2 NR
DATA NR /7/
DO J = 1, 5
  DO I = 1, 7
    X(I,J) = 10*I+J
  END DO
END DO
OPEN(UNIT=20,FILE='DUM.DAT',FORM='UNFORMATTED')
WRITE(20)NR       ! first dimension, number of rows
DO J = 1, 5
  WRITE(20)(X(I,J),I=1,7)  ! write by records ( 5 records )
END DO
```

commands	*result*
`READ\UNFORM\SCALARS DUM.DAT (I2) NR` `FMT = '('//RCHAR(NR)//'R8)'` `READ\UNFORM\MATRIX DUM.DAT\2 FMT M NR`	$M = \begin{pmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \\ 61 & 62 & 63 & 64 & 65 \\ 71 & 72 & 73 & 74 & 75 \end{pmatrix}$

Reading data into a string variable

By default, the `READ\TEXT` command reads the first record from an ASCII file as a string, making a string variable. The maximum line length that can be read is 255 characters. If a single line is read, a string variable will be made. If multiple lines are read, an array string

variable will be made.

A scalar appended to the file name, `file\n`, specifies the record to read as data. The first $n-1$ records will be skipped. A string variable will be made.

A vector appended to the file name, `file\x`, specifies from which records to read data. The first $x[1]-1$ records will be skipped. The data will be read from records $x[1], x[2], \ldots, x[\#]$. Records $x[i]+1$ to $x[i+1]-1$ will be skipped. The vector `x` must be monotonically increasing. Multiple lines will be read only if multiple line numbers are indicated. a string array variable will be made.

ASCII files

Syntax	`READ\TEXT file{\line_range} txtvar` `READ\TEXT\FORMAT file{\line_range} (frmt) txtvar`
Qualifiers	`\FORMAT, \CONTINUE, \CLOSE, \MESSAGES`
Defaults	`\-FORMAT, \-CONTINUE, \-CLOSE, \MESSAGES`

By default, when no format is entered, the entire line, up to 255 characters is read. The `\FORMAT` qualifier must be used to indicate that a format is present. The format must be enclosed in parentheses, (and).

The `frmt` codes:	`nX`	means to skip over n characters
	`An`	means to read n characters

For example: `(2X,A10)` means to skip the first two characters and read the next ten.

Examples

command	*result*
`READ\TEXT DUM.DAT\4 TXT`	reads the fourth record into a scalar string variable
`READ\TEXT DUM.DAT\[4:8] TXT`	reads 5 strings from records 4 to 8 into a string array variable

Unformatted binary files

Syntax	`READ\TEXT\UNFORMATTED file{\line_range} (frmt) txtvar`
Qualifiers	`\CONTINUE, \CLOSE, \MESSAGES`
Defaults	`\-CONTINUE, \-CLOSE, \MESSAGES`

If the `\UNFORMATTED` qualifier is used, the file is assumed to be an unformatted binary file. By default, the first record is read. The `(frmt)` paramter is a prescription that specifies how the record is to be read and is required when the \UNFORMATTED qualifier is used. The prescription must be enclosed in parenthesis, (and). The only allowable prescription is (An), which means to read n characters. For example: (A10) means to read the first ten characters from the record.

REBIN

Syntax `REBIN v vout n`
`REBIN m mout nr nc`

The REBIN command rebins the data in either:

the vector v by the compression factor n; or
the matrix m by the row compression factor nr and the column compression factor nc.

Rebinning vectors

Syntax `REBIN v vout n`

Suppose that the length of vector v is N then:

$$\mathtt{vout}[i] = \sum_{k=1}^{\mathtt{n}} \mathtt{v}[(i-1)\mathtt{n}+k] \quad \text{for} \quad i = [1 : \frac{N}{\mathtt{n}}],$$

that is, the length of vout will be $\frac{N}{\mathtt{n}}$, and

$$\mathtt{vout}[1] = \sum_{i=1}^{\mathtt{n}} \mathtt{v}[i]$$

$$\mathtt{vout}[2] = \sum_{i=\mathtt{n}+1}^{2\mathtt{n}} \mathtt{v}[i]$$

$$\cdots$$

$$\mathtt{vout}[\frac{N}{\mathtt{n}}] = \sum_{i=(\frac{N}{\mathtt{n}}-1)\mathtt{n}+1}^{\frac{N}{\mathtt{n}}\mathtt{n}} \mathtt{v}[i]$$

If $(\frac{N}{\mathtt{n}}) \times \mathtt{n}$ is not equal to N, then the last element of vout will be incomplete. For example, if $N = 10$ and $\mathtt{n} = 3$ then vout will have 3 elements:

```
vout[1] = v[1] + v[2] + v[3]
vout[2] = v[4] + v[5] + v[6]
vout[3] = v[7] + v[8] + v[9]
```


and v[10] will not be included in vout.

Examples

Suppose that vector X has 20 elements. After the command: `REBIN X XOUT 2`

```
 XOUT[1] = X[ 1] + X[ 2]
 XOUT[2] = X[ 3] + X[ 4]
 XOUT[3] = X[ 5] + X[ 6]
        ...
XOUT[10] = X[19] + X[20]
```

After the command: `REBIN [1:1000] X 3`

```
  X[1] = 1+2+3
  X[2] = 4+5+6
  X[3] = 7+8+9
      ...
X[333] = 997+998+999
```

A warning will be given that the length of [1:1000] is not evenly divisible by 3 and so the last bin is incomplete.

Rebinning matrices

Syntax `REBIN m mout nr nc`

Suppose that matrix m has N rows and M columns, then:

$$\mathtt{mout}[i,j] = \sum_{l=1}^{\mathtt{nc}} \sum_{k=1}^{\mathtt{nr}} \mathtt{m}[(i-1)\mathtt{nr}+k,(j-1)\mathtt{nc}+l] \quad \text{for} \quad i = [1 : \frac{N}{\mathtt{nr}}], j = [1 : \frac{M}{\mathtt{nc}}],$$

that is, the matrix mout will have $\frac{N}{\mathtt{nr}}$ rows and $\frac{M}{\mathtt{nc}}$ columns.

$$\mathtt{mout}[1,1] = \sum_{j=1}^{\mathtt{nc}} \sum_{i=1}^{\mathtt{nr}} \mathtt{m}[i,j]$$

$$\mathtt{mout}[1,2] = \sum_{j=\mathtt{nc}+1}^{2\mathtt{nc}} \sum_{i=1}^{\mathtt{nr}} \mathtt{m}[i,j]$$

$$\texttt{mout}[2,1] = \sum_{j=1}^{\texttt{nc}} \sum_{i=\texttt{nr}+1}^{2\texttt{nr}} \texttt{m}[i,j]$$

$$\texttt{mout}[2,2] = \sum_{j=\texttt{nc}+1}^{2\texttt{nc}} \sum_{i=\texttt{nr}+1}^{2\texttt{nr}} \texttt{m}[i,j]$$

$$\ldots$$

If $(\frac{N}{\texttt{nr}} \times \texttt{nr}$ is not equal to N, then the last row of `mout` will be incomplete. If $(\frac{M}{\texttt{nc}} \times \texttt{nc}$ is not equal to M, then the last column of `mout` will be incomplete.

Example

Suppose that `M` is a matrix:

$$\texttt{M} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 6 & 8 & 10 & 12 \\ 3 & 6 & 9 & 12 & 15 & 18 \\ 4 & 8 & 12 & 16 & 20 & 24 \\ 5 & 10 & 15 & 20 & 25 & 30 \\ 6 & 12 & 18 & 24 & 30 & 36 \\ 7 & 14 & 21 & 28 & 35 & 42 \\ 8 & 16 & 24 & 32 & 40 & 48 \end{pmatrix}$$

Then after the command: `REBIN M MOUT 2 3`

$$\texttt{MOUT} = \begin{pmatrix} 18 & 45 \\ 42 & 105 \\ 66 & 165 \\ 90 & 225 \end{pmatrix}$$

Note that:

$$\texttt{MOUT}[1,1] = 1+2+3+2+4+6$$

$$\texttt{MOUT}[1,2] = 4+5+6+8+10+12$$

$$\texttt{MOUT}[2,1] = 3+6+9+4+8+12$$

$$\texttt{MOUT}[2,2] = 12+15+18+16+20+24, \quad \text{etc.}$$

REFRESH

Syntax `REFRESH`

The `REFRESH` command is relevant only when using an X Window monitor. The X Window graphics window, and the zoom window, if it exists, will be redrawn. This has no affect on a graphics hardcopy.

RENAME

Syntax `RENAME oldname newname`
`RENAME old* new*`
`RENAME *old *new`

The `RENAME` command renames variables. The simplest case is to rename one variable. For example:

```
RENAME XOLD XX   ! renames the variable XOLD to XX
```

If the wildcard * is the last character in the old name, all variables beginning with the preceding characters will be renamed. For example:

```
RENAME X1* XX*   ! renames X1 to XX, X1X to XXX, and X1Y to XXY
```

If the wildcard is the first character in the old name, all variables that end with the succeeding characters will be renamed. For example:

```
RENAME *X *Y2    ! renames X to Y2, AX to AY2, XX to XY2
```

REPLOT

Syntax `REPLOT { nw }`
`REPLOT\ALLWINDOWS`

Qualifiers `\AXES, \ALLWINDOWS, \TEXT`

Defaults `\AXES`, `nw` = current window number, `\-TEXT`

By default, information pertaining to curves produced by the `GRAPH` command, as well as an automatically plotted curve from the `ELLIPSE` command, are saved in replot buffers. When a set of axes is drawn and then data is overlayed on those axes, all of the data may not appear within the axis boundaries. You can use the `REPLOT` command to have all of the data appear on a single autoscaled graph. The window number, `nw`, defaults to the current window number. When the `\NOAXES` qualifier is used, the data will be replotted but the graph axes will not be drawn.

What is saved

For each data set, the window number in which it was drawn is saved, along with the plotting symbols, including their types, colours, connectivity, angles, and sizes are saved for replotting. Also saved are: error bar types, line thicknesses, line types, colours, and histogram types.

Strings

Strings that were drawn with the TEXT command will also be replotted. If the \GRAPH qualifier is used with the TEXT command, the location of the replotted text will remain fixed with respect to the graph's coordinate system. Otherwise, the location of the replotted text will remain fixed with respect to the window.

The \-TEXT qualifier means to replot the data curves but not any strings that had also been drawn using the TEXT command. The default is \TEXT, which means to also replot any such strings.

Enable/Disable

After the DISABLE REPLOT command is entered, subsequent data curves will not be saved in the replot buffers. Use the ENABLE REPLOT command to re-enable saving. If replotting is enabled, it can be disabled for individual commands by use of the \NOREPLOT qualifier, for example, GRAPH\NOREPLOT.

Windows

If the window number nw is entered, the current window becomes window nw. Only the data drawn into window nw will be replotted.

If the REPLOT command is entered without a window number, and without the \ALLWINDOWS qualifier, the graphics will be automatically cleared. If using a window number or \ALLWINDOWS, it is up to the user to ensure that the appropriate windows are cleared before entering the REPLOT command.

Clearing the graphics

Use the CLEAR\NOREPLOT command to clear all graphics without emptying the replot buffers.

Use the ERASEWINDOW, page 76, command to erase the graphics from a window. This only applies to the terminal monitor, to PostScript graphics, and to bitmap graphics.

Use the CLEAR\REPLOTONLY command to clear the replot buffers without clearing the graphics.

Redraw all windows

Use the REPLOT\ALLWINDOWS command to replot all windows that have some data stored in the replot buffers. It is up to the user to ensure that the appropriate windows are cleared before entering the REPLOT\ALLWINDOWS command.

Examples

The following sequence of commands draws a graph of 20*X versus 10*X and overlays a graph of (X^3)/10 versus X^2 on the same set of axes. To see the complete drawing, the REPLOT command is used.

```
X=[1:20]                  ! create some data
WINDOW 5                  ! choose graphics window
SET PCHAR -1              ! set plotting symbol
GRAPH 20*X 10*X           ! plot 10*X vs 20*X with axes
SET PCHAR -2              ! choose different plotting symbol
GRAPH\NOAXES X^2 (X^3)/10 ! overlay (X^3)/10 vs X^2 on same axes
CLEAR\NOREPLOT            ! clear graphics but not replot buffers
REPLOT                    ! replot all data on common scale
```

Suppose that you have plotted a graph of vectors Y1 vs. X1 in window number 5; a graph of vectors Y2 vs. X2 in window number 7, and overlain a plot of Y3 vs. X3 also in window 7. To replot both windows, use the REPLOT\ALLWINDOWS command.

```
WINDOW 5                  ! choose window
GRAPH X1 Y1               ! plot data and axes
WINDOW 7                  ! choose window
GRAPH X2 Y2               ! plot data and axes
GRAPH\NOAXES X3 Y3        ! overlay a curve
CLEAR\NOREPLOT            ! clear graphics but not replot buffers
REPLOT\ALL                ! replot window 5 and window 7
```

RESIZE

Syntax RESIZE

The RESIZE command brings up the graphics cursor, allowing PHYSICA to know when the graphics window is resized by dragging a corner or side of the graphics window. Quit RESIZE by pressing the Q key while the cursor is in the graphics window.

Using the RESIZE command before changing the shape of the graphics window is not necessary under VMS or Digital Unix. Resizing should be done with the RESIZE command under Linux, Silicon Graphics IRIX, Sun SOLARIS, and IBM AIX or the PHYSICA window will not match the actual graphics window.

Use the REFRESH command to simply redraw the contents of the graphics window, which is sometimes needed after resizing or uncovering the graphics window.

RESTORE

Syntax	`RESTORE filename`
Qualifiers	`\PHYSICA, \FIOWA, \XFIOWA, \MUD, \MSR, \IMSR, \CHAOS, \YBOS, \HBOOK`
Defaults	`\PHYSICA`

The RESTORE command is used to restore specially formatted data sets:

previously saved PHYSICA sessions; FIOWA type data sets; μSR type data sets; μSR MUD type data sets; IμSR type data sets; CHAOS type data sets; YBOS type data sets, HBOOK type data sets.

Environment variables in file names

For UNIX users, it is now possible to use an environment variable in a file name, if the environment variable is preceeded by a $. For example,

```
setenv FILE mysession
physica
restore $FILE
```

The environment variable can be just the first part of the filename, for example,

```
setenv FILE my
physica
restore $FILEsession
```

PHYSICA sessions

Syntax	`RESTORE file`
Qualifier	`\TTBUFFERS`
Default	`\TTBUFFERS`

By default, the RESTORE command, or RESTORE\PHYSICA, restores a previously saved PHYSICA run. Normally it would be used immediately after entering PHYSICA to continue a previous session that was saved with the SAVE command, page 224. An attempt is made to restore

the plots that were active at the time the SAVE command was issued, similar to entering the REPLOT\ALLWINDOWS command. Since some graphics is not REPLOTable, such as contour plots and density plots, this does not always work as expected. The hardcopy device as chosen with the DEVICE command is restored, as well as the orientation.

Note: This command should be used with caution, since the current run will be irrevocably lost in the process of restoring a previous run.

The run time loaded subroutine or function information is not restored, but must be regenerated explicitly in the restored session with the LOAD command, page 147.

The input line recall buffers, that is, the dynamic buffer, the static buffer, and the keypad buffer, are restored also, by default. If you do not want to restore these buffers when resuming a PHYSICA session, use the \-TTBUFFERS qualifier.

FIOWA data sets

Syntax	`RESTORE\FIOWA filename`

The RESTORE\FIOWA command restores FIOWA type data files containing histograms and scatterplots. No special qualifiers are need to restore data sets made with the "big" version of FIOWA. When you restore a file (or map to shared memory), the following PHYSICA variables are created automatically:

variable name	*type*	*description*
DATA	vector	all the histogram and scatterplot data
NHIST	scalar	number of histograms
NSCAT	scalar	number of scatterplots
HTITLE	string	string containing all the histogram titles
HXLABEL	string	string containing all the histogram labels
STITLE	string	string containing all the scatterplot titles
SXLABEL	string	string containing all the scatterplot xlabels
SYLABEL	string	string containing all the scatterplot ylabels

Histograms

The following histogram related vectors are created automatically:

variable name	*description*
NBINS	number of bins
HSTART	starting index
HXLO	x minimum
HXINC	bin size
HYLO	y minimum
HYINC	y increment
HLO	underflow
HHI	overflow
STRTHT	starting index for title
STRTHX	starting index for xlabel
LENHT	length of title
LENHX	length of xlabel

Scatterplots

The following scatterplot related variables are created automatically:

variable name	*type*	*description*
NSBINX	vector	number of bins in x
NSBINY	vector	number of bins in y
ISCAT	vector	index of first bin
SXLO	vector	x minimum
SXINC	vector	x increment
SYLO	vector	y minimum
SYINC	vector	y increment
OUTSID	matrix	underflows and overflows
STRTST	vector	starting index for title
STRTSX	vector	starting index for xlabel
STRTSY	vector	starting index for ylabel
LENST	vector	length of title
LENSX	vector	length of xlabel
LENSY	vector	length of ylabel

The scatterplot overflows and underflows are stored in a matrix named OUTSID, which always has 8 rows, and is defined below for scatterplot number j.

x-low y-high OUTSID[4,j]	x-ok y-high OUTSID[3,j]	x-high y-high OUTSID[2,j]
x-low y-ok OUTSID[5,j]	x-ok y-ok	x-high y-ok OUTSID[1,j]
x-low y-low OUTSID[6,j]	x-ok y-low OUTSID[7,j]	x-high y-low OUTSID[8,j]

FIOWA examples

To draw histogram n:

```
label\xaxis htitle[strtht[n]:strtht[n]+lenht[n]-1]
scalar\dummy j
graph\hist loop(hxlo[n]+(j-0.5)*hxinc[n],j,1:nbins[n]) -
 data[hstart[n]:hstart[n]+nbins[n]-1]
```

To draw scatterplot n, using the diffusion type of density plot:

```
label\xaxis stitle[strtst[n]:strtst[n]+lenst[n]-1]
scalar\dummy j
x = loop(sxlo[n]+(j-0.5)*sxinc[n],j,1:nsbinx[n])
y = loop(sylo[n]+(j-0.5)*syinc[n],j,1:nsbiny[n])
m = fold(data[iscat[n]:iscat[n]+nsbinx[n]*nsbiny[n]-1],nsbinx[n])
density\profiles\diffusion x y <-m  ! <- is the transpose operator
```

XFIOWA data sets

Syntax RESTORE\XFIOWA filename

The RESTORE\XFIOWA command restores eXpanded FIOWA type data files containing histograms and scatterplots (as modified by Tom Huber from the original FIOWA). These data files are ASCII and so are transportable between platforms, unlike the original FIOWA files which are written as unformatted data.

No special qualifiers are need to restore data sets made with the “big” version of FIOWA. When you restore an XFIOWA file, you get the same PHYSICA variables as with a FIOWA file.

HBOOK data sets

Syntax	`RESTORE\HBOOK filename`
	`RESTORE\HBOOK\DIR filename directory`
Examples	`RESTORE\HBOOK FILE.DAT`
	`RESTORE\HBOOK\DIR FILE.DAT '/SUB1/SUB2'`

The `RESTORE\HBOOK` command restores HBOOK type data files containing histograms and $2d$ histograms (scatterplots). If the `\DIR` qualifier is used, the ZEBRA RZ directory within the file can be specified, using the absolute pathname in the ZEBRA syntax. When you restore a file, you get the following PHYSICA variables:

variable name	*type*	*description*
DATA	vector	all the histogram and scatterplot data
HERROR	vector	histogram and scatterplot errors
NHIST	scalar	number of histograms
NSCAT	scalar	number of scatterplots
HTITLE	string	string containing all the histogram titles
STITLE	string	string containing all the scatterplot titles

The `HERROR` vector contains the errror contents of all the histograms and scatterplots. Access this information in the same way as you access the /verb+DATA+ vector, for example, for histogram /verb+n+: error = `herror[hstart[n]:hstart[n]+nbins[n]-1]`.

Listing

Syntax	`RESTORE\HBOOK\LIST filename`
	`RESTORE\HBOOK\LIST\DIR filename directory`

If you use the `\LIST` qualifer, a listing of the contents of the data file will be displayed on your monitor screen. If the `\DIR` qualifier is used, the ZEBRA RZ directory within the file can be specified, using the absolute pathname in the ZEBRA syntax. The listing is produced by the `HLDIR` routine from the CERN library.

Ntuples

Syntax	`RESTORE\HBOOK\RWN filename id`
	`RESTORE\HBOOK\RWN\DIR filename directory id`

This command will restore the Row Wise Ntuple with identification number `id` from an HBOOK data file. If the `\DIR` qualifier is used, the ZEBRA RZ directory within the file can be entered, using the absolute pathname in the ZEBRA syntax. By default, a vector will be created for each variable in the Ntuple, where the names of these vectors will be the tag

Commands

names of the variables in the Ntuple.

For now, Column Wise Ntuples cannot be restored in PHYSICA.

If the `\MATRIX` qualifier is used, an array string variable, named `NTAGSn`, will be created containing the tag names for the variables in the Ntuple, and a matrix, named `NTUPLEn`, will be created containing the data, where `n` is the identification number `id`.

Histograms

The following histogram related vector variables will be created automatically:

variable name	*description*
ID1	histogram identifier
NBINS	number of bins
NOENT	number of entries
HMEAN	mean value
HSTD	standard deviation
HEQUIV	number of equivalent events
HSTART	starting index
HXLO	x minimum
HXINC	bin size
HLO	underflow
HHI	overflow
STRTHT	starting index for title
LENHT	length of title

`NOENT[j]` is the number of original entries in histogram `[j]`, including underflows and overflows.

`HLO[j]` is the sum of the weights for events below the histogram lower limit. If you did not use weights when filling the histogram, then `HLO` is the number of events below the histogram lower limit.

`HHI[j]` is the sum of the weights for events above the histogram upper limit. If you did not use weights when filling the histogram, then `HHI` is the number of events above the histogram upper limit.

The number of equivalent events returned in `HEQUIV` is based on the channel contents only. If w_i represents the contents of event i, then

$$\text{number of equivalent events} = \sum(w_i)^2/\sum(w_i^2)$$

This is what it is supposed to be, *but* it seems to be just the total number of entries in the histogram, which is the same as NOENT. To calculate this value inside PHYSICA, you could use:

```
y = data[hstart[n]:hstart[n]+nbins[n]-1]
! hsum will be the sum of the weights inside the histogram limits
statistics\-message y hsum\sum
! hsum2 will be the sum of the squares of the weights
statistics y*y hsum2\sum
hequiv[n] = hsum*hsum/hsum2
```

The standard deviation returned in HSTD is based on the channel contents only. If x_i and w_i represent the value and contents of event i, and $W = \sum(w_i)$, then

$$\begin{aligned} \text{expectation value } E(x) &= \sum(w_i x_i)/W \\ \text{mean value} &= E(x) \\ \text{central moment of order n, } M(n) &= E((x - E(x))^n) \\ \text{standard deviation} &= \sqrt{(M(2))} \end{aligned}$$

The mean value, or expectation value, returned in HMEAN is based on the channel contents only. If x_i and w_i represent the value and contents of event i, and $W = \sum(w_i)$, then

$$\text{mean value} = \sum(w_i x_i)/W$$

Scatterplots

The following $2d$ histogram related vector variables will be created automatically:

variable name	*type*	*description*
ID2	vector	scatterplot identifier
SNOENT	vector	number of entries
NSBINX	vector	number of bins in x
NSBINY	vector	number of bins in y
ISCAT	vector	index of first bin
SXLO	vector	x minimum
SXINC	vector	x increment
SYLO	vector	y minimum
SYINC	vector	y increment
OUTSID	matrix	underflows and overflows
STRTST	vector	starting index for title
LENST	vector	length of title

SNOENT[j] is the number of original entries in scatterplot [j], including underflows and overflows.

The overflows and underflows for scatterplot j are stored in a matrix named OUTSID, which always has 8 rows, and is defined below.

x-low y-high OUTSID[4,j]	x-ok y-high OUTSID[3,j]	x-high y-high OUTSID[2,j]
x-low y-ok OUTSID[5,j]	x-ok y-ok	x-high y-ok OUTSID[1,j]
x-low y-low OUTSID[6,j]	x-ok y-low OUTSID[7,j]	x-high y-low OUTSID[8,j]

HBOOK examples

To draw histogram n:

```
label\xaxis htitle[strtht[n]:strtht[n]+lenht[n]-1]
scalar\dummy j
graph\hist loop(hxlo[n]+(j-0.5)*hxinc[n],j,1:nbins[n]) -
 data[hstart[n]:hstart[n]+nbins[n]-1]
```

To draw scatterplot n, using the diffusion type of density plot:

```
label\xaxis stitle[strtst[n]:strtst[n]+lenst[n]-1]
scalar\dummy j
x = loop(sxlo[n]+(j-0.5)*sxinc[n],j,1:nsbinx[n])
y = loop(sylo[n]+(j-0.5)*syinc[n],j,1:nsbiny[n])
m = fold(data[iscat[n]:iscat[n]+nsbinx[n]*nsbiny[n]-1],nsbinx[n])
density\profiles\diffusion x y <-m  ! <- is the transpose operator
```

YBOS data sets

Syntax	RESTORE\YBOS filename
Qualifier	\DOTPLOT
Default	\NODOTPLOT

The RESTORE\YBOS command restores YBOS type data files containing histograms or dotplots. Use the \DOTPLOT qualifier to specify that you want to restore a file containing dotplots. By default, the file is assumed to contain histogram data.

Histograms

When you restore (or map to shared memory) a YBOS histogram file, for histogram number j, you get the following PHYSICA variables:

variable name	*type*	YBOS *structure name or description*
HRUNNO	scalar	YBS.I_RUNNO
HTITLE[j]	array string	YBS.HISTO_ST(j).DOT_NAME
HTEST[j]	array string	YBS.HISTO_ST(j).TEST_NAME
HDATA[j]	array string	YBS.HISTO_ST(j).DATA_NAME
HIST	vector	YBS.HISTO
HSTART[j]	vector	starting index for histo j
HNBIN[j]	vector	YBS.HISTO_ST(j).N_BINS
HLOW[j]	vector	YBS.HISTO_ST(j).XLOW
HHI[j]	vector	YBS.HISTO_ST(j).XHI
HBINW[j]	vector	(HHI[j]-HLOW[j])/HNBIN[j]
HUNDER[j]	vector	YBS.HISTO_ST(j).XUNDER
HOVER[j]	vector	YBS.HISTO_ST(j).XOVER
HSUM[j]	vector	sum of histo j
HMEAN[j]	vector	mean value of histo j
HSIGMA[j]	vector	sigma of histo j

Dotplots

Syntax RESTORE\YBOS\DOTPLOT file

When you restore (or map to shared memory) a YBOS dot plot file, for dot plot number j, you get the following PHYSICA variables:

variable name	*type*	YBOS *structure name or description*
DRUNNO	scalar	YBS.I_RUNNO
X_SYMBOL[j]	array string	YBS.DOT_EXTRA(j).X_SYMBOL
Y_SYMBOL[j]	array string	YBS.DOT_EXTRA(j).Y_SYMBOL
STEST_ALL[j]	array string	YBS.DOT_EXTRA(j).STEST_ALL
DOT_NAME[j]	array string	YBS.DOT_ST(j).DOT_NAME
SCTEST[j]	array string	YBS.DOT_EXTRA(j).SCTEST[1-4]
XDOT	vector	YBS.XDOT
YDOT	vector	YBS.YDOT
XLOW[j]	vector	YBS.DOT_ST(j).X_LOW
XHIGH[j]	vector	YBS.DOT_ST(j).X_HIGH
YLOW[j]	vector	YBS.DOT_ST(j).Y_LOW
YHIGH[j]	vector	YBS.DOT_ST(j).Y_HIGH
SCTEST_START[j,1:4]	matrix	starting character index for SCTEST[j]
SCTEST_END[j,1:4]	matrix	final character index for SCTEST[j]
COL_POINTER[j,1:4]	matrix	YBS.DOT_ST(j).COL_ST(1:4).COL_POINTER
COL_OFF[j,1:4]	matrix	YBS.DOT_ST(j).COL_ST(1:4).COL_OFF

Furthermore: SCTEST[j][SCTEST_START[j,k]:SCTEST_END[j,k]] = YBS.DOT_EXTRA(j).SCTEST[k]

YBOS examples

To draw histogram n:

```
label\xaxis htitle[n]
scalar\dummy j
graph loop(hlow[n]+(j-1)*hbinw[n],j,1:hnbin[n]) -
 hist[hstart[n]:hstart[n]+hnbin[n]-1]
```

To draw dotplot n:

```
do j = [1:4]
  if (col_off[n,j] > 0 ) then
    scale xlow[n] xhigh[n] 5 ylow[n] yhigh[n] 5
    graph\axesonly
    colour j
    graph\noaxes -
     xdot[col_pointer[n,j]:col_pointer[n,j]+col_off[n,j]-1] -
     ydot[col_pointer[n,j]:col_pointer[n,j]+col_off[n,j]-1]
    set
     %xloc 32
     %yloc 90-3*j
     %txthit 2.0
      cursor -2

    text rchar(col_off[n,j],'F6.0')//'/'//sctest[n]
  endif
enddo
```

μSR MUD data sets

Syntax `RESTORE\MUD file`

The `RESTORE\MUD` command restores μSR MUD type data files containing histograms. The actual histogram data is stored in a vector, `HISTDATA`, and histogram number `j` can be extracted using: `HISTDATA[HISTSTART[j]:HISTEND[j]]`. The global title is in a string variable called `TITLE`. The title for histogram number `j` is in the array string variable `HISTITLE[j]`. For more information on the MUD data format, visit the URL:

`http://www.triumf.ca/dagroup/modas/mud_friendly.html`

General variables

scalars	*strings*
RUNNUMBER	TITLE
RUNDESC	LAB
EXPTNUMBER	AREA
ELAPSEDSEC	METHOD
TIMEBEGIN	APPARATUS
TIMEEND	INSERT
	SAMPLE
	ORIENT
	DAS
	EXPERIMENTER

Comments

scalars	*vectors*	*strings*
COMMENTTYPE	COMMENTPREV	COMMENTAUTHOR
COMMENTNUM	COMMENTNEXT	COMMENTTITLE
	COMMENTTIME	COMMENTBODY

Histograms

scalars	*vectors*	*strings*
HISTNUM	NUMBINS	HISTTITLE
	HISTSTART	
	HISTEND	
	HISTDATA	
	HISTTYPE	
	HISTNUMBYTES	
	HISTBYTESPERBIN	
	HISTFSPERBIN	
	HISTTO_PS	
	HISTTO_BIN	
	HISTGOODBIN1	
	HISTGOODBIN2	
	HISTBDGD1	
	HISTBKGD2	
	NUMEVENTS	

Scalers

scalars	*vectors*	*strings*
SCALERTYPE	SCALERCOUNTS	SCALERLABEL

Independent variables

scalars	*vectors*	*strings*
INDVARTYPE	INDVARLOW	INDVARNAME
	INDVARHIGH	INDVARDESCRIPTION
	INDVARMEAN	INDVARUNITS
	INDVARSTDDEV	
	INDVARSKEWNESS	

μSR data sets

Syntax `RESTORE\MSR file`

The `RESTORE\MSR` command restores μSR type data files containing histograms. Histogram number `j` is stored in column `j` of the matrix `IH`. The global title is in a string variable called `TITLE`. The title for histogram number `j` is in the array string variable `HTITLE[j]`. There are also associated scalars:

`IRUN`	is the run number
`NHISTS`	is the number of histograms
`NBINS`	is the number of bins
`NS_BIN`	is the binning increment

The associated vectors:

`TOTEV[j]`	is the total number of events in histogram `j`
`IT0[j]`	is the start of background for histogram `j`
`IT1[j]`	is the start of data for histogram `j`
`IT2[j]`	is the end of data for histogram `j`

To plot histogram `J`, enter:

```
GRAPH\HISTOGRAM [0:(NBINS-1)*NSBIN:NSBIN] IH[1:NBINS,J]
```

IμSR data sets

Syntax `RESTORE\IMSR file`

The `RESTORE\IMSR` command restores IμSR type data files. Histogram number `j` is stored in column `j` of the matrix `IX`. The run number is stored in a scalar called `IRUNNO`. The version number is stored in a scalar called `IVERS`. The global title is in a string variable called `ITITLE`. The subtitle, if it exists in the file, is in a string variable called `ISUBTITLE`. The title for histogram number `j` is in the array string variable `XTITLE[j]`.

To plot histogram `J`, enter:

```
GRAPH\HISTOGRAM [1:VLEN(M)(1)] IX[*,J]
```

CHAOS data sets

The `RESTORE\CHAOS` command restores CHAOS type data files containing histograms. The run number is stored in a scalar variable named `RUN_NUMBER`. The number of events analyzed will be stored in a scalar variable named `I_ANALYZED`. The number of histograms and the number of channels are stored in scalar variables named `NHIST` and `NCHAN`. The vectors `XLO`, `XHI`, `NBINS`, and `HSTART` will be created. Each of these will have `NHIST` elements. The histogram data will be stored in a vector named `HIST`, with `NCHAN` elements.

The array string variable `HNAMES` will be created, with `NHIST` elements. `HNAMES[i]` is the name of histogram number `i`. The array string variable `TITLES` will be created, with `NHIST` elements. `TITLES[i]` is the title of histogram number `i`. The array string variable `EVENT_CALIB` will be created, with `NHIST` elements. `EVENT_CALIB[i]` is the event/calibration flag for histogram number `i`.

The data for histogram number `I` is located in `HIST[HSTART[i]:NBINS[i]+HSTART[i]-1]`. To make an x vector for plotting histogram number `I`, enter:

```
GENERATE X XLO[I],,XHI[I] NBINS[I]+HSTART[I]-1
```

RETURN

Syntax `RETURN`

The `RETURN` command is to be used in conjunction with script files.

If the `RETURN` command is encountered in a script file, control passes back to the calling script, if there is one, or to the keyboard, if that script was the top level script. You can also type `RETURN` from the keyboard after a `TERMINAL` command to abort that script.

SAVE

Syntax `SAVE file`

The `SAVE` command saves all the data associated with the current run, that is, all variable names, contents, and histories. This includes all scalars, vectors, matrices, and string variables. This information is written in a special binary format into the specified file. The run may be resumed later by means of the `RESTORE` command, page 211.

The plot information necessary for a replot is also saved. The window definitions are saved, as well as the colour, the default filename extension for executable files, the stack file, autoscaling type, and any aliases that have been defined. The hardcopy device as chosen with the `DEVICE` command is saved, as well as the graphics `ORIENTATION`.

User defined routine information is not saved, but must be regenerated explicitly in the restored session.

SCALAR

Syntax	`SCALAR s1 { s2 ... }`
Qualifiers	`\DUMMY, \VARY`
Examples	`SCALAR A B C` `SCALAR\DUMMY I J K` `SCALAR\VARY P1 P2`

The `SCALAR` command defines `sI` to be a scalar. If `sI` exists and is a scalar, there is no affect. If `sI` exists but is not a scalar, it will be destroyed first. If `sI` does not exist, it will be created and initialized to the value one (1).

Fit parameters

Syntax `SCALAR\VARY s1 { s2 ... }`

The `SCALAR\VARY` command defines `sI` to be a scalar variable as above, but it also allows `sI` to vary during a `FIT`. These fit parameters are treated as fixed value scalars, except by the `FIT` command. The `FIT` command can vary a parameter's value to minimize the least squares residual and the result is still a scalar.

To fix a parameter so it can no longer vary use the `SCALAR` command with no qualifier.

Dummy variables

Syntax `SCALAR\DUMMY s1 { s2 ... }`

The `SCALAR\DUMMY` command defines `sI` to be a scalar dummy variable for use in functions requiring dummy variables: `LOOP, SUM, PROD, RSUM,` and `RPROD`. Dummy variables cannot be used in other contexts, since they have no fixed values.

SCALES

Syntax	`SCALES { minx maxx nlxinc miny maxy nlyinc }` `SCALES minx maxx miny maxy`
Qualifiers	`\COMMENSURATE`
Defaults	non-commensurate axes
Examples	`SCALES` `SCALES 0 0 0 .1 .5 0` `SCALES\COMM -10 5 3 .01 .05 4`

The `SCALES` command turns off the autoscaling feature, as set with the `SET AUTOSCALE` command, and sets the graph scales for subsequent graphs as follows:

`minx`	–	minimum value to display on the x-axis
`maxx`	–	maximum value to display on the x-axis
`nlxinc`	–	number of large (numbered) tic marks on the x-axis
`miny`	–	minimum value to display on the y-axis
`maxy`	–	maximum value to display on the y-axis
`nlyinc`	–	number of large (numbered) tic marks on the y-axis

If no parameters are entered, the scales will be frozen at their current values, and the current values of `minx`, `maxx`, `nlxinc`, `miny`, `maxy`, and `nlyinc` will be displayed.

If four (4) parameters are entered, the number of labeled tic marks on the x and y-axes, that is, `nlxinc` and `nlyinc`, default to zero (0), which means the program will choose "nice" numbers of increments.

Commensurate axis scaling

If the `\COMMENSURATE` qualifier is used, the axes will be drawn with commensurate scaling, that is, the lengths of both axes will be adjusted to give a commensurate graph. The lengths of the axes are adjusted at the time the `SCALES\COMMENSURATE` command is given, which means that if you change windows, with the `WINDOW` command, after the `SCALES\COMMENSURATE` command, the axis lengths will be incorrect. Choose the window you want and then set the axis scales.

Labeled tic marks

The number of large, labeled, tic marks to be displayed on the x-axis is controlled by `nlxinc`. Its initial value is 5%.

`nlxinc`	<0	the number of large tic marks on the x-axis will be `nlxinc` $+1$. If the x-axis is logarithmic, that is, if `XLOG` >1, then `nlxinc` may be changed to avoid fractional powers. If the x-axis is linear, that is, if `XLOG` <1, then the x-axis minimum and maximum will not be labeled with numbers.
`nlxinc`	$=0$	the number of large tic marks on the x-axis will be automatically determined for the user. The virtual x-axis minimum and maximum will be determined, so the x-axis may not begin or end at a large tic mark
`nlxinc`	>0	the number of large tic marks on the x-axis will be `nlxinc` $+1$. If `XLOG` >1, logarithmic x-axis, `nlxinc` may be changed to avoid fractional powers. If `XLOG` <1, linear x-axis, `nlxinc` will not be changed.

The number of large, labeled, tic marks to be displayed on the y-axis is controlled by `nlyinc`. Its initial value is 5%.

`nlyinc`	<0	the number of large tic marks on the y-axis will be `nlyinc` $+1$. If the y-axis is logarithmic, that is, if `YLOG` >1, then `nlyinc` may be changed to avoid fractional powers. If the y-axis is linear, that is, if `YLOG` <1, then the y-axis minimum and maximum will not be labeled with numbers.
`nlyinc`	$=0$	the number of large tic marks on the y-axis will be automatically determined for the user. The virtual y-axis minimum and maximum will be determined, so the y-axis may not begin or end at a large tic mark
`nlyinc`	>0	the number of large tic marks on the y-axis will be `nlyinc` $+1$. If `YLOG` >1, logarithmic y-axis, `nlyinc` may be changed to avoid fractional powers. If `YLOG` <1, linear y-axis, `nlyinc` will not be changed.

Example

The following script produces Figure 2.23:


```
WINDOW 5                         ! choose a window
SCALES -.5 .5 2 -5 5 2           ! set the graph scales
GRAPH\AXESONLY                   ! plot only the axes
GET                              ! get some GPLOT keyword values
 %XLAXIS XLX                     !
 %XUAXIS XUX                     !
                                 ! don't forget the blank line
SET                              ! set some GPLOT keyword values
 CURSOR -2                       ! centred text
 %XLOC (XLX+XUX)/2               ! x reference point for text
 %YLOC 50                        ! y reference point for text
                                 ! don't forget the blank line
TEXT 'SCALES -.5 .5 2 -5 5 2'    ! draw a string
WINDOW 6                         !
SCALES 10 50 4 -100 -80 2        !
GRAPH\AXESONLY                   !
TEXT 'SCALES 10 50 4 -100 -80 2' !
WINDOW 7                         !
SET YLOG 10                      ! y-axis logarithmic, base 10
SCALES 10 50 4 2 6 4             !
GRAPH\AXESONLY                   !
TEXT 'SCALES 10 50 4 2 6 4'      !
SET %YLOC 40                     !
TEXT 'YLOG = 10'                 !
SET %YLOC 50                     !
WINDOW 8                         !
SET YLOG -2                      ! base 2, numbered without powers
GRAPH\AXESONLY                   !
TEXT 'SCALES 10 50 4 2 6 4'      !
SET %YLOC 40                     !
TEXT 'YLOG = -2'                 !
```

SET

Syntax	`SET { keyword { value }}`
	`SET { keyword = value }`
Examples	`SET %XLAXIS 20`
	`SET %XLAXIS=XLX`

The `SET` command sets the values of the GPLOT plot characteristic keywords and the PHYSICA specific keywords. Use the `GET` command, page 107, to obtain the values of these keywords.

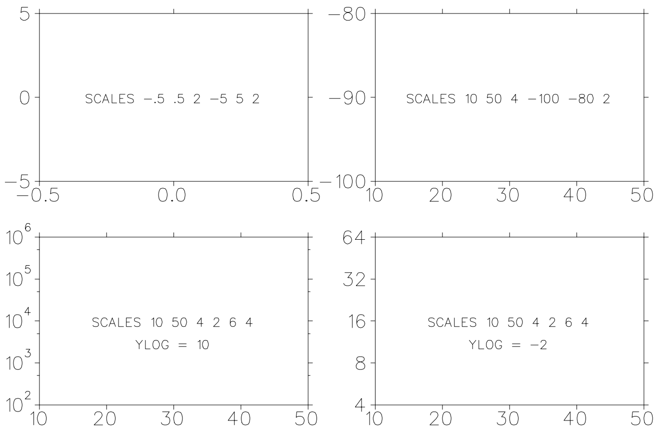


Figure 2.23: An example using the SCALES command

How the SET command works

If the SET command is entered with no parameters, more than one keyword value can be obtained without re-entering the SET command. Other keywords and values will be requested, until a blank line is entered, at which time the user is put back into command line entry mode.

If the SET command is entered with no parameters in a script file, a blank line is *necessary* to indicate that the SET command is finished.

If a keyword is entered with the SET command, then only that one keyword's value can be changed, with that command. If a value is not entered after the keyword, the current value is displayed. The current value is unchanged if no new value is entered.

Examples

To display the current value of XMIN, enter: `GET XMIN`

The following script gets the current value of XMIN and then changes it to XMIN−10, and sets the value of XMAX to XMIN+100


```
...
GET XMIN A    ! makes a scalar A
SET
 XMIN A-10
 XMAX A+100   ! don't forget the blank line to finish

...
```

GPLOT keywords

See **Appendix A** for descriptions of all the GPLOT plotting characteristic keywords. The tables produced by the MENU contain most of the keywords that can be accessed with the SET and GET commands, along with their current values.

The GPLOT keywords: MASK, ALIAS, PMODE, PTYPE, and ERRBAR, should *not* be changed in PHYSICA, as these are internally adjusted and used by various commands.

The PHYSICA keywords

The keywords, ARROLEN, ARROTYP, and ARROWID, apply to arrows drawn with the FIGURE ARROW command, page 82. See Figure 2.24 for examples of the available arrow styles.

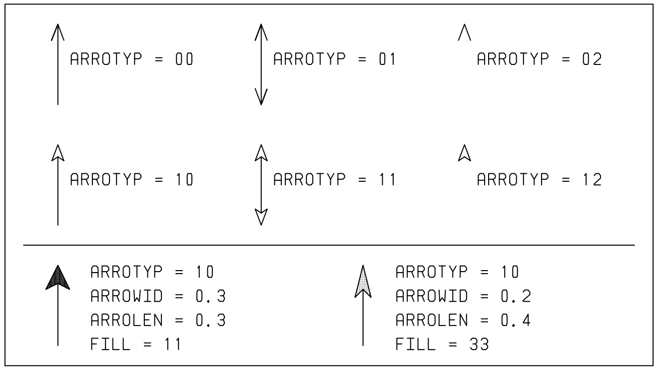


Figure 2.24: Arrow styles

ARROLEN

Default value: ARROLEN = 0.20

ARROLEN is the arrow head length expressed as a fraction of the length of the arrow's shaft.

ARROTYP

Default value: ARROTYP $= 00$

ARROTYP determines the type of arrow to draw. See Table 2.54.

ARROTYP		*arrow style*
00	open head,	line segment with arrowhead at one end
01	open head,	line segment with arrowhead at each end
02	open head,	arrowhead at one end with no line segment
10	closed head,	line segment with arrowhead at one end
11	closed head,	line segment with arrowhead at each end
12	closed head,	arrowhead at one end with no line segment

Table 2.54: The ARROTYP code and corresponding arrow styles

ARROWID

Default value: ARROWID $= 0.15$

ARROWID is the arrow head width expressed as a fraction of the length of the arrow's shaft.

AUTOSCALE

Parameters ON | OFF | XAXIS | YAXIS | COMMENSURATE

Parameter Qualifier \VIRTUAL

Examples SET AUTOSCALE ON
SET AUTOSCALE XAXIS\VIRTUAL
SET AUTOSCALE COMMENSURATE

The AUTOSCALE keyword controls autoscaling for graph axes. Autoscaling means to automatically choose the minimum and maximum values for the axes, as well as the number of large, numbered, tic marks for the axes. Autoscaling affects commands that draw axes, that is, the commands GRAPH, CONTOUR, DENSITY, REPLOT, and SLICES. The type of autoscaling that is done depends on the keyword that is used.

keyword	*result*
ON	Autoscale the horizontal and the vertical axes
COMMENSURATE	Autoscale the horizontal and vertical axes and change the lengths of the axes so that they will be commensurate
XAXIS	Autoscale the horizontal axis only, the vertical axis will remain as currently set
YAXIS	Autoscale the vertical axis only, the horizontal axis will remain as currently set
OFF	turn off all autoscaling, the axes will appear as they are currently set

Autoscaling remains in effect until either the command SET AUTOSCALE OFF is entered, or the SCALES command is entered. The default is ON.

When the \VIRTUAL qualifier is used, the virtual minima and maxima for the axes will be determined, so that the axes may not begin or end at a large tic mark. If the keyword ON is used, both x- and y-axes will have virtual minima and maxima. If the keyword XAXIS is used, only the x-axis will have virtual minimum and maximum. If the keyword YAXIS is used, only the y-axis will have virtual minimum and maximum.

CNTSEP

Default value: %CNTSEP $= 50\%$

The separation between contour labels can be set with CNTSEP or %CNTSEP. If %CNTSEP is used, the separation is a percentage of the height of the window, that is, YUWIND-YLWIND. If CNTSEP is used, the separation is expressed in centimeters or inches, depending on the units type as set with the SET UNITS command.

LABSIZ

Default value: %LABSIZ $= 1.5$

LABSIZ, or %LABSIZ, is the size of the contour labels. %LABSIZ is the size as a percentage of the height of the window, that is, YUWIND-YLWIND, while LABSIZ is the size expressed in centimeters or inches, depending on the units type as set with the SET UNITS command.

LEGSIZ

Default value: %LEGSIZ $= 1.6$

`LEGSIZ`, or `%LEGSIZ`, is the size of the contour plot and density plot legend entries. `%LEGSIZ` is the size as a percentage of the height of the window, that is, `YUWIND-YLWIND`, while `LEGSIZ` is the size expressed in centimeters or inches, depending on the units type as set with the `SET UNITS` command.

LEGFRMT

Default value: `LEGFRMT` $= `1PE10.3'$

The numeric legend entries drawn by the `DENSITY` and `CONTOUR` commands are written using the `LEGFRMT` format.

ERRFILL

Default value: `ERRFILL` $= 0$

If the `\ERRFILL` qualifier is used with the `READ` command, and an invalid field is encountered on a record in the data file, then `ERRFILL` will be used.

format used:	all fields on that record will be set to `ERRFILL`
no format used:	only the invalid field will be set to `ERRFILL`

FILL

Default value: `FILL` $= 0$

`FILL` is used in the `FIGURE` command, with the fillable figures: `BOX`, `POLYGON`, `WEDGE`, `CIRCLE`, `ELLIPSE`, and `ARROW`s with closed heads. It is also used for filling the boxes with the `DENSITY\BOX` command. See Table 2.55 for a description of the interpretations of the `FILL` keyword.

`FILL` $= 0$	no filling
$1 \leq$ `|FILL|` ≤ 10	fill with hatch pattern `|FILL|`
$11 \leq$ `|FILL|` ≤ 99	fill with dithering pattern if `|FILL|` $= nm$, n is the increment number of dots horizontally and m is the increment number of dots to light up vertically
`FILL` < 0	erase using `|FILL|` as above

Table 2.55: Interpretations of the `FILL` keyword

For example, `FILL` $= 34$ means to light up every 3^{rd} dot horizontally and every 4^{th} dot verti-

cally. `FILL` $= -34$ means to erase every 3^{rd} dot horizontally and every 4^{th} dot vertically. `FILL` $= 8$ means to fill with hatch pattern 8. `FILL` $= -8$ means to erase using hatch pattern 8.

See the `SET HATCH` command for information on changing the hatch pattern definitions. See the `DISPLAY HATCH` command for information on how to display examples of the hatch patterns.

HATCH

Parameters: `hnum` { `spaces angle` }

The `SET HATCH` command is used for changing the hatch pattern definitions that are used for text bolding, for filling areas under curves or histograms, and for use by the `TILE`, `PIEGRAPH`, and `FIGURE` commands.

The `SET HATCH` command does *not* choose the hatch pattern to be used by other commands. It only alters the definition of a hatch pattern.

If just the keyword `HATCH` is entered, a table of the spacings and angles for all ten hatch patterns is displayed. You will be requested to enter a hatch pattern number, `hnum`, a spacing scalar or a vector of spacings, `spaces`, and an angle, `angle`. Typing the `<RETURN>` key, without entering anything, will leave the current values unchanged. If the hatch pattern number, a spacing scalar or vector and an angle are entered, then nothing will be displayed. For example, to set hatch pattern `3` to have spacings of `[0.05;0.1;0.2]` and an angle of `45`°, enter:

```
SET HATCH 3 [0.05;0.1;0.2] 45
```

while to set hatch pattern `7` to have a single spacing of `0.5` and an angle of `-45`°, enter:

```
SET HATCH 7 .5 -45
```

The hatch pattern number, `hnum`, should be between one and ten. A spacing vector, `spacevec`, must have no more than ten elements. A hatch pattern is composed of an angle and from one to ten spacings. The default spacings and angles are listed in Table 2.56. The angles are in degrees and the spacing lengths, by default, are expressed in centimeters, but if the units are changed to inches, with the `SET UNITS` command, the lengths will be converted to inches. See Figure 2.7 on page 61 for examples of the hatch patterns. See the `DISPLAY` command, page 58, for information on how to display examples of the hatch patterns.

When an object is being filled, a line is drawn inside the object at the specified angle, then a parallel line is drawn at the first spacing, and so on for the number of spacings in that pattern. This process is repeated until the object is filled.

Pattern Number	*Spacings* 1	2	angle
1	0.01		0
2	0.01		90
3	0.05		0
4	0.05		90
5	0.10		0
6	0.10		90
7	0.20		45
8	0.20		−45
9	0.20	0.10	45
10	0.20	0.10	−45

Table 2.56: The hatch pattern defaults

LINE

Parameters: `n { v }`

The `SET LINE` command is used for changing the definition of the line types that are used by the commands: `GRAPH`, `LINE`, `PICK`, `ELLIPSE`, `FIGURE`, and `ZEROLINES`.

This command does *not* choose the line type to be used by other commands. It only alters the definition of a line type. To choose a line type, use the `SET LINTYP` command.

If just the keyword `LINE` is entered, a table of the spacings for all ten line types is displayed. You will be requested to enter a line type number, `n`, and a vector of spacings, `v`. Typing the `<RETURN>` key, without entering anything, will leave the current values unchanged. If the line type number and a spacing vector are entered, then nothing will be displayed. For example, to set line type 2 to `[0.05;0;0]`, enter:

```
SET LINE 2 [0.05;0;0]
```

The line type number, `n`, should be between one and ten. The spacing vector, `v`, must have three elements. The line types come in four different styles:

- ordinary solid line
- double line of specified width
- dashed line with specified dash and space lengths
- dashed line with two different dash lengths

The different line types are achieved by specifying the three lengths `v[1]`, `v[2]`, and `v[3]` as shown in Table 2.57.

`v[1]`	`v[2]`	`v[3]`	*Result*
$=0$			ordinary solid line
>0	$=0$		double line with width `v[1]`
>0	>0	$=0$	dashed line with dash length `v[1]`, space length `v[2]`
>0	>0	>0	dashed line with first dash length `v[1]`, space length `v[2]`, and second dash length `v[3]`

Table 2.57: Line type definitions

There are ten line types available. The defaults are listed in Table 2.58. The lengths are expressed in centimeters, the default, but if the units are changed to inches, with the `SET UNITS` command, the lengths will be converted to inches. See Figure 2.8 on page 62 for examples of the default line types. See the `DISPLAY` command, page 58, for information on how to display examples of the line types.

line type	`v[1]`	`v[2]`	`v[3]`
1	0.00	0.00	0.00
2	0.07	0.00	0.00
3	0.50	0.30	0.00
4	0.50	0.30	0.10
5	0.30	0.30	0.00
6	0.30	0.30	0.10
7	0.20	0.20	0.00
8	0.20	0.20	0.05
9	0.05	0.20	0.00
10	0.05	0.30	0.00

Table 2.58: The line type defaults

SHOWHISTORY

Default value: `SHOWHISTORY` $=5$

`SHOWHISTORY` controls how many lines of history to display for each numeric variable as a result of the `SHOW` command.

MAXHISTORY

SHOWHISTORY		
$n < 0$	$\rightarrow$	all stored history lines will be displayed
$n = 0, 1$	$\rightarrow$	only the latest history line will be displayed
$n > 0$	$\rightarrow$	a maximum of n lines of history will be displayed for each variable

Table 2.59: The SHOWHISTORY keyword interpretation

Default value: MAXHISTORY $= 5$

MAXHISTORY is the maximum number of history lines to store for each numeric variable. MAXHISTORY was added because if a variable had its value changed within a large DO loop, a new history line was added each time the loop was processed, which could lead to virtual memory problems.

WRAP

Default value: WRAP $= 0$

If WRAP $= 0$, history lines and string variable contents lines are not wrapped when displayed with the SHOW command. If WRAP is non-zero, these lines are wrapped.

TENSION

Default value: TENSION $= 1.0$

TENSION controls the spline tension for the functions using cubic splines:

DERIV, INTEGRAL, INTERP, SMOOTH, SPLINTERP, and SPLSMOOTH.

SEED

Default value: SEED $= 12345$

SEED is the random number seed value. This seed is updated whenever the GENERATE\RANDOM command is entered, or the RAN is used.

POSTRES

Default value: POSTRES $= 180$

POSTRES controls the PostScript graphics output resolution, in dots per inch. This applies to dot filled text characters and dot types of DENSITY plots. The resolution can be changed at

any time, so different parts of a single drawing can be drawn with different resolutions.

SPEED

Default value: SPEED $= 20$

SPEED controls the pen plotter speed. This applies to Hewlett-Packard, Houston, and Roland RDGL II pen plotters. The speed can be changed at any time, so different parts of a single drawing can be drawn at different speeds.

WIDTH

Default value: WIDTH $= 80$

WIDTH controls the character width of the alphanumeric monitor screen. The value for WIDTH should be between 2 and 132.

XPREV

Default value: XPREV $= 0$

XPREV is the last world x-coordinate that was drawn by any graphics command. The value of this keyword is automatically updated.

YPREV

Default value: YPREV $= 0$

YPREV is the last world y-coordinate that was drawn by any graphics command. The value of this keyword is automatically updated.

NCURVES

Default value: NCURVES $= 0$

NCURVES is the total number of data curves that have been drawn, using the GRAPH command, since the last CLEAR command. The value of this keyword is automatically updated.

PCHAR

Parameters: `symbol { size { colour { angle }}}`

PCHAR controls the plotting symbols, or the appearance of the histogram bars, when the GRAPH command is entered. Each of the parameters may be either a literal constant, a scalar, or a vector. Entering a constant or a scalar is similar to entering a vector whose elements are all equal to the constant or the value of the scalar.

The interpretations of the parameters depend on the value of the GPLOT keyword HISTYP *at the time* that the GRAPH command is entered. HISTYP is also changed with the SET command. See Table 2.39 on page 118 for an explanation of HISTYP. If HISTYP $= 0$, the default value, subsequent graphs will be normal line graphs.

Plotting symbols

When plotting normal line graphs, the first parameter, `symbol`, determines the plotting symbol to draw at each data point, and whether or not to connect the symbols with line segments. Plotting symbol zero means do not draw a symbol, but just draw line segments joining the data points. This is the default.

The symbol that will be plotted at the point `(x[i],y[i])` will be the ASCII decimal equivalent of the absolute value of `symbol[i]`. For example, 54 represents the ASCII character 6, 35 represents the ASCII character #, and 78 represents the ASCII character N. The maximum absolute value of a plotting symbol is ninety-five (95), so the lower case alphabetic characters are not available. These symbols have their lower left corners at the data point.

There are eighteen (18) special plotting symbols which are centred at the data points using the code numbers from one (1) to eighteen (18). See Figure 2.9 on page 62. These symbols are *not* translated into their ASCII decimal equivalents. See the DISPLAY command, page 58, for information on displaying the plotting symbols on the monitor screen.

A plotting symbol value of zero means do not draw a symbol, but connect that point to the previous point. If the value of `symbol[i+1]` is greater than zero, the symbol drawn at point `(x[i+1],y[i+1])` will be connected to the symbol drawn at point `(x[i],y[i])`. If the value of `symbol[i+1]` is less than zero, the symbol drawn at point `(x[i+1],y[i+1])` will *not* be connected to the symbol drawn at point `(x[i],y[i])`.

Plotting symbol size

When plotting normal line graphs, the first optional parameter, `size`, controls the relative size of each individual plotting symbol. If `size` is entered, the size of the symbol at point `(x[i],y[i])` will be `size[i]`×CHARSZ. The GPLOT keyword CHARSZ can be changed with the SET command.

Plotting symbol colour

When plotting normal line graphs, the second optional vector, `colour`, is interpreted as the colour code for the plotting symbol. If `colour` is entered, the colour code of the plotting symbol at point `(x[i],y[i])` will be `colour[i]`. Table 2.6 on page 20 shows the recognized colours and their associated colour codes.

Plotting symbol angle

When plotting normal line graphs, the third optional parameter, `angle`, is interpreted as the rotation angle, in degrees, of the plotting symbol. If `angle` is entered, the angle of the plotting symbol drawn at point `(x[i],y[i])` will be `angle[i]`. The angle of all of the plotting symbols defaults to CHARA, which has a default value of $0°$. The GPLOT keyword CHARA can be changed with the SET command.

Histograms

Parameters: `fill { w { c }}`

The GRAPH command used with the \HISTOGRAM qualifier plots histograms. Alternatively, SET HISTYP n where $n > 0$, forces subsequent graphs to be histogram type graphs. See Table 2.39 on page 118 for an explanation of HISTYP.

HISTYP $= 0$	non-histogram
HISTYP $= 2$ or 4	tails will be drawn the user has control over individual histogram bar filling, width, and colour
HISTYP $= 1$ or 3	no tails are drawn the user has control over individual histogram bar width, and colour

The third optional parameter, `angle`, is ignored if a histogram is plotted.

Fill patterns

If HISTYP is 1 or 3, the first parameter, `fill`, is ignored. To fill the area under the histogram in this case, use the SET command to set the value of LINTYP to $100+$ the hatch pattern number or to $200+$ the dot pattern number. For example, 107 refers to hatch pattern number 7, while 244 refers to a dot fill pattern of every fourth dot both horizontally and vertically.

If HISTYP is 2 or 4, the first parameter, `fill`, determines the fill pattern for the individual bars of the histogram. The filling can be done with grey scales or with hatch patterns.

100	$\leq$	`|fill[i]|`	$\leq$	110	means fill with a hatch pattern
200	$\leq$	`|fill[i]|`	$\leq$	299	means fill with a grey scale

If $200 \leq$ `|fill[i]|` ≤ 299, the grey scale pattern `fill[i]`-200 will be used to fill the histogram bar at `(x[i],y[i])`. For example, if `fill[i]` $= 234$, then grey scale dot pattern 34 will be used.

`fill[i]`	$=$	200	means every dot is lit (same as 211)
`fill[i]`	$>$	0	means draw
`fill[i]`	$<$	0	means erase

A grey scale dot pattern is of the form: uv, where the digit u is the increment number of dots to light up horizontally, $1 \leq u \leq 9$, and the digit v is the increment number of dots to light up vertically, $1 \leq v \leq 9$. For example, a grey scale dot pattern of 34 means to light up every third dot horizontally and every fourth dot vertically. If uv is negative, then the dots are erased instead of lighted. A grey scale pattern of zero, 00, is interpreted the same as pattern 11, that is, every dot is lit.

For PostScript output, set the `POSTRES` keyword to the appropriate resolution for your hard-copy device, using the `SET` command. Use a combination of line thickness and resolution for a quicker resulting picture. For example, the Lexmark inkjet printer has a resolution of 360 dpi. A "good" picture can be obtained with `POSTRES` $= 180$ and `LINTHK` $= 2$.

If $100 \leq$ `|symbol[i]|` ≤ 110, the hatch pattern `fill[i]`-100 will be used to fill the histogram bar at `(x[i],y[i])`. For example, if `fill[i]` $= 108$, then hatch pattern 8 will be used.

`fill[i]`	$=$	100	means do no fill the histogram bar
`fill[i]`	$>$	0	means the histogram bar outline will be drawn
`fill[i]`	$<$	0	means the histogram bar outline will not be drawn

A hatch pattern is composed of an angle and one to ten spacings. These spacings are simply cycled through as each histogram bar is being filled, that is, a line is drawn inside the histogram bar at the specified angle, then a parallel line is drawn at the first spacing, then another parallel line is drawn at the second spacing, and so on for the number of spacings in that pattern. This process is repeated until the histogram bar is filled. There are ten hatch patterns available. See Figure 2.7 on page 61 for hatch pattern examples.

See the `HATCH` keyword for information on changing the hatch pattern definitions. See the `DISPLAY` command for information on how to display examples of the hatch patterns.

Histogram bar width

The first optional parameter, `w`, allows the user to control the widths of individual histogram bars. If `w` is entered, the width of the histogram bar at `(x[i],y[i])` will be the normal width times `w[i]`.

Histogram bar colour

The second optional parameter, `c`, allows the user to control the colours of individual histogram bars. If `c` is entered, the colour code for the histogram bar at `(x[i],y[i])` will be `c[i]`. See Table 2.6, page 20, for the colour associated with each colour code.

UNITS

Parameter values: `CM | IN`
Default value: `UNITS` = `CM`

Note: The value of the `UNITS` keyword is a string instead of a numeric value.

`UNITS` controls the plotting units type, either centimeters, `CM`, the default, or inches, `IN`.

CUNITS

Parameter values: `GRAPH | PERCENT | WORLD`
Default value: `CUNITS` = `PERCENT`

`CUNITS` controls the units type for the cursor readout when the graphics cursor is invoked by the `PICK`, `PEAK`, `LINE`, or `FIGURE` command when running under X Windows and mouse button two is pressed. If `WORLD` is chosen, the numbers displayed depend on the current units type, either centimeters or inches, as chosen with `SET UNITS`. If `GRAPH` is chosen, the numbers displayed depend on the current graph axis scales.

FONT

Default value: `FONT` = `TSAN`

Note: The value of the `FONT` keyword is a string instead of a numeric value.

`FONT` controls the graphics font. For a list of the font names, see Table 2.60. If just the `FONT` keyword is entered, a table of the font names is displayed, and you will be asked to enter a new font name. If you type the `<RETURN>` key, without entering anything, the font is left unchanged.

The `DISPLAY FONT` command will draw a font table for any font.

STANDARD	ITALIC.2	GOTHIC.ENGLISH	ROMAN.2
	ITALIC.2A	GOTHIC.FRAKTUR	ROMAN.2A
SANSERIF.1	ITALIC.3	GOTHIC.ITALIAN	ROMAN.3
SANSERIF.2			
SANSERIF.CART	SCRIPT.1	CYRILLIC.2	OLDALPH
	SCRIPT.2		
HELVETICA.1		KATAKANA	KANJI1
	GREEK.1	HIRAGANA	KANJI2
TRIUMF.1	GREEK.2		KANJI3
TRIUMF.2	GREEK.2A		KANJI4
TSAN	GREEK.CART		KANJI5
ROMAN.FUTURA	ROMAN.SWISSL	SPECIAL	HEBREW
ROMAN.SERIF	ROMAN.SWISSM	MATH	
ROMAN.FASHON	ROMAN.SWISSB	TRIUMF.OUTLINE	
ROMAN.LOGO1			

Table 2.60: The font names

SHOW

Syntax	`SHOW v1 { v2 ... }`
Qualifiers	`\VECTORS, \SCALARS, \MATRICES, \TEXT, \FIXED, \DUMMY, \VARY`
Examples	`SHOW` `SHOW X*` `SHOW\VECTORS\SCALARS *ABC*`

The `SHOW` command displays, on the terminal screen, a list of:

- the current vectors, their lengths and histories
- the current scalars, their values and histories
- the current matrices, their dimensions and histories
- the current string variables, their lengths and values

If a variable name is entered, only the information about that variable is listed. Wildcards, `*`, are allowed in the name, and then all variables that match will be listed.

If the `\VECTORS` qualifier is used, only the vector variables will be listed.
If the `\SCALARS` qualifier is used, only the scalar variables will be listed.
If the `\MATRICES` qualifier is used, only the matrix variables will be listed.
If the `\TEXT` qualifier is used, only the string variables will be listed.

The qualifiers `FIXED`, `DUMMY`, and `VARY` are *only* valid when used with the `\SCALARS` qualifier.

`\-FIXED`	⟶	do not show scalars that are fixed (with respect to fitting)
`\-DUMMY`	⟶	do not show scalars that are dummy variables
`\-VARY`	⟶	do not show scalars allowed to vary in a fit

Combinations are allowed, for example, `\-FIXED\-DUMMY` means only show scalars that are allowed to vary in a fit.

Ordered Vectors

All vectors now have an order property. Vectors are either in ascending order, descending order, or un-ordered. The type is displayed in the `SHOW` command, where `+O` means ascending order, `-O` means descending order, and no symbol means un-ordered. For now, being ordered only has an affect on the vector union, `/|`, and the vector intersection, `/&`. These operations are much faster if the vector operands are ordered. The `WHERE` function produces an ascending order vector, as does the `SORT\UP` command. The `SORT\DOWN` command produces a descending order vector. This new vector property will be utilised more in the future to enhance speed and efficiency.

Examples

Suppose you have defined the following variables:

vectors:	X	XX	AX	XB
scalars:	SX	XSX	S_1	
matrices:	M1	MX1		
string variables:	T1	T2	ST	TXT

`SHOW *X`	the information about `X`, `XX`, `AX`, `SX` and `XSX` will be displayed
`SHOW\S *X`	the information about `SX` and `XSX` will be displayed
`SHOW X*`	the information about `X`, `XX`, `XB` and `XSX` will be displayed
`SHOW\V X*`	the information about `X`, `XX` and `XB` will be displayed
`SHOW X`	the information about `X` only will be displayed
`SHOW *X*`	the information about `X`, `XX`, `AX`, `XB`, `SX`, `XSX`, `MX1`, and `TXT` will be displayed
`SHOW\T *X*`	the information about `TXT` will be displayed

SLICES

Syntax	`SLICES n x y z { a }`
Defaults	a = 30°
Examples	`SLICES 5 X Y Z`
	`SLICES 10 X Y Z 45`

The `SLICES` command bins the `x`, `y`, and `z` vectors into `n` slices, and produces a graphical representation of this binned data. The `x`, `y` and `z` vectors are assumed to represent scattered data points `(x[i],y[i])` with altitude `z[i]`. The z-axis is drawn at an angle, `a`, in degrees. `a` defaults to 30°.

This command is open for suggestions.

Example

The following script produced Figure 2.25.

```
X=COS([-100:100:.5])   ! generate some "data"
Y=SIN([10:20:.025])    !
Z=SIN(X)*COS(Y)        !
SET PCHAR -16          ! set plotting character to a point
SLICES 5 X Y Z 35      !
```

SORT

Syntax	`SORT x { x1 x2 ... }`
Qualifiers	`\UP`, `\DOWN`
Defaults	`\UP`
Examples	`SORT X Y`
	`SORT\DOWN X Y Z`
	`SORT\UP X Y Z`

The `SORT` command is used for sorting vectors into ascending or descending order. By default, the vector `x` is sorted into ascending order. To sort vector `x` into ascending order, it is not necessary to use the `\UP` qualifier. Ascending order means that element 1 will be the smallest element. To sort vector `x` into descending order, use the `\DOWN` qualifier. Descending order means that element 1 will be the largest. Vector `x` will be altered.

Associated vectors

If other vectors, `xI`, are entered, they will not be sorted. They will be re-arranged in the same

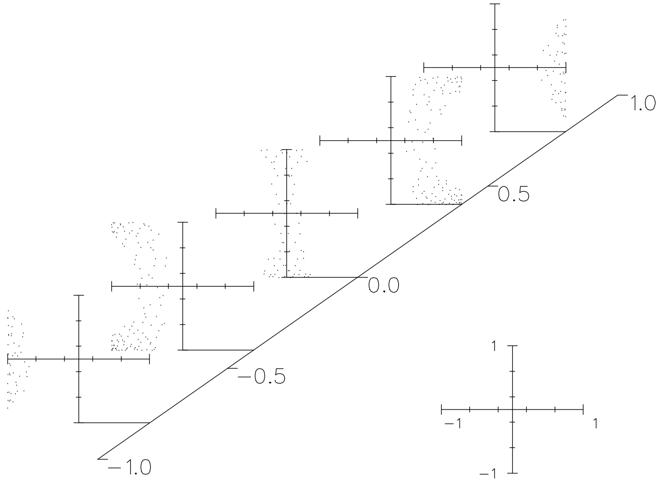


Figure 2.25: An example using the `SLICES` command

way that `x` is re-arranged, that is, if element `x[i]` becomes `x[j]` because of sorting then `xI[i]` will become `xI[j]`. This feature was included so that if data vectors are associated with each other they can be sorted without the associations being lost.

Examples

Suppose you have three data vectors, `X`, `Y`, and `Z`, which represent rectangular coordinates and an associated altitude:

```
X=[1;2;3;4;5]
Y=[10;8;6;4;2]
Z=[-0.3;-1.0;-0.5;2.0;-2.0]
```

If you want to sort `Z` into descending order, without breaking up the triplets, enter:

```
SORT\DOWN Z X Y
```

After which the vectors `X`, `Y`, and `Z` are:

```
X=[4;1;3;2;5]
Y=[4;10;6;8;2]
Z=[2.0;-0.3;-0.5;-1.0;-2.0]
```

However, if you want to sort Z into descending order, and are not concerned about keeping the Y and Z associations, simply enter:

```
SORT\DOWN Z
```

Suppose you now want to re-sort Y data into ascending order, and you are concerned about keeping the X and Z associations. Enter:

```
SORT Y X Z
```

after which:
```
X=[5;4;3;2;1]
Y=[2;4;6;8;10]
Z=[-2.0;2.0;-0.5;-1.0;-0.3]
```

STACK

Syntax	`STACK filename`
Qualifiers	`\APPEND, \EXECUTE`
Defaults	`\NOAPPEND, \EXECUTE`
Examples	`STACK FILE.STK` `STACK\APPEND FILE.STK` `STACK\NOEXE FILE.STK`

A stack file will contain all the subsequently entered commands. The STACK command is a way of interactively creating a command script file, and is meant to be used in conjunction with the EXECUTE command, page 76. Commands that are being stacked in a file can be executed later.

To turn off the stacking feature, use the DISABLE STACK command, after which commands will no longer be written to the previously named stack file. To re-enable the same stack file, use the command ENABLE STACK, after which subsequent commands will be appended to that file.

Environment variables in file names

For UNIX users, it is now possible to use an environment variable in a file name, if the environment variable is preceeded by a $. For example,

```
setenv FILE mysession
physica
stack $FILE
```

The environment variable can be just the first part of the filename, for example,

```
setenv FILE my
```



```
physica
stack $FILEsession
```

Appending to a stack file

The default, \NOAPPEND, is to open a new file. If the \APPEND qualifier is used, then subsequent commands will be appended to the stack file.

Executing commands while stacking

The default, \EXECUTE, is to execute commands normally. If the \NOEXECUTE qualifier is used, then subsequently entered commands will be written to the file, but will *not* be executed at that time.

STATISTICS

Syntax	`STATISTICS x { s1\keyword { s2\keyword ... }}` `STATISTICS\MOMENT w x n { sout }` `STATISTICS\PEARSON x y rcof prob`
Qualifiers	`\MESSAGES, \WEIGHTS, \MOMENT, \PEARSON`
Defaults	`\MESSAGES, \NOWEIGHTS, \NOMOMENT, \NOPEARSON`
Examples	`STATISTICS X` `STATISTICS\NOMESS X XMED\MEDIAN XMEAN\MEAN` `STATISTICS\WEIGHTS W X XVAR\VARIANCE XSUM\SUM` `STATISTICS\MOMENT Y X 3 M3`

The STATISTICS command calculates[1] various statistics for the input variable x, which can be a vector or a matrix. Specific statistics are chosen with qualifier keywords which are appended to the output parameters with the backslash, \.

Table 2.61 shows the parameter qualifier keywords and corresponding output values for extrema. Table 2.62 shows the parameter qualifier keywords and corresponding output values for central measures. Table 2.63 shows the parameter qualifier keywords and corresponding output values for dispersion and skewness.

Informational messages

The default is to display all the calculated statistics on the terminal screen. If the \NOMESSAGES command qualifier is used, and if at least one output scalar is entered, then the values of

[1]The definitions used here are taken from "Numerical Recipes – The Art of Scientific Computing" by W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, Cambridge University Press, 1986.

keyword	output value
\MAX	maximum value of x
\IMAX	index of the maximum if x is a vector
	row index of the maximum if x is a matrix
\JMAX	column index of the maximum if x is a matrix
\MIN	minimum value of x
\IMIN	index of the minimum if x is a vector
	row index of the minimum if x is a matrix
\JMIN	column index of the minimum if x is a matrix

Table 2.61: The STATISTICS command extrema keywords

keyword	output value
\SUM	arithmetic sum (unweighted)
\MEAN	arithmetic mean
\GMEAN	geometric mean
\MEDIAN	median value
\RMS	root-mean-square

Table 2.62: The STATISTICS command central measure keywords

keyword	output value
\VARIANCE	variance
\SDEV	standard deviation
\ADEV	average deviation
\KURTOSIS	kurtosis
\SKEWNESS	skewness

Table 2.63: The STATISTICS command dispersion and skewness keywords

all the statistics will not be displayed on the terminal screen.

Weights

Syntax `STATISTICS\WEIGHTS w x { s1\keyword { s2\keyword ... }}`

You *must* use the `\WEIGHTS` qualifier to indicate that a weight vector is present. Weights cannot be applied to matrix data.

A weighting factor, $w_i \geq 0$, could be the frequency, the probability, the mass, the reliability, or some other multiplier. The minimum of the lengths of w and x will be used. If the lengths are different, a warning message to that effect will be displayed on the terminal screen.

Definitions

Suppose that x is a vector with N elements.

If a weight vector, w, is entered, remember to use the `\WEIGHTS` command qualifier. The length of w is assumed to also be N. If no weight is entered, let w_i default to 1, for $i = 1, \ldots, N$. Define the total weight: $W = \sum_{j=1}^{N} w_j$

Sum

The sum is defined by $\sum_{j=1}^{N} x_j$

Mean value

The mean value, $\bar{x}$, is defined by $\bar{\mathtt{x}} = \frac{1}{W} \sum_{j=1}^{N} w_j x_j$

Geometric mean

The geometric mean, $\mathcal{G}_x$, is defined, if each $x_i \geq 0$, by: $\mathcal{G}_x = e^{\frac{1}{W} \sum_{j=1}^{N} w_j \log(x_j)}$

Median

The median is the element of x which has equal numbers of values above it and below it. If N is even, the median is the average of the unique two central values.

Root-mean-square

The root-mean-square, $\mathcal{RMS}$, is defined by $\mathcal{RMS} = \sqrt{\frac{1}{W} \sum_{j=1}^{N} w_j x_j^2}$

Variance

The variance, μ, is defined by $\mu = \frac{N}{W(N-1)}\sum_{j=1}^{N} w_j(x_j-\bar{x})^2$

Standard deviation

The standard deviation, σ, is defined by $\sigma = \sqrt{\mu}$

Average deviation

The average deviation, or mean deviation, δ, is defined by $\delta = \sum_{j=1}^{N} w_j|x_j-\bar{x}|/W$

Skewness

The skewness, or third moment, $skew$, is a nondimensional quantity that characterizes the degree of asymmetry of a distribution around its mean. The skewness is a pure number that characterizes only the shape of the distribution, and is defined by

$$skew = \sum_{j=1}^{N} w_j[\frac{x_j-\bar{x}}{\sigma}]^3/W$$

A positive value of skewness signifies a distribution with an asymmetric tail extending out towards more positive x; a negative value signifies a distribution whose tail extends out towards more negative x.

Kurtosis

The kurtosis, $kurt$, is a nondimensional quantity which measures the relative peakedness or flatness of a distribution, relative to a normal distribution. A distribution with positive kurtosis is termed leptokurtic; a distribution with negative kurtosis is termed platykurtic. An in-between distribution is termed mesokurtic. The kurtosis is defined by

$$kurt = (\sum_{j=1}^{N} w_j[\frac{x_j-\bar{x}}{\sigma}]^4) - 3$$

where the -3 term makes the value zero for a normal distribution.

Moments

Syntax `STATISTICS\MOMENT w x n { s }`

If the `\MOMENT` command qualifier is used, the `n`th moment of vector `x`, with weight `w`, is calculated and optionally stored in output scalar `s`. The moment number, `n`, can be any integer > 0.

$s = S/W$, where: $S = \sum_{i=1}^{N} w_i \times x_i^n$ and $W = \sum_{i=1}^{N} w_i$ and where N is the length of x and w.

If `x` and `w` have different lengths, a warning message to that effect will be displayed, and the minimum of the lengths of `x` and `w` will be used. If $W = 0$, then an error message is displayed and `s` is returned as 0.

Linear correlation coefficient

Syntax `STATISTICS\PEARSON x y r p`

Pearson's r, or the linear correlation coefficient, is widely used as a measure of association between variables that are continuous. For pairs of quantities (x_i, y_i), for $i = 1, \ldots, N$, the linear correlation coefficient r is given by the formula:

$$r = \frac{\sum_{i=1}^{N}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{N}(y_i-\bar{y})^2}}$$

where $\bar{x}$ is the mean of the x_i's, and $\bar{y}$ is the mean of the y_i's.

The value of r lies between -1 and $+1$, inclusive. It takes on a value of $+1$ when the data points lie on a straight line with positive slope, x and y increase together. The value $+1$ holds independent of the magnitude of this slope. If the data points lie on a straight line with negative slope, y decreases as x increases, then r has the value -1. A value of r near zero indicates that the variables x and y are uncorrelated.

r is a way of summarizing the strength of a correlation which is known to be significant, but it is a poor statistic for deciding whether an observed correlation is statistically significant, and/or whether one observed correlation is significantly stronger than another. The reason is that r is ignorant of the individual distributions of x and y, so there is no universal way to compute its distribution in the case of the null hypothesis.

The `STATISTICS\PEARSON` command returns Pearson's r in the scalar variable `r`. It also returns scalar `p`, the significance level at which the null hypothesis of zero correlation is disproved. A small value of `p` indicates a significant correlation.

$$p = I_{\frac{N-2}{N-2+t^2}}(\frac{N-2}{2}, \frac{1}{2})$$

where I is the incomplete Beta function and t is defined by: $t = r\sqrt{\frac{N-2}{1-r^2}}$

Examples

Suppose you have a vector X=[1.2;2.1;3.2;4.5;5;6;7]. Entering STATISTICS X produces the following display:

```
Minimum =             1.2              | Maximum =           7
Index of minimum =  [1]                | Index of maximum = [7]
Mean =                4.142857         | Geometric mean =    4.13629
Median not requested                   | Variance =          4.36619
Standard deviation = 2.089543          | Average deviation = 1.693878
Skewness =            -0.0696135       | Kurtosis =          -1.708838
```

If you want to use the values for the maximum, minimum and mean of X, enter:

```
STATISTICS X XMEAN\MEAN XMIN\MIN XMAX\MAX
```

and you will have the scalars: XMAX=7, XMIN=1.2, and XMEAN=4.142857

If you also want the index values for the maximum and the minimum of X, enter:

```
STATISTICS X XMEAN\MEAN XMIN\MIN XMAX\MAX IMX\IMAX IMN\IMIN
```

and you will also have scalars: IMX=7 and IMN=1.

STATUS

Syntax STATUS

The STATUS command displays on the terminal screen most of the internal PHYSICA flags and settings. Example output from the STATUS command, is shown below.


```
PHYSICA version number:   2.10
        version date:  January 16, 1998
Echoing           disabled |  Confirm           enabled
Prompting          enabled |  Recall shell      enabled
Border             enabled |  Replot            enabled
X replay           enabled |  Autoscale    X and Y axes
History recording  enabled |
Journaling enabled, file= PHYSICA.JOURNAL
                    macro journaling disabled
Stack     disabled
Macro file extension default:  pcm
Graphics hardcopy device: (bitmap) HPLASERJET 150 dpi
Colour:  WHITE    ( 1 )
World units type: CM
 Window   0 coordinates:   0.00   0.00 100.00 100.00 percentages
                           0.00   0.00  27.94  21.59 centimeters
 Window   0 coordinates:   0.00   0.00 100.00 100.00 percentages
                           0.00   0.00  27.94  21.59 centimeters
```

SURFACE

Syntax	SURFACE m { xangle { zangle { vsf { psf }}}}
Qualifiers	\COLOUR, \NORMAL, \HISTOGRAM, \XZLINES, \XLINES, \ZLINES
Defaults	\NOCOLOUR, \NORMAL, \XZLINES xangl $= -45°$, zangl $= -45°$, vsf $= 1$, psf $= 0$
Examples	SURFACE M SURFACE\COLOUR M SURFACE\HIST M -30 -60 SURFACE\NORMAL\ZLINES M -45 -45 2

The SURFACE command displays a matrix in the form of a 3-dimensional figure which can be rotated in space. Figure 2.26 shows the coordinate system used in this command. The appearance of the figure is controlled by optional qualifiers and parameters.

Note: Unlike most other commands, once one of the optional parameters has been changed to a new value, that value becomes the default.

The qualifiers that control the appearance of the figure are:

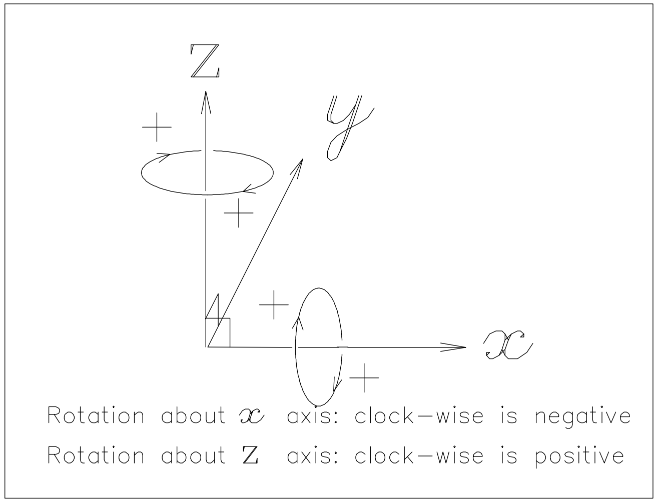


Figure 2.26: Surface coordinate system

qualifier	*results*
`\NORMAL`	a normal surface plot (default)
`\HISTOGRAM`	a histogram surface
`\XZLINES`	lines are drawn parallel to the x- and the z-axis (default)
`\XLINES`	lines are drawn parallel to the x-axis only
`\ZLINES`	lines are drawn parallel to the z-axis only

The parameters that control the appearance of the figure are:

parameter	*results*
`xangl`	rotation angle about the x-axis (default $-45°$)
`zangl`	rotation angle about the z-axis (default $-45°$)
`vsf`	vertical scale factor (default 1)
`psf`	pedestal size scale factor (default 0)

Colour

If the `\COLOUR` qualifier is used, the height of the surface will be divided into seven (7) regions

and a different colour associated with each region. This allows the user to quickly identify areas of similar height.

Examples

The following script produced Figure 2.27.

```
R=MOD([0:143],4)+1
SORT\UP R
T=MOD([0:143],36)*10
GRID R*COSD(T) R*SIND(T) EXP(-R/2)*COSD(180*(R-1)) M
SURFACE M -15 -45 1 1.5
```

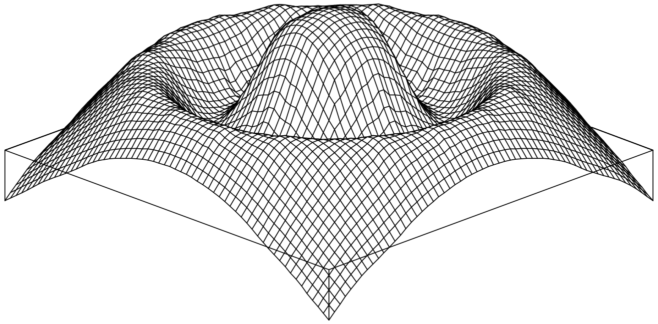


Figure 2.27: A `SURFACE` example

TERMINAL

Syntax	`TERMINAL { 'string' }`
Default	`'string'` = `'type <RETURN> to continue'`
Examples	`TERMINAL`
	`TERMINAL 'display this message'`

If the `TERMINAL` command is encountered in a script file that is being executed, control is passed back to the terminal keyboard. Commands can then be entered interactively. To continue execution of the script file, type the `<RETURN>` key without entering anything. The script file will then continue execution with the command immediately following the `TERMINAL` command.

By default, the message `'type <RETURN> to continue'` will be displayed. You can specify the message by entering a string with the command.

TEXT

Syntax `TEXT { ‘string’ }`

Qualifiers `\GRAPH, \ERASE, \CONFIRM`

Defaults `\CONFIRM, \NOGRAPH, \NOERASE`

Examples

```
TEXT\NOCONFIRM ‘This is a text string’
TEXT ‘The value of A is ’//RCHAR(A)//‘ and B = ’//RCHAR(B)
TEXT ‘<alpha,^>2<_,gamma>=<Phi>’
TEXT TIME//‘ ’//DATE
TEXT\ERASE\NOCONFIRM
```

By default, the TEXT command draws strings. The ‘string’ defaults to the last string that was entered with the TEXT command. Use the SET FONT command to set the font. The default font is TSAN.

Confirmation

If the TEXT command is entered interactively, the default is to request confirmation as to whether the text is acceptable as drawn. If you answer NO, the text will not appear on any hardcopies, or in an EDGR file. The text will not be entered into the hardcopy plot file, or into an open EDGR file, unless confirmed.

If the DISABLE CONFIRM command is entered before the TEXT command, the default will be that no such confirmation will be requested. The initial default is for CONFIRM to be enabled.

If the \CONFIRM qualifier is used, the default, confirmation will be forced. If the \NOCONFIRM qualifier is used, confirmation will be suppressed.

If the TEXT command is encountered in a script file, no confirmation is requested, even if ENABLE CONFIRM has been entered and the \CONFIRM qualifier is used.

Stack files

If a stack file is open, via the STACK command, page 247, then the (x, y) coordinates of the text location will be written to the stack file, even if they are chosen by the graphics cursor. When the stack file is executed, using the EXECUTE command, page 76, the graphics cursor will not be used.

If confirmation is requested and the figure is not acceptable, then the command will *not* be written to the stack file.

Text characteristics

The font can be changed with the SET FONT command, page 228. The default font is TSAN.

The text height can be controlled with the keywords TXTHIT or %TXTHIT, using the SET command. The default is %TXTHIT $= 3$.

The text angle can be controlled with the keyword TXTANG, , using the SET command. The default is TXTANG $= 0$.

Justification and location

The justification of the text is controlled by the keyword CURSOR. See Table 2.64 and Table 2.65. The origin of the text is always the lower left corner of the string. The justification determines where this origin is placed with respect to a reference point, see Figure 2.28. The default value of CURSOR is $+1$.

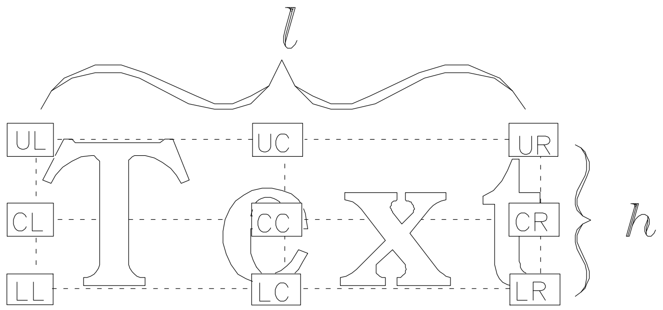


Figure 2.28: Text extent rectangle with two-character justification codes

If CURSOR ≤ 0, XLOC and YLOC will be used to determine the reference point for the text, and the justification is determined by the value of CURSOR.

If CURSOR > 0, the reference point is determined with the graphics cursor. A menu is available by typing key M, see Table 2.65. The justification of the text may be selected by typing key J and then selecting a justification, CC, LU, etc., or by simply typing one of the special justification keys, R, L, C, U, D, or V. No carriage return is needed after these special keys.

The value of CURSOR will be updated to the value corresponding to the justification that was chosen. For example, if CURSOR is currently 5 and the user chooses the justification code CV, then the value of CURSOR is changed to 6. The values of %XLOC and %YLOC will also be updated to the graphics cursor position that was chosen.

Text Formats

Commands

CURSOR	Justification			
	(x_0, y_0)	= origin point of string (lower left corner)		
	(x_{ref}, y_{ref})	= reference point (x_{ref} = XLOC, y_{ref} = YLOC)		
	h	= maximum height of string (default = TXTHIT)		
	l	= length of string in world coordinates		
	a	= angle of string in degrees (default = TXTANG)		
> 0.0	See the following table			
0.0 or -1.0	LL	$x_0 = x_{ref}$	$y_0 = y_{ref}$	
-2.0	LC	$x_0 = x_{ref} + l/2$	$y_0 = y_{ref}$	
-3.0	LR	$x_0 = x_{ref} + l$	$y_0 = y_{ref}$	
-4.0	LU	$x_0 = x_{ref}$	$y_0 = y_{ref}$	$a = 90°$
-5.0	LD	$x_0 = x_{ref}$	$y_0 = y_{ref}$	$a = 270°$
-6.0	CV	$x_0 = x_{ref} + l/2$	$y_0 = y_{ref}$	$a = 90°$
-7.0	CL	$x_0 = x_{ref}$	$y_0 = y_{ref} + h/2$	
-8.0	UL	$x_0 = x_{ref}$	$y_0 = y_{ref} + h$	
-9.0	CC	$x_0 = x_{ref} + l/2$	$y_0 = y_{ref} + h/2$	
-10.0	UC	$x_0 = x_{ref} + l/2$	$y_0 = y_{ref} + h$	
-11.0	CR	$x_0 = x_{ref} + l$	$y_0 = y_{ref} + h/2$	
-12.0	UR	$x_0 = x_{ref} + l$	$y_0 = y_{ref} + h$	

Table 2.64: Text justification interaction with CURSOR

key typed	justification with respect to reference point at crosshair location when CURSOR > 0.0
M	display the menu
Q	quit, do not draw any text
/	clear the alpha-numeric terminal screen, but not the graphics
J	a menu of justifications will be displayed. To draw the text string, choose one of the following two-character codes and type the RETURN key

Two character code	Justification chosen	
LL	lower left	(CURSOR set to 1)
CL	centre left	(CURSOR set to 7)
UL	upper left	(CURSOR set to 8)
LC	lower centre	(CURSOR set to 2)
CC	center centre	(CURSOR set to 9)
UC	upper centre	(CURSOR set to 10)
LR	lower right	(CURSOR set to 3)
CR	centre right	(CURSOR set to 11)
UR	upper right	(CURSOR set to 12)
LU	lower left at 90°	(CURSOR set to 4)
LD	lower left at 270°	(CURSOR set to 5)
CV	lower centre at 90°	(CURSOR set to 6)

key typed	justification
L	lower left (LL)
C	lower centre (LC)
R	lower right (LR)
U	lower left with an angle of 90° (LU)
D	lower left with an angle of 270° (LD)
V	lower centre with an angle of 90° (CV)
X	lower left (LL); using the y location selected by the crosshair and the x location that is stored in XLOC
Y	lower left (LL); using the x location selected by the crosshair and the y location that is stored in YLOC
other	use current value of CURSOR for justification use the crosshair position for the reference point

Table 2.65: Text menu and justification

The string may contain format commands and special characters which are included inside the string, and must be bracketed by the command delimiters, < and >. The special characters include all of the greek letters as well as some math symbols and other symbols. The names for the special characters can be seen by entering the DISPLAY SPECIAL command, page 58. See Figure 2.6 on page 61. The format commands are listed in Table 2.66.

bolding	<Bn>
colour	<Cn>
font	<Ffontname>
height	<Hnn.n> or <Hnn.n%>
sub-scripts	<_>
super-scripts	<^>
emphasis	<em>
hexadecimal input	<X>
vertical spacing	<Vnn.n> or <Vnn.n%>
horizontal spacing	<Znn.n> or <Znn.n%>

Table 2.66: TEXT command text formatting commands

When <NOD> is included in a string, the bolding, colour, font, emphasis and hexadecimal mode will be left in their current state. This is a way of changing the defaults for these characteristics.

When <DEF> is included in a string, at the end of processing that string, bolding will be turned off, the colour will be reset to the colour chosen by the COLOUR command, the font will be reset to the font chosen by the SET FONT command, slant mode will be turned off, and hexadecimal mode will be turned off. This is the default action, so it is not necessary to include <DEF> in a string. It has been included for completeness.

Refer to **Appendix A** for explanations of CURSOR, XLOC, YLOC, TXTHIT, TXTANG.

Replotting text

When text strings are added to a graph, they will be replotted, along with any data curves, with the REPLOT command, page 208. The default is to store the text locations in world coordinates, that is, centimeters or inches.

If you want the text locations to be recorded in graph units, use the \GRAPH qualifier. In this case, the text will be replotted in the same location in terms of the graph, so that if you label a curve, the label will follow the curve on the new graph scales. However, the height of the text will be increased or decreased proportionally. To restore the text height after a REPLOT, you can get, using the GET command, the value of %TXTHIT before REPLOT and re-set %TXTHIT

after the REPLOT, using the SET command.

Drawing the Date and Time

The DATE function returns the current date as a string. The TIME function returns the current time as a string. For example:

```
=DATE
‘16-APR-1993’
=TIME
‘11:58:16’
```

So the current date and time could be drawn with: TEXT TIME//‘ ’//DATE

Erasing text

Text can only be erased from the monitor screen and from a bitmap. Erasing will *not* work with plotter files and it will *not* work with EDGR. Erasing of rectangular regions is possible with PostScript graphics using the ERASEWINDOW command, page 76.

If the \ERASE qualifier is used, the last text drawn will be erased. If the TEXT\ERASE command is issued again, the next to last text drawn will be erased. And so on.

If the \NOCONFIRM qualifier is used, no confirmation will be requested. If the \CONFIRM qualifier is used, the string to be erased will be displayed on the screen, and you will be asked if this is the string to be erased. If the answer is NO, then the next string in reverse order of drawing is displayed, and so on. The default is \CONFIRM.

Example

The following script produced Figure 2.29.

```
SET
 %XLOC 50     ! x reference location
 %YLOC 80     ! y reference location
 CURSOR -2    ! lower centre justify at (x,y) reference location

! preceding blank line is necessary
TEXT ‘This is a simple text string’
SET  %YLOC 65
TEXT ‘<h10%,fscript.2>He<h8%>igh<h5%>t, font <alpha,beta,aleph>’
SET  %YLOC 50
TEXT ‘<B22,froman.serif,h5%>DOT FILLED CHARACTERS’
SET  %YLOC 35
TEXT ‘<INTEGRL>xdx = x<^>2<_>/2’
```

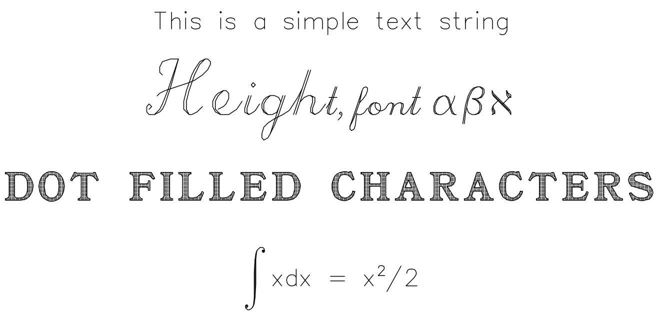


Figure 2.29: An example using the TEXT command

TILE

Syntax `TILE file`

The `TILE` command draws complicated bar graphs. The numbers which define the rectangular bars are read from the `file`. Strings may also be drawn. Comment lines are ignored, where a comment line is any line that begins with a `!`.

Bar definitions

A bar is defined by the following values:

parameter	*result*
`xlower`	the x coordinate of the left edge of the bar, in graph units
`xupper`	the x coordinate of the right edge of the bar, in graph units
`ymid`	the y coordinate of the middle of the bar, in graph units
`nh`	the dot or hatch pattern with which to fill the bar $\lvert$`nh`$\rvert \geq 11$ then fill with grey scale dot pattern $1 \leq \lvert$`nh`$\rvert \leq 10$ then fill with hatch pattern $\lvert$`nh`$\rvert = 0$ then no filling `nh` < 0 then the outline of the bar is not drawn, but hatch pattern $\lvert$`nh`$\rvert$ is drawn
`height`	the height of the bar, in graph units
`colour`	the colour of the bar outline and fill pattern

Hatch patterns

A hatch pattern is composed of an angle and one to ten spacings. These spacings are simply cycled through as the region is being filled, that is, a line is drawn inside the region at the specified angle, then a parallel line is drawn at the first spacing, then another parallel line is drawn at the second spacing, and so on for the number of spacings in that pattern. This process is repeated until the region is filled. The hatch patterns can be redefined with the `SET HATCH` command and displayed with the `DISPLAY FILL` command. There are ten hatch patterns available.

Dot fill patterns

A dot pattern is of the form: uv, where the digit u is the increment number of dots to light up horizontally, $1 \leq u \leq 9$, and the digit v is the increment number of dots to light up vertically, $1 \leq v \leq 9$. For example, a dot pattern of 34 means to light up every third dot horizontally and every fourth dot vertically. If uv is negative, then the dots are erased instead of turned on.

PostScript output

For PostScript output, set the `POSTRES` keyword to the appropriate resolution for your hardcopy device, using the `SET` command. Use a combination of line thickness and resolution for a quicker resulting picture. For example, the Lexmark inkjet printer has a resolution of 360 dpi. A “good” picture can be obtained with `POSTRES` $= 180$ and `LINTHK` $= 2$.

String definitions

A string is defined by the following values:

parameter	*result*
xlower	the x coordinate of the lower left corner of the string, in graph units
just	the justification to use for positioning the text, L → left, C → centre, R → right
ymid	the y coordinate of the middle of the text
font	the name of the font to use for drawing the text
height	the height of the text, in graph units
colour	the colour of the text (1 – 8), see the COLOUR command for an explanation of the colour code
'text'	the text itself, enclosed in quotes

Example

Suppose you have a file, TILE.DAT, as below:

```
!
!xlo  xup ymid nfill  height colour
!
 0    4   1     2     .8      1
 5   10   1    33     .8      2
 0    3   2    44     .8      3
 4    6.5 2    55     .8      4
 7    9   2    10     .8      5
 0    2   3     6     .8      2
 3    5   3     8     .8      3
 6    8   3    66     .8      5
 8.5 10   3    77     .8      1
!
!xlo just ymid  font  height colour  text
!
 -2    R    2  TSAN        .3      1 'Second label'
 -2    R    1  TRIUMF.2    .3      1 'First label'
 -2    R    3  ROMAN.SERIF .3      1 'Third label'
```

The following script produced Figure 2.30.


```
SCALES -10 12 11 0 4 4    ! manually set axes scales
TILE TILE.DAT             ! draw the tiles
SET
  BOX 0                   ! turn off axis box
  YAXIS 0                 ! turn off y-axis
  NSXINC 2                ! number of small x increments = 2

GRAPH\AXES                ! draw axes only
SET
  XAXIS 0                 ! turn off x-axis
  YAXIS 1                 ! turn on y-axis
  YITICA -90              ! draw numbers on right side of y-axis
  YTICA -90               ! draw tics on right side of y-axis
  %XLAXIS 95              ! draw y-axis at right end of x-axis

GRAPH\AXES                ! draw axes only
```

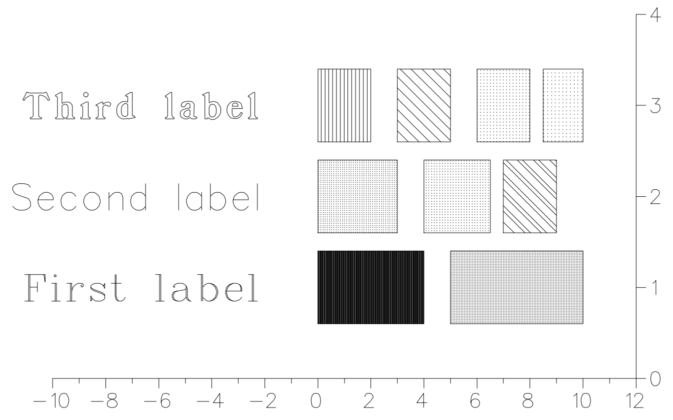


Figure 2.30: An example using the TILE command

TLEN

Syntax TLEN txt lenout

The TLEN command returns a scalar, lenout, which is the number of string elements of the array string variable, txt.

Example

Suppose that: `T[1]='string 1'; T[2]='string 2'; T[3]='string 3'`
then `TLEN T L` would return the scalar `L = 3`.

TRANSFORM

Syntax `TRANSFORM x p f(x,p) y`

Example `TRANSFORM X P EXP(-P*X^2) XT`

The TRANSFORM command performs integral transforms of the function f(x,p). The function is integrated over the entire range of definition, as defined by the independent variable, vector x. The integral is evaluated for each value of the parameter vector p, with the result stored in the vector y.

$$y_i = \int f(x, p_i) dx$$

The function f(x,p), which forms the integrand of the transform integral, may contain other scalars, or other vectors dimensioned identically to x. Such vectors are considered, and integrated, as functions of the independent variable x. Vectors of mismatched size will generate an error.

This command can be used to perform standard transforms such as Laplace or Hankel transforms. Fourier transforms should be done using the FFT function, as this is much faster.

Example

The following script integrates the expression e^{-px^2} over $x = [1:20]$ for each $p = [1:3]$, and stores the results in output vector F.

```
X=[1:20]
P=[1:3]
TRANSFORM X P EXP(-P*X^2) F
```

after which: `F=[0.1601404;0.05141661;0.01877862]`

A check on this result is the following:

```
FI[1]=(INTEGRAL(X,EXP(-P[1]*X^2)))[LEN(X)]
FI[2]=(INTEGRAL(X,EXP(-P[2]*X^2)))[LEN(X)]
FI[3]=(INTEGRAL(X,EXP(-P[3]*X^2)))[LEN(X)]
```

after which: `FI=[0.1601404;0.05141661;0.01877862]`

UNIQUE

Syntax `UNIQUE x y xout yout`
`UNIQUE\INDICES x y idx { (index_expression) }`

Qualifiers `\MESSAGES, \INDICES`

Defaults `\MESSAGES, \NOINDICES`

The UNIQUE command eliminates adjacent duplicate points in the vectors x and y. A duplicate point means that x[i] = x[i+1] and y[i] = y[i+1].

The vectors xout and yout are created, and will contain no adjacent duplicate points. If none are found, xout = x and yout = y.

To disable informational messages from the UNIQUE command, use the \NOMESSAGES qualifier.

Indices

Syntax `UNIQUE\INDICES x y idx { (index_expression) }`

The UNIQUE\INDICES command finds adjacent duplicate points in the vectors x and y. A duplicate point means that x[i] = x[i+1] and y[i] = y[i+1]. The vector idx is created, and will contain the indices of x and y where there are no adjacent duplicate points. If none are found, idx will not be made.

If you wish to search a subset of x and y, use the optional parameter (index_expression). Enclosing the index_expression in parenthesis allows you to have blanks within the expression. The index_expression can be a simple range, for example, [1:10], or a mathematical expression, for example, where(z=n). The index_expression can be used directly on the input vectors x and y, but then the output idx index vector will contain indices relative to [1:len(x[index_expression])].

Examples

Suppose that:

```
X=[1;2;2;3;4;5;5;7;8;9;9;10]
Y=[-6;-5;-4;-3;-2;-1;-1;1;2;3;3;4]
```

After the command: `UNIQUE X Y XO YO`

```
X= [ 1; 2; 2; 3; 4; 5; 5;7;8;9;9;10]
Y= [-6;-5;-4;-3;-2;-1;-1;1;2;3;3; 4]
XO=[ 1; 2; 2; 3; 4; 5;   7;8;9;  10]
YO=[-6;-5;-4;-3;-2;-1;   1;2;3;   4]
```

After the command: `UNIQUE\INDICES X Y IXY`

```
X=  [ 1; 2; 2; 3; 4; 5; 5;7;8; 9;9;10]
Y=  [-6;-5;-4;-3;-2;-1;-1;1;2; 3;3; 4]
IXY=[ 1; 2; 3; 4; 5; 6;   8;9;10;  12]
```

Suppose that:

```
X = [ 1; 2; 3; 3; 5; 6; 7; 8; 9;10; 1; 2; 3; 4; 5; 5; 7; 8; 9;10]
Y = [-1;-2;-3;-3;-5;-6;-7;-8;-9;-1;-1;-2;-3;-4;-5;-5;-7;-8;-9;-1]
Z = [ 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 2; 2; 2; 2; 2; 2; 2; 2; 2; 2]
```

After the command: `UNIQUE\INDICES X Y IDX`

`IDX = [1;2;3;5;6;7;8;9;10;11;12;13;14;15;17;18;19;20]`

After the command: `UNIQUE\INDICES X[WHERE(Z=2)] Y[WHERE(Z=2)] IDX`

`IDX = [1;2;3;4;5;7;8;9;10]`

After the command: `UNIQUE\INDICES X Y IDX WHERE(Z=2)`

`IDX = [11;12;13;14;15;17;18;19;20]`

USE

Syntax `USE filename`

The `USE` command causes program input to come from a file, instead of from the keyboard. When the end of file is reached, input will again be expected to be entered from the keyboard. Nesting is not allowed. Within `USE` files, labels, `GOTO` statements, `IF` statements and blocks, and `DO` loops are not allowed.

This command should be useful for entering blocks of commands when the `TERMINAL` command has been encountered while executing a macro script file, since another script cannot be executed from that mode.

The default file extension is `.PCM` If the filename is a text variable, it is first replaced by its value.

VMS: If `filename` is a logical name, it is replaced by its translation.
If `filename` does not contain a file name extension, the default file name extension is appended to filename.
If filename doesn't exist in the current directory, and if the logical name `PHYSICA$LIB` has been defined, that location is checked.

UNIX: If `filename` is an environment variable, it is replaced by its translation.
If filename cannot be found in the current location, `filename` with the default file extension appended is checked. If this file cannot be found, and if the environment variable `PHYSICA_LIB` has been defined, then `PHYSICA_LIB/filename` is tried.
If this file cannot be found, then `PHYSICA_LIB/filename` with the default file name extension appended is tried.

Environment variables in file names

For UNIX users, it is now possible to use an environment variable in a file name, if the environment variable is preceeded by a `$`. For example,

```
setenv FILE mystuff
physica
use $FILE
```

The environment variable can be just the first part of the filename, for example,

```
setenv FILE my
physica
use $FILEstuff
```

VECTOR

Syntax `VECTOR x1 { x2 ... } n`

The `VECTOR` command creates new vectors or changes the lengths of existing vectors.

If `x1` is already a vector, it will be either trimmed down to the specified length or zero filled to expand it. If `x1` exists but is not a vector, it will be destroyed first. If `x1` does not exist, it will be created as a zero filled vector with length `n`. Other variable names, `xI`, may be entered. They will be treated just like `x1`.

VOLUME

Syntax `VOLUME x y z { volm }`
`VOLUME\MATRIX { x y } m { volm }`

Qualifiers `\MATRIX`, `\POLAR`

Defaults `\NOPOLAR`, `x=[1;2;...]`, `y=[1;2;...]`

By default, the `VOLUME` command calculates the volume under the scattered points given by vectors `x`, `y` and `z=f(x,y)`. The output is an optional scalar, `volm`. If `volm` is not entered, the value is simply displayed. The region given by `x` and `y` is first triangulated, using a Thiessen triangulation, then the integral is approximated by integrating the piecewise linear interpolants of the data values.

If the `\POLAR` qualifier is used, the data is assumed to be in polar coordinates with `x` the radial components and `y` the angular components, in degrees.

Volume under a matrix

Syntax `VOLUME\MATRIX { x y } m { volm }`

Defaults `x=[1;2;...]`, `y=[1;2;...]`

If the `\MATRIX` qualifier is used, a matrix is expected. The grid region is triangulated. The volume is approximated by summing the averages of the matrix values at the triangle vertices multiplied by the area of each triangle. If `x` and `y` are not entered then `x` defaults to `[1;2;...;ncol]` and `y` defaults to `[1;2;...;nrow]`, where `ncol` is the number of columns of the matrix `m` and `nrow` is the number of rows. The `\POLAR` qualifier is not allowed with matrix data.

WAIT

Syntax `WAIT n`

The `WAIT` command causes a delay of `n` seconds. This is intended for use when `EXECUT`ing a command script file.

Suppose some message is to be displayed on the terminal screen, via the `DISPLAY` command, and the user is expected to interactively enter some information. One could ring the bell, using the `BELL` command, display the message, and then cause a wait of a few seconds before clearing the screen. The user would then have a better opportunity to read the message and then act on it.

Window

Syntax `WINDOW { n { lowx lowy { upx upy }}}`
`WINDOW\TILE nx ny nstart`

Qualifiers `\TILE`, `\MESSAGES`

Defaults `\NOMESSAGES`, `nstart` $= 1$

The `WINDOW` command chooses a subset of the graphics page. The default window is the entire current graphics page, that is, window number zero (0).

If the `WINDOW` command is entered with no parameters, a listing of the currently defined windows will be displayed, along with their corner coordinates.

Use the `\NOMESSAGES` qualifier to turn off the display of informational messages to the terminal screen.

What are windows

Windows are an easy way to subdivide the graphics output page into rectangular regions, allowing multiple graphs and/or multiple figures and/or multiple text regions. A window is a subset of the page. A window, other than the default zero level window, has a smaller plotting unit range than the full page.

Commensurateness is never lost in a sub-window. Windows are transparent to EDGR.

Boundaries

There is usually at least one rectangle drawn on the monitor screen. The largest rectangle represents the world boundary, that is, the maximum extent of the hardcopy page. A smaller, inner, rectangle, drawn with a dashed line, represents a sub-window within the page. These rectangles are for the user's reference only and will not appear on a hardcopy. These boundary rectangles can be turned off with the `DISABLE BORDER` command, page 55.

Plotting units

Sub-windows have different plotting unit ranges than the full page. Thus, when a sub-window has been selected, it is possible that not all subsequent graphics will be contained within this window. However, commensurateness is always preserved within a sub-window. For example, circles, which appear as circles when drawn into the full page, will still be circular when drawn into any sub-window.

Defining a new window

Syntax `WINDOW n { lowx lowy { upx upy }}`

Window zero, the full page or world, cannot be redefined.

A new window can be defined by including the optional final four parameters with the command. The lower left corner will be `(lowx,lowy)` and the upper right corner will be `(upx,upy)`. These four parameters should be expressed as percentages of the full page, for example, `lowx` = 50, `lowy` = 50 represents the centre of the world, while `lowx` = 100, `lowy` = 100 represents the upper right hand corner.

If a window number, `n`, is entered but not the final four parameters, and that window number is undefined, then the graphics cursor will be used to choose the lower left and upper right corners of the new window.

Pre-defined windows

The initial pre-defined windows are displayed in Table 2.67. See also Figure 2.31.

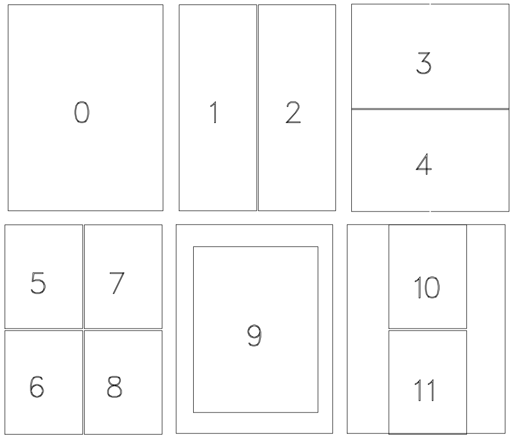


Figure 2.31: The initial pre-defined windows in `PORTRAIT` orientation

Windows and GPLOT

Choosing a window with the `WINDOW` command, which has its lower left corner at `(lowx,lowy)` and its upper right corner at `(upx,upy)`, is equivalent to choosing the GPLOT window with the `SET` command. That is, the following command:

	percentages				centimeters			
window number	lowx	lowy	upx	upy	lowx	lowy	upx	upy
0	0	0	100	100	0.00	0.00	25.00	19.00
1	0	0	50	100	0.00	0.00	12.50	19.00
2	50	0	100	100	12.50	0.00	25.00	19.00
3	0	50	100	100	0.00	9.50	25.00	19.00
4	0	0	100	50	0.00	0.00	25.00	9.50
5	0	50	50	100	0.00	9.50	12.50	19.00
6	0	0	50	50	0.00	0.00	12.50	9.50
7	50	50	100	100	12.50	9.50	25.00	19.00
8	50	0	100	50	12.50	0.00	25.00	9.50
9	10	10	90	90	2.50	1.90	22.50	17.10
10	25	50	75	100	6.25	9.50	18.75	19.00
11	25	0	75	50	6.25	0.00	18.75	9.50
15	0	75	50	100	0.00	14.25	12.50	19.00
16	50	75	100	100	12.50	14.25	25.00	19.00
17	0	50	50	75	0.00	9.50	12.50	14.20
18	50	50	100	75	12.50	9.50	25.00	14.20

Table 2.67: The initial pre-defined windows

```
WINDOW n lowx lowy upx upy
```

is equivalent to the following:

```
SET
 %XLWIND lowx
 %YLWIND lowy
 %XUWIND upx
 %YUWIND upy
```

Using the `SET` command to define a window is *not* recommended. When the `CLEAR` command is entered, the window chosen with the last `WINDOW` command is set up, and the values of `XLWIND`, `XUWIND`, `YLWIND`, and `YUWIND` are changed. Thus, if their values had been re-defined with the `SET` command, these values would be lost when the `CLEAR` command is entered.

Multiple window creation

Syntax `WINDOW\TILE nx ny { nstart }`
Default `nstart` = 1

The WINDOW\TILE command divides the graphics page up into nx horizontal by ny vertical windows, giving the first window the number nstart, which must be > 0. The total number of windows just defined will be nx*ny. The window numbered nstart will be in the upper left corner of the page, while the window numbered nx*ny-nstart+1 will be in the lower right corner.

WORLD

Syntax	`WORLD vxin vyin { vxout vyout }`
Qualifiers	`\PERCENT`
Defaults	absolute coordinates returned

The WORLD command converts graph coordinates into world coordinates, or into percentages if the \PERCENT qualifier is used.

The world coordinates are either centimeters or inches depending on the units chosen with the SET UNITS command. The default units are centimeters.

The variables vxin and vyin may be scalars or vectors, but they must both be the same type. The resultant values are displayed on the terminal screen. If output variables vxout and vyout are present, the values are stored there. The type of variable created depends on the variable type of vxin and vyin.

WRITE

Syntax
```
WRITE file { (format) } x1 { x2 ... }
WRITE\SCALAR file s1 { s2 ... }
WRITE\MATRIX file matrix
WRITE\TEXT file txtvar
```

Qualifiers `\SCALAR, \MATRIX, \TEXT, \FORMAT, \APPEND`

Examples
```
WRITE FILE.DAT X Y Z
WRITE\FORMAT FILE.DAT ('X=',F10.3,' Y=',F7.1,' Z=',F9.2) X Y Z
WRITE\APPEND FILE.DAT X Y Z

WRITE\SCALAR FILE.DAT X[2] Y[3] M[2,4]
WRITE\SCALAR\FORMAT FILE.DAT ('String ',3(F10.3)) A Y[3] C

WRITE\MATRIX\APPEND FILE.DAT M[1:100,1:10]
WRITE\MATRIX\FORMAT FILE.DAT (7F10.3,2X) M[1:100,1:10]

WRITE\TEXT FILE.DAT `String'
WRITE\TEXT FILE.DAT T3
WRITE\TEXT\APPEND FILE.DAT `A = '//RCHAR(A,`F10.3')
```


WRITE is a general purpose command for writing vectors, scalars, a matrix, or a string. The variable type that will be written is determined by command qualifier. The parameters that are expected also depend on this qualifier. By default, the WRITE command writes vectors to a file.

Environment variables in file names

For UNIX users, it is now possible to use an environment variable in a file name, if the environment variable is preceeded by a $. For example,
```
setenv FILE dum.dat
physica
write $FILE x y z
```
The environment variable can be just the first part of the filename, for example,
```
setenv FILE dum
physica
write $FILE.dat x y z
```

Appending to a file

By default, a new file is opened to receive the output data. If the \APPEND qualifier is used, and if the output file already exists, the data will be appended onto the end of the file.

Formats

Syntax
```
WRITE\FORMAT file (format) x1 { x2 ... }
WRITE\SCALAR\FORMAT file (format) s1 { s2 ... }
WRITE\MATRIX\FORMAT file (format) matrix
```

By default, free format is used to write the data to the file. You must use the \FORMAT qualifier to indicate a format is present. The format must be enclosed in parentheses, (and). Any standard FORTRAN format is valid, but only REAL variables can be written. Do not use INTEGER, LOGICAL or CHARACTER formats.

You may use the PHYSICA defined format BINARY, minimum abbreviation B, to write binary unformatted files containing 8 byte numbers.

Vectors

The default is to write vectors. The WRITE command writes the vectors xI to a file. The minimum of the lengths of the vectors will be used. A maximum of twenty-nine (29) vectors can be written with one WRITE command.

If the same vector name is entered more than once, consecutive elements of that vector will be written to the same record. For example, if you enter:

```
WRITE FILE.DAT X X X
```

the following data will be written to the file:

```
X[1] X[2] X[3]
X[4] X[5] X[6]
 ...
```

or, if you enter: `WRITE FILE.DAT X Y X Y X Y`

the following data will be written to the file:

```
X[1] Y[1] X[2] Y[2] X[3] Y[3]
X[4] Y[4] X[5] Y[5] X[6] Y[6]
 ...
```

Examples

To write the vectors X, Y and Z to the file DUM.DAT using free format, that is, there will be three columns of numbers in the file, enter:

```
WRITE DUM.DAT X Y Z
```

To write the vectors X, Y and Z to DUM.DAT using a specified format, enter:

```
WRITE\FORMAT DUM.DAT (' X=',F6.2,' Y=',F6.2,' Z=',F6.2) X Y Z
```

and DUM.DAT will look like:

```
X=123.45 Y=-23.45 Z= 23.45
X=124.45 Y=-24.45 Z= 24.45
X=125.45 Y=-25.45 Z= 25.45
X=126.45 Y=-26.45 Z= 26.45
...       ...       ...
```

Scalars

The WRITE\SCALAR command writes scalars to a file. One line will be written to the file. A maximum of twenty-nine (29) scalars can be written with one WRITE command.

Examples

To write scalars A and B to file FILE.DAT using some format, and then to append to the file the vectors X, Y and Z, enter:

```
WRITE\SCALAR\FORMAT FILE.DAT ('A = ',F10.3,' B = ',F10.3) A B
WRITE\APPEND FILE.DAT X Y Z
```

Matrix

The WRITE\MATRIX command writes a matrix to a file. Only one matrix can be written with each WRITE\MATRIX command.

String

The WRITE\TEXT command writes a string variable, or literal string, to a file. Only one text string can be written with each WRITE\TEXT command and only one record will be written.

Examples

Suppose you want to write a header line to a file and then write some data stored in vectors to that file. For example:

```
WRITE\TEXT FILE.DAT ‘This is a header line’
WRITE\APPEND FILE.DAT X Y Z
```

Suppose X is a vector, X=[1.1;2.2;3.3;4.4], and you want to write the values of X to a file, with some text. For example:

```
DO J = [1:LEN(X)]
 WRITE\TEXT\APPEND FILE.DAT ‘X['//RCHAR(J)//‘] = '//RCHAR(X[J],‘F4.1')
ENDDO
```

ZEROLINES

Syntax	ZEROLINES
Qualifiers	\HORIZONTAL, \VERTICAL
Defaults	both horizontal and vertical lines drawn

The ZEROLINES command draws horizontal and/or vertical lines on a graph through $(0,0)$ depending on the qualifier that is used. The line(s) will have the current line type, as set with the SET LINTYP command.

qualifier	*result*
\HORIZONTAL	a line is drawn through $(0,0)$, parallel to the x-axis, and from the left edge, XMIN, to the right edge, XMAX
\VERTICAL	a line is drawn through $(0,0)$, parallel to the y-axis, and from the bottom edge, YMIN to the top edge, YMAX
none	both horizontal and vertical lines are drawn

Example

The following script produces Figure 2.32.

```
WINDOW 5                ! choose pre-defined window
SCALES -5 5 2 -5 5 2    ! set graph scales
GRAPH\AXESONLY          ! just draw axes
ZEROLINES               ! overlay horiz. and vert. lines
WINDOW 6
GRAPH\AXESONLY
ZEROLINES\HORIZONTAL    ! overlay horizontal line
WINDOW 12 50 0 75 50    ! define a window
GRAPH\AXESONLY
ZEROLINES\VERTICAL      ! overlay vertical line
```

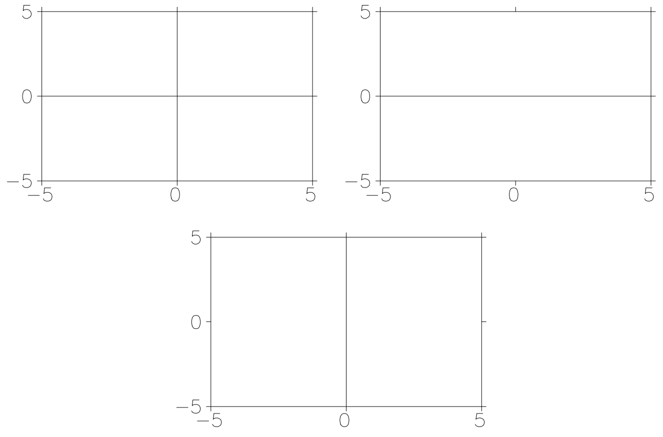



Figure 2.32: An example illustrating the ZEROLINES command

3 OPERATORS

Table 3.68 lists the Boolean operators that are recognized in PHYSICA expressions. Table 3.69 lists the other operators. Operators requiring some explanation are described individually in following sections.

string form	*symbolic form*	*description*
"OR"	|	or
"XOR"	||	exclusive or
"AND"	&	and
"NOT"	~	not
"LT"	<	less than
"GT"	>	greater than
"EQ"	<	equal
"LE"	<= or ~>	less than or equal
"GE"	>= or ~<	greater than or equal
"NE"	~=	not equal

Table 3.68: Boolean operators

operator	*description*	*operator*	*description*
( )	parentheses	^ or **	exponentiation
*	multiplication	/	division
+	addition	-	subtraction
><	outer product	<>	inner product
<-	matrix transpose	>-	matrix reflect
/|	vector union	/&	vector intersection
//	append		

Table 3.69: Other operators

Boolean operators

The Boolean operators, listed in Table 3.68, all come in two forms, symbolic and string. The double quotes, " ", are required with the string form. Boolean operators return a value of zero (0) when false and a value of one (1) when true.

The Boolean operators can operate on scalars, vectors, or matrices, but both operands must be the same size and shape. The result of the operation is a variable with this size and shape.

All of the Boolean operators are binary, except for the not operator, "NOT" or ~, which is unary.

Examples

Suppose you have two vectors: `X = [1;2;3;4;5;6;7]` `Y = [-2;-1;0;1;2;3;4]` Then:

```
X"OR"Y  =  X|Y   = [1;1;1;1;1;1;1]
X"XOR"Y =  X||Y  = [0;0;1;0;0;0;0]
X"AND"Y =  X&Y   = [1;1;0;1;1;1;1]
X"EQ"Y  =  X=Y   = [0;0;0;0;0;0;0]
X"NE"Y  =  X~=Y  = [1;1;1;1;1;1;1]
X"GT"Y  =  X>Y   = [1;1;1;1;1;1;1]
X"LT"Y  =  X<Y   = [0;0;0;0;0;0;0]
X"GE"Y  =  X>=Y  = [1;1;1;1;1;1;1]
X"LE"Y  =  X<=Y  = [0;0;0;0;0;0;0]
"NOT"(X"OR"Y) = ~(X|Y) = [0;0;0;0;0;0;0]
```

Transpose

The transpose operator, <-, is a unary operator that produces the transpose of a matrix, that is, the rows and columns are interchanged. Suppose the matrix m has n_r rows and n_c columns. $m[i,j] \longleftrightarrow m[j,i]$ for $i = 1, 2, \ldots, n_r$ and $j = 1, 2, \ldots, n_c$.

Example

$$\mathtt{M} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} \qquad \mathtt{<-M} = \begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix}$$

Reflect

The reflect operator, >-, is a unary operator that interchanges the columns of a matrix. Suppose the matrix m has n_c columns and n_r rows. Column n_c is interchanged with column 1, that is, $m[1:n_r,1] \longleftrightarrow m[1:n_r,n_c]$; column $n_c - 1$ is interchanged with column 2, that is, $m[1:n_r,2] \longleftrightarrow m[1:n_r,n_c - 1]$; and so on.

Examples

$$\texttt{M} = \begin{pmatrix} 1 & 10 & -1 \\ 2 & 20 & -2 \\ 3 & 30 & -3 \\ 4 & 40 & -4 \end{pmatrix} \qquad \texttt{>-M} = \begin{pmatrix} -1 & 10 & 1 \\ -2 & 20 & 2 \\ -3 & 30 & 3 \\ -4 & 40 & 4 \end{pmatrix}$$

You can reflect the rows of a matrix, by using the reflect operator and the transpose operator.

$$\texttt{M} = \begin{pmatrix} 1 & 10 & -1 \\ 2 & 20 & -2 \\ 3 & 30 & -3 \\ 4 & 40 & -4 \end{pmatrix} \qquad \texttt{<-(>-(<-M))} = \begin{pmatrix} -4 & 40 & 4 \\ -3 & 30 & 3 \\ -2 & 20 & 2 \\ -1 & 10 & 1 \end{pmatrix}$$

Union

The union operator, `/|`, is a binary operatore that only accepts vectors as operands, and returns a vector which contains the union of the elements of these two vectors.

All vectors have an order property. Vectors are either in ascending order, descending order, or un-ordered. The type is displayed in the `SHOW` command, where `+O` means ascending order, `-O` means descending order, and no symbol means un-ordered. For now, being ordered only has an affect on the vector union, `/|`, and the vector intersection, `/&`. These operations are much faster if the vector operands are ordered. The `WHERE` function produces an ascending order vector, as does the `SORT\UP` command. The `SORT\DOWN` command produces a descending order vector. This new vector property will be utilised more in the future to enhance speed and efficiency.

Example

To illustrate vector union, suppose you have two vectors:

`X = [1;2;3;4;5;6;7]`   `Y = [-2;-1;0;1;2]`

Then: `X/|Y = [-2;-1;0;1;2;3;4;5;6;7]`

Intersection

The intersection operator, `/&`, is a binary operatore that only accepts vectors as operands, and returns a vector which contains the intersection of the elements of these two vectors.

All vectors have an order property. Vectors are either in ascending order, descending order, or un-ordered. The type is displayed in the `SHOW` command, where `+O` means ascending order, `-O` means descending order, and no symbol means un-ordered. For now, being ordered only

has an affect on the vector union, `/|`, and the vector intersection, `/&`. These operations are much faster if the vector operands are ordered. The `WHERE` function produces an ascending order vector, as does the `SORT\UP` command. The `SORT\DOWN` command produces a descending order vector. This new vector property will be utilised more in the future to enhance speed and efficiency.

Example

To illustrate vector intersection, suppose you have two vectors:

`X = [1;2;3;4;5;6;7]`   `Y = [-2;-1;0;1;2]`

Then: `X/&Y = [1;2]`

Append

The meaning of the append operator, `//`, depends on its operands. If the operands are both vectors, the second vector is appended to the first. If the operands are strings, the second string is appended to the first.

Examples

To illustrate appending vectors, suppose you have two vectors:

`X = [3;5;7]`   `Y = [-2;-4;-5]`

Then: `X//Y = [3;5;7;-2;-4;-5]`

To illustrate appending strings, suppose you have a scalar string variable:

`T = ‘this is a string’`

Then: `T//‘ and another’ = ‘this is a string and another’`

Outer product

The outer product operator, `><`, operates on two vectors and produces a matrix composed of all possible combinations of products of elements of the vectors.

$$\text{If}\quad x = [x_1;x_2;\dots;x_m] \quad\text{and}\quad y = [y_1;y_2;\dots;y_n] \quad\text{then}\quad x\texttt{><}y = \begin{pmatrix} x_1y_1 & x_1y_2 & \dots & x_1y_n \\ x_2y_1 & x_2y_2 & \dots & x_2y_n \\ \vdots & & & \vdots \\ x_my_1 & x_my_2 & \dots & x_my_n \end{pmatrix}$$

Example

Suppose you have two vectors: `X = [1;3;5]` `Y = [2;4]`

Then: $\texttt{X><Y} = \begin{pmatrix} 2 & 4 \\ 6 & 12 \\ 10 & 20 \end{pmatrix}$

Inner product

The inner product, `<>`, operating on two vectors produces a scalar; operating on a vector and a matrix produces a vector, operating on two matrices produces a matrix.

Operating on two vectors

The inner product operating on two vectors produces a scalar, whose value is equal to the sum of the products of the vectors' elements. The two vectors *must* be the same length.

$$\text{If} \quad x = [x_1; x_2; \ldots ; x_n] \quad \text{and} \quad y = [y_1; y_2; \ldots ; y_n] \quad \text{then} \quad x \texttt{<>} y = \sum_{i=1}^{n} x_i y_i$$

Example

Suppose you have two vectors: `X = [1;3;5]` `Y = [2;4;6]`

Then: $\texttt{X<>Y} = 44$

Operating on a vector and a matrix

The inner product operating on a vector and a matrix produces a vector. If the vector is the first operand, its length *must* be equal to the number of rows of the matrix. The resultant vector length will be the number of columns of the second operand matrix.

$$\text{If} \quad x = [x_1; x_2; \ldots ; x_m] \quad \text{and} \quad a = \begin{pmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,n} \\ a_{2,1} & a_{2,2} & \ldots & a_{2,n} \\ \vdots & & & \vdots \\ a_{m,1} & a_{m,2} & \ldots & a_{m,n} \end{pmatrix}$$

$$\text{then} \quad (x \texttt{<>} a)_i = \sum_{j=1}^{m} x_j a_{j,i} \quad \text{for} \quad i = 1, 2, \ldots , n$$

Examples

The inner product of a vector and a matrix:

`X = [1;3;5]` $\quad \texttt{M} = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$

Then: $\texttt{X<>M} = [22; 49]$

Operating on a matrix and a vector

The inner product operating on a matrix and a vector produces a vector. If the vector is the second operand, its length *must* be equal to the number of columns of the matrix, and the resultant vector length will be the number of rows of the first operand matrix.

$$\text{If} \quad a = \begin{pmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,n} \\ a_{2,1} & a_{2,2} & \ldots & a_{2,n} \\ \vdots & & & \vdots \\ a_{m,1} & a_{m,2} & \ldots & a_{m,n} \end{pmatrix} \quad \text{and} \quad x = [x_1; x_2; \ldots ; x_n]$$

$$\text{then} \quad (a \texttt{<>} x)_i = \sum_{j=1}^{n} a_{i,j} x_j \quad \text{for} \quad i = 1, 2, \ldots , m$$

Examples

The inner product of a matrix and a vector:

`X = [1;3;5]`
$\texttt{M} = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$

Then: $\texttt{M<>X} = [35; 44]$

Operating on two matrices

The inner product operating on two matrices produces a matrix. The number of columns of the first operand matrix *must* be equal to the number of rows of the second operand matrix. The resultant matrix will be a square matrix with the number of rows and the number of columns equal to the number of rows of the first operand.

$$\text{If} \quad a = \begin{pmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,n} \\ a_{2,1} & a_{2,2} & \ldots & a_{2,n} \\ \vdots & & & \vdots \\ a_{m,1} & a_{m,2} & \ldots & a_{m,n} \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} b_{1,1} & b_{1,2} & \ldots & b_{1,m} \\ b_{2,1} & b_{2,2} & \ldots & b_{2,m} \\ \vdots & & & \vdots \\ b_{n,1} & b_{n,2} & \ldots & b_{n,m} \end{pmatrix}$$

$$\text{then} \quad (a\texttt{<>}b)_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j} \quad \text{for} \quad i = 1, 2, \ldots, m \quad \text{and} \quad j = 1, 2, \ldots, m$$

Example

The inner product of two matrices:

$$\texttt{A} = \begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{pmatrix} \text{ and } \texttt{B} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{pmatrix} \text{ Then: } \quad \texttt{A<>B} = \begin{pmatrix} 74 & 148 & 222 \\ 134 & 268 & 402 \\ 194 & 388 & 582 \end{pmatrix}$$

4 FUNCTIONS

PHYSICA supports many types of functions, both numeric and string. The basic numeric type of function operates on scalars, vectors, or matrices, but one number at a time. In other words, it performs its calculations on an element by element basis. These include the trigonometric functions, and the basic arithmetic functions such as the exponential and logarithmic functions. The resultant variable type of one of these element by element functions will be the same as the variable type of its argument.

PHYSICA also supports array functions, which operate on variables in their entirety, such as derivative, integral and smoothing functions. Some of these functions have a different resultant variable type than their arguments. There are also functions that operate on strings, such as case changing functions, and functions that have numeric arguments but result in strings.

A special type of the array functions are called looping functions, such as the sum and product functions. The looping functions all require a previously declared scalar dummy variable as second argument. The looping functions mimic standard mathematical notation, for example, the sum:

$$\texttt{SUM}(f(j), j, 1:N) \equiv \sum_{j=1}^{N} f(j)$$

Where j is the dummy variable. A dummy variable is different from other scalar variables in that its value is only defined while inside the looping function. The first argument of a looping function would normally be some function of the dummy variable, but it is not necessary that the dummy variable appear in the first argument.

Element by element functions

Table 4.70 on page 289 lists the trigonometric functions, while Table 4.71 on page 290 lists other basic numeric functions that are intrinsic to the program. These functions all expect numeric arguments, and operate on an element by element basis.

The trigonometric and basic single argument functions will always have a resultant type which will be the same as the type of its argument, that is, a scalar argument results in a scalar, a vector argument results in a vector with the same length as the argument, and a matrix argument results in a matrix with the same size and shape as the argument. Some of these element by element functions, such as `MAX`, `DIM` and `ELTIME`, do require some explanation, but the definitions of most are assumed to be obvious to the reader.

function	*description*	*function*	*description*
`SIN(x)`	Sine (radians)	`SINH(x)`	Hyperbolic Sine
`SIND(x)`	Sine (degrees)		
`COS(x)`	Cosine (radians)	`COSH(x)`	Hyperbolic Cosine
`COSD(x)`	Cosine (degrees)		
`TAN(x)`	Tangent (radians)	`TANH(x)`	Hyperbolic Tangent
`TAND(x)`	Tangent (degrees)		
`COT(x)`	Cotangent (radians)	`COTH(x)`	Hyperbolic Cotangent
`SEC(x)`	Secant (radians)	`SECH(x)`	Hyperbolic Secant
`CSC(x)`	Cosecant (radians)	`CSCH(x)`	Hyperbolic Cosecant
`ASIN(x)`	Arc Sine (radians)	`ASINH(x)`	Hyperbolic Arc Sine
`ASIND(x)`	Arc Sine (degrees)		
`ACOS(x)`	Arc Cosine (radians)	`ACOSH(x)`	Hyperbolic Arc Cosine
`ACOSD(x)`	Arc Cosine (degrees)		
`ATAN(x)`	Arc Tangent (radians)	`ATANH(x)`	Hyperbolic Arc Tangent
`ATAND(x)`	Arc Tangent (degrees)		
`ATAN2(y,x)`	Arc Tangent of `y` / `x` (radians)		
`ATAN2D(y,x)`	Arc Tangent of `y` / `x` (degrees)		
`ACOT(x)`	Arc Cotangent (radians)	`ACOTH(x)`	Hyperbolic Arc Cotangent
`ASEC(x)`	Arc Secant (radians)	`ASECH(x)`	Hyperbolic Arc Secant
`ACSC(x)`	Arc Cosecant (radians)	`ACSCH(x)`	Hyperbolic Arc Cosecant

Table 4.70: Trigonometric functions

function	*description*
`ABS(x)`	absolute value $\|x\|$
`EXP(x)`	exponential e^x
`FACTORIAL(x)`	factorial $x!$
`INT(x)`	integer portion of x
`NINT(x)`	nearest integer ($x + 0.5 \times \texttt{SIGN}(1, x)$)
`LOG(x)`	base e logarithm
`LN(x)`	base e logarithm
`LOG10(x)`	base 10 logarithm
`RAN(x)`	random number
`SQRT(x)`	square root $\sqrt{x}$
`ELTIME(x)`	elapsed time in seconds
`DIM(x,y)`	positive difference function
`MOD(x,y)`	modulus function
`SIGN(x,y)`	transfer of sign
`MAX(x1,x2,...)`	maximum of argument list
`MIN(x1,x2,...)`	minimum of argument list

Table 4.71: Basic element by element numeric functions

ATAN2

Syntax `vout = ATAN2( v1, v2 )`

The `ATAN2(v1,v2)` function returns the Arc Tangent of `v1/v2`, with $-\pi <$ `ATAN2(v1,v2)` $\leq \pi$.

- If `v1` > 0, the result is positive.
- If `v1` $= 0$, the result is zero if `v2` > 0 and π if `v2` < 0.
- If `v1` < 0, the result is negative. If the value of the second argument is zero, the absolute value of the result is $\pi/2$.

Both arguments must not have the value zero.

ATAN2D

Syntax `vout = ATAN2D( v1, v2 )`

The `ATAN2D(v1,v2)` function returns the Arc Tangent of `v1/v2` in degrees, with $-180° <$ `ATAN2(v1,v2)` $\leq 180°$.

- If $\texttt{v1} > 0$, the result is positive.
- If $\texttt{v1} = 0$, the result is zero if $\texttt{v2} > 0$ and $180°$ if $\texttt{v2} < 0$.
- If $\texttt{v1} < 0$, the result is negative. If the value of the second argument is zero, the absolute value of the result is $90°$.

Both arguments must not have the value zero.

RAN

Syntax `vout = RAN( v )`

The `RAN` function is an element by element function that requires one (1) argument. The argument can be a scalar, vector or matrix. A scalar argument results in a scalar. A vector argument results in a vector with the same length as the argument, and a matrix argument results in a matrix with the same dimensions as the argument.

`RAN` uses the current value of the seed to generate the next random number, $0 \le \texttt{RAN(v)} < 1$. The initial value for the random number seed is 12345. Every time a random number is requested, either from the `GENERATE\RANDOM` command or from the `RAN` function, the seed is updated. You can change the seed value with the `SET SEED` command.

ELTIME

Syntax `sout = ELTIME( s )`

The `ELTIME` function is an element by element function that requires one (1) argument. The argument can be a scalar, vector or matrix, but it is intended to have a scalar argument. `ELTIME` returns the elapsed time in seconds. Initialize the time by calling `ELTIME(0)`, which return 0, and then subsequent calls of `ELTIME(s)`, with $\texttt{s} > 0$, will give the elapsed time since initialization. If not initialized, `ELTIME(s)`, with $\texttt{s} > 0$, returns the elapsed time since midnight. `ELTIME(s)`, with $\texttt{s} < 0$, always returns the elapsed time since midnight. For example:

call	*result*
`ELTIME(0)`	returns 0 (initialization)
`ELTIME(1)`	returns elapsed time since initialization, or if not initialized, returns elapsed time since midnight
`ELTIME(-1)`	returns elapsed time since midnight

DIM

Syntax `v = DIM( v1, v2 )`

The `DIM` function is an element by element function that requires two (2) arguments. The arguments can be scalars, vectors or matrices, but vectors and matrices cannot be mixed, and all arrays must be the same size. Scalar arguments result in a scalar. A vector argument result in a vector with the same length as the argument, and matrix arguments result in a matrix with the same dimensions as the arguments.

argument 1	*argument 2*	*result*			
scalar	scalar	scalar	`DIM( a, b )`	=	`max( 0, a-b )`
vector	vector	vector	`DIM( x, y )[j]`	=	`max( 0, x[j]-y[j] )`
scalar	vector	vector	`DIM( a, x )[j]`	=	`max( 0, a-x[j] )`
vector	scalar	vector	`DIM( x, a )[j]`	=	`max( 0, x[j]-a )`
matrix	matrix	matrix	`DIM( m1, m2 )[i,j]`	=	`max( 0, m1[i,j]-m2[i,j] )`
scalar	matrix	matrix	`DIM( a, m )[i,j]`	=	`max( 0, a-m[i,j] )`
matrix	scalar	matrix	`DIM( m, a )[i,j]`	=	`max( 0, m[i,j]-a )`

MOD

Syntax `v = MOD( v1, v2 )`

The `MOD` function is an element by element function that requires two (2) arguments. The arguments can be scalars, vectors or matrices, but vectors and matrices cannot be mixed, and all arrays must be the same size. Scalar arguments result in a scalar. A vector argument result in a vector with the same length as the argument, and matrix arguments result in a matrix with the same dimensions as the arguments.

argument 1	*argument 2*	*result*			
scalar	scalar	scalar	`MOD( a, b )`	=	`a-b*INT(a/b)`
vector	vector	vector	`MOD( x, y )[j]`	=	`x[j]-y[j]*INT(x[j]/y[j])`
scalar	vector	vector	`MOD( a, x )[j]`	=	`a-x[j]*INT(a/x[j])`
vector	scalar	vector	`MOD( x, a )[j]`	=	`x[j]-a*INT(x[j]/a)`
matrix	matrix	matrix	`MOD( m1, m2 )[i,j]`	=	`m1[i,j]-m2[i,j]*INT(m1[i,j]/m2[i,j])`
scalar	matrix	matrix	`MOD( a, m )[i,j]`	=	`a-m[i,j]*INT(a/m[i,j])`
matrix	scalar	matrix	`MOD( m, a )[i,j]`	=	`m[i,j]-a*INT(m[i,j]/a)`

SIGN

Syntax `v = SIGN( v1, v2 )`

The `SIGN` function is an element by element function that requires two (2) arguments. The arguments can be scalars, vectors or matrices, but vectors and matrices cannot be mixed, and all arrays must be the same size. Scalar arguments result in a scalar. A vector argument result in a vector with the same length as the argument, and matrix arguments result in a matrix with the same dimensions as the arguments.

argument 1	*argument 2*	*result*			
scalar	scalar	scalar	`SIGN( a, b )`	=	`|a|(sign of b)`
vector	vector	vector	`SIGN( x, y )[j]`	=	`|x[j]|(sign of y[j])`
scalar	vector	vector	`SIGN( a, x )[j]`	=	`|a|(sign of x[j])`
vector	scalar	vector	`SIGN( x, a )[j]`	=	`|x[j]|(sign of a)`
matrix	matrix	matrix	`SIGN( m1, m2 )[i,j]`	=	`|m1[i,j]|(sign of m2[i,j])`
scalar	matrix	matrix	`SIGN( a, m )[i,j]`	=	`|a|(sign of m[i,j])`
matrix	scalar	matrix	`SIGN( m, a )[i,j]`	=	`|m[i,j]|(sign of a)`

MIN

Syntax `v = MIN( v1,{v2,...} )`

The `MIN` function is an element by element function that accepts from one (1) to a maximum of twenty (20) arguments. If only one argument is supplied, the minimum element of that argument is returned. If two or more arguments are supplied, the arguments are compared element by element, and the minimum values are returned. The arguments can be scalars, vectors or matrices, but cannot be mixed, and all arrays must be the same size. Scalar arguments result in a scalar. Vector arguments result in a vector with the same length as the arguments, and matrix arguments result in a matrix with the same dimensions as the arguments.

arguments	*result*			
one vector	scalar	`MIN(x)`	=	`minimum(x[1],x[2],...,x[#])`
one matrix	scalar	`MIN(m)`	=	`minimum(m[1,1],...,m[#,#])`
scalars	scalar	`MIN(a,b,...)`	=	`minimum(a,b,...)`
vector	vector	`MIN(x,y,...)[j]`	=	`minimum(x[j],y[j],...)`
matrix	matrix	`MIN(m1,m2,...)[i,j]`	=	`minimum(m1[i,j],m2[i,j],...)`

MAX

Syntax `v = MAX( v1,{v2,...} )`

The `MAX` function is an element by element function that accepts from one (1) to a maximum of twenty (20) arguments. If only one argument is supplied, the maximum element of that argument is returned. If two or more arguments are supplied, the arguments are compared element by element, and the maximum values are returned. The arguments can be scalars, vectors or matrices, but cannot be mixed, and all arrays must be the same size. Scalar arguments result in a scalar. Vector arguments result in a vector with the same length as the arguments, and matrix arguments result in a matrix with the same dimensions as the arguments.

arguments	*result*			
one vector	scalar	`MAX(x)`	=	`maximum(x[1],x[2],...,x[#])`
one matrix	scalar	`MAX(m)`	=	`maximum(m[1,1],...,m[#,#])`
scalars	scalar	`MAX(a,b,...)`	=	`maximum(a,b,...)`
vector	vector	`MAX(x,y,...)[j]`	=	`maximum(x[j],y[j],...)`
matrix	matrix	`MAX(m1,m2,...)[i,j]`	=	`maximum(m1[i,j],m2[i,j],...)`

Special mathematical functions

Airy's functions

The Airy functions Ai and Bi occur in electromagnetic theory and in quantum mechanics.

Airy's functions of the first kind, `AIRY`, and second kind, `BIRY` are

$$\texttt{AIRY}(x) = Ai(x) = c_1 f(x) - c_2 g(x)$$

$$\texttt{BIRY}(x) = Bi(x) = \sqrt{3}\,[c_1 f(x) + c_2 g(x)]$$

$$\text{where}\quad c_1 = \frac{3^{-2/3}}{\Gamma(\frac{2}{3})}\qquad c_2 = \frac{3^{-1/3}}{\Gamma(\frac{1}{3})}$$

$$f(x) = 1 + \frac{1}{3!}x^3 + \frac{1\cdot 4}{6!}x^6 + \frac{1\cdot 4\cdot 7}{9!}x^9 + \cdots$$

$$g(x) = x + \frac{2}{4!}x^4 + \frac{2\cdot 5}{7!}x^7 + \frac{2\cdot 5\cdot 8}{10!}x^{10} + \ldots$$

Beta functions

Complete beta function

$$\texttt{BETA}(a,b) = \beta(a,b) = \int_0^1 t^{a-1}(1-t)^{b-1}dt = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

Where a and b are both real and positive.

Incomplete beta function

$$\texttt{BETAIN}(x,a,b) = I_x(a,b) = \frac{\int_0^x t^{a-1}(1-t)^{b-1}dt}{\int_0^1 t^{a-1}(1-t)^{b-1}dt} = \frac{\int_0^x t^{a-1}(1-t)^{b-1}dt}{\beta(a,b)}$$

Bessel functions

First and second kinds

The Bessel functions of the first and second kinds, J_n and Y_n, are linearly independent solutions to the differential equation

$$x^2\frac{d^2}{dx^2}y + x\frac{d}{dx}y + (x^2-n^2)y = 0$$

Bessel functions arise in solving differential equations for systems with cylindrical symmetry.

$$\texttt{BESJ0}(x) = J_0(x) = \frac{1}{\pi}\int_0^\pi \cos(x\sin t)dt = \sum_{k=0}^\infty \frac{(-\frac{x^2}{4})^k}{k!k!}$$

$$\texttt{BESJ1}(x) = J_1(x) = \frac{1}{\pi}\int_0^\pi \cos(x\sin t - t)dt = \frac{x}{2}\sum_{k=0}^\infty \frac{(-\frac{x^2}{4})^k}{k!(k+1)!}$$

$$\begin{aligned}\texttt{BESY0}(x) &= Y_0(x) = \frac{4}{\pi^2}\int_0^{\frac{\pi}{2}} \cos(x\cos t)[\gamma + \ln(2x\sin^2(t)]dt \\ &= \frac{2}{\pi}[-\frac{1}{x} + \ln(\frac{x}{2}+\gamma)J_0(x) - \sum_{k=1}^\infty(\sum_{j=1}^k\frac{1}{j})\frac{(-\frac{x^2}{4})^k}{k!k!}] \\ &= \frac{2}{\pi}[J_0(x)\ln(\frac{\gamma x}{2}) + \sum_{k=1}^\infty\frac{(\frac{x^2}{4})^k}{k!k!}(\sum_{j=1}^k\frac{1}{j})]\end{aligned}$$

$$\texttt{BESY1}(x) = Y_1(x) = \frac{2}{\pi}[-\frac{1}{x} + \ln(\frac{x}{2})J_1(x) - \frac{x}{4}\sum_{k=0}^\infty[\psi(k+1)+\psi(k+2)]\frac{(-\frac{x^2}{4})^k}{k!(k+1)!}$$

where ψ is the digamma function, and where γ is Euler's constant

$$\gamma = \lim_{m\to\infty}[\sum_{k=1}^m\frac{1}{k} - \ln(m)] = 0.5772156649\ldots$$

Modified Bessel functions

The modified Bessel functions of the first and second kinds, I_n and K_n, are solutions to the differential equation

$$x^2\frac{d^2}{dx^2}y + x\frac{d}{dx}y - (x^2+n^2)y = 0$$

$$\texttt{BESI0}(x) = I_0(x) = \sum_{k=0}^\infty\frac{(\frac{x}{2})^2k}{k!k!}$$

$$\texttt{BESI1}(x) = I_1(x) = \frac{x}{2}\sum_{k=0}^\infty\frac{(\frac{x}{2})^2k}{k!(k+1)!}$$

$$\texttt{BESK0}(x) = K_0(x) = -[\ln(\frac{x}{2}+\gamma]I_0(x) - \sum_{k=1}^\infty\sum_{j=1}^k\frac{1}{j}\frac{(\frac{x}{2})^2k}{k!k!}$$

$$\texttt{BESK1}(x) = K_1(x) = \frac{1}{x} + \ln(\frac{x}{2})I_1(x) - \frac{x}{4}\sum_{k=0}^\infty[\psi(k+1)+\psi(k+2)]\frac{(\frac{x}{2})^2k}{k!(k+1)!}$$

where ψ is the digamma function, where γ is Euler's constant

$$\gamma = \lim_{m\to\infty}[\sum_{k=1}^m\frac{1}{k} - \ln(m)] = 0.5772156649\ldots$$

Binomial coefficient

$$\texttt{BINOM}(n,m) = \begin{pmatrix} n \\ m \end{pmatrix} = \frac{n!}{(n-m)!m!}$$

Chebyshev polynomials

Series of Chebyshev polynomials are used in making numerical approximations to functions.

$$\texttt{CHEBY}(n,x) = T_n(x) = \cos(n\cos^{-1}x)$$

Probability functions

Bivariate normal probability function

$$\texttt{BIVARNOR}(h,k,r) = \text{Probability}(x > h, y > k)$$

$$= \int_h^{+\infty} \int_k^{+\infty} \frac{\exp(-[\frac{x^2-2rxy+y^2}{2(1-r^2)}])}{2\pi\sqrt{1-r^2}} dxdy$$

Chi-square probability function

$$\texttt{CHISQ}(x,n) = \chi^2(x|n) = [2^{\frac{n}{2}}\Gamma(\frac{n}{2})]^{-1} \int_x^{+\infty} t^{\frac{n}{2}-1} e^{-\frac{t}{2}} dt \quad \text{where} \quad 0 \le x < \infty, \;\; n \ge 1$$

Inverse Chi-square

Given $y = \chi^2(x|n)$ and n, then x is found: $\texttt{CHISQINV}(y,n) = x$

Probability integral of Chi-square distribution

$$\texttt{PROB}(\chi^2, n) = \frac{1}{2^{n/2}\Gamma(n/2)} \int_{\chi^2}^{\infty} t^{\frac{n}{2}-1} e^{-t/2} dt$$

where n is the number of degrees of freedom, $\texttt{n} \ge 1$. For more information, please refer to:

Handbook of Mathematical Functions
by Abramowitz and Stegun, 1964, pages 978ff.

Gaussian or normal probability function

The Gaussian probability function is also known as the normal probability function.

$$\texttt{GAUSS}(x) = \texttt{FREQ(x)} = P(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt$$

Normalized Gaussian distribution

$$\texttt{NORMAL(x,a,b)} = \frac{1}{b\sqrt{2\pi}} e^{-\frac{(x-a)^2}{2b^2}}$$

Inverse Gaussian

Given $y = \texttt{GAUSS}(x)$, then x is found: $\texttt{GAUSSIN}(y) = x$

Cosine integral

$$\texttt{COSINT}(x) = Ci(x) = -\int_x^{\infty} \frac{cos(t)}{t} dt = \gamma + \ln|x| + \int_0^x \frac{\cos(t)-1}{t} dt$$

where γ is Euler's constant

$$\gamma = \lim_{m\to\infty} [\sum_{k=1}^{m} \frac{1}{k} - \ln(m)] = 0.5772156649\ldots$$

Sine integral

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

Dawson's integral

$$\texttt{DAWSON}(x) = e^{-x^2} \int_0^x e^{t^2} dt$$

Digamma Psi function

$$\texttt{DIGAMMA}(x) = \psi(x) = \frac{d}{dx}[\ln\Gamma(x)] = \frac{\Gamma'(x)}{\Gamma(x)}$$

$$\text{Note that} \quad \psi(1) = -\gamma, \;\; \psi(n) = -\gamma + \sum_{k=1}^{n-1} k^{-1}$$

where γ is Euler's constant

$$\gamma = \lim_{m\to\infty}[\sum_{k=1}^{m}\frac{1}{k} - \ln(m)] = 0.5772156649\ldots$$

Dilogarithm

The dilogarithm, Li_2, occurs in Feynman diagram integrals in particle physics.

$$\texttt{DILOG}(x) = Li_2(x) = -\int_0^x \frac{\ln|1-t|}{t}dt = \sum_{k=1}^{\infty}\frac{x^k}{k^2}$$

$Li_2(1-x)$ is sometimes known as Spence's integral.

Elliptic integrals

Elliptic integrals have the form $\int \mathcal{F}(x,y)dx$ where $\mathcal{F}$ is a rational function of x and y, and y^2 is a cubic or quartic polynomial in x. Any elliptic integral can be expressed in terms of the three canonical forms. The elliptic integrals are said to be "complete" when the amplitude is $\frac{\pi}{2}$.

First kind

For $|x| \leq 1$ and $|p| \leq \pi/2$

$$\texttt{FINELLIC}(x,p) = F(p,x) = \int_0^p \frac{dt}{\sqrt{1-x^2\sin^2 t}} = \int_0^y \frac{dy}{\sqrt{(1-y^2)(1-k^2y^2)}}$$

For $|x| < 1$, `ELLICK(x)` $= F(\pi/2, x)$

Second kind

For $|x| \leq 1$ and $|p| \leq \pi/2$

$$\texttt{EINELLIC}(x,p) = E(p,x) = \int_0^p \sqrt{1-x^2\sin^2 t}dt = \int_0^y \frac{\sqrt{1-x^2y^2}}{\sqrt{1-y^2}}dy$$

where $y = \sin(p)$. For $|x| \leq 1$ $\texttt{ELLICE}(x) = E(\pi/2, x)$

Error function

For more information, please refer to:

Algorithm 610, A Portable FORTRAN Subroutine for Derivatives of the Psi Function, D.E. Amos, ACM Transactions on Mathematical Software, December 1983, Vol. 9, No. 4, pages 494–502.

Handbook of Mathematical Functions, M. Abramowitz, I.A. Stegun, New York, Dover Publications Inc., 1965.

The error function, `ERF(x)`, is the integral of the Gaussian distribution.

$$\texttt{ERF}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$$

Inverse

Given $y = \texttt{ERF}(x)$, then x is found: $\texttt{AERF}(y) = x$

Complementary error function

$$\begin{aligned}\texttt{ERFC}(x) &= \frac{2}{\sqrt{\pi}}\int_x^{\infty} e^{-t^2}dt \\ &= 1 - \texttt{ERF}(x)\end{aligned}$$

Inverse

Given $y = \texttt{ERFC}(x)$, then x is found: $\texttt{AERFC}(y) = x$

Exponential integrals

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

$$\texttt{EI}(x) = \int_{-\infty}^{x}\frac{e^{-t}}{t}dt = -\texttt{EXPINT}(-x)$$

Exponential integrals of order n

$$\texttt{EXPN}(x,n) = \int_1^{\infty}\frac{e^{-xt}}{t^n}dt \quad \text{where} \quad n = 0,1,2,\ldots \quad \text{and} \quad x > 0$$

$\texttt{EXPN}(x, n)$ is related to $\texttt{EXPINT}(x)$ via $\texttt{EXPN}(1, 1) = \texttt{EXPINT}(1)$.

Fermi-Dirac function

$$\texttt{FERDIRAC}(x, p) = \int_0^\infty \frac{t^p}{e^{t-x}+1} dt \quad \text{only for} \quad p = -\frac{1}{2}, \frac{1}{2}, \frac{3}{2}$$

Fisher's F-distribution function

Fisher's F-distribution function is also known as the variance-ratio distribution function.

$$\texttt{FISHER}(m, n, x) = \frac{\Gamma(\frac{m+n}{2})}{\Gamma(\frac{m}{2})\Gamma(\frac{n}{2})} \int_0^{\frac{m}{n}x} \frac{t^{\frac{m}{2}-1}}{(t+1)^{\frac{m+n}{2}}} dt = \frac{m^{\frac{m}{2}} n^{\frac{n}{2}}}{\beta(\frac{m}{2}, \frac{n}{2})} \int_0^x t^{\frac{m-2}{2}} (n+mt)^{-\frac{m+n}{2}} dt$$

where $x \geq 0$ and β is the complete β function.

Fresnel integrals

Fresnel and associated Fresnel integrals are related to the error function and occur in diffraction theory.

$$\texttt{FREC1}(x) = C(x) = \int_0^x \cos(\frac{\pi}{2} t^2) dt$$

$$\texttt{FRES1}(x) = S(x) = \int_0^x \sin(\frac{\pi}{2} t^2) dt$$

$$\texttt{FREC2}(x) = \frac{1}{\sqrt{2\pi}} \int_0^x \frac{\cos(t)}{\sqrt{t}} dt = \texttt{FREC1}(\sqrt{\frac{2x}{\pi}})$$

$$\texttt{FRES2}(x) = \frac{1}{\sqrt{2\pi}} \int_0^x \frac{\sin(t)}{\sqrt{t}} dt = \texttt{FRES1}(\sqrt{\frac{2x}{\pi}})$$

Gamma function

$$\texttt{GAMMA}(x) = \Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

Natural logarithm of the Gamma function

$$\texttt{GAMMLN}(x) = \texttt{LOGAM}(x) = \ln \Gamma(x)$$

Incomplete Gamma functions

$$\texttt{GAMMACIN}(x, a) = \Gamma(x, a) = \int_a^\infty t^{x-1} e^{-t} dt$$

$$\texttt{GAMMAIN}(x, a) = \gamma(x, a) = \int_0^a t^{x-1} e^{-t} dt$$

$$\texttt{GAMMATIN}(x, a) = \gamma^*(x, a) = \frac{a^{-x}\gamma(x, a)}{\Gamma(x)}$$

$$\texttt{GAMMQ}(x, a) = \frac{1}{\Gamma(x)} \int_a^\infty t^{x-1} e^{-t} dt$$

Hermite polynomials

Hermite polynomials arise as the quantum mechanical wave functions for a harmonic oscillator.

$$\texttt{HERMITE}(n, x) = H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} (e^{-x^2})$$

Hypergeometric function

The hypergeometric function is also called the Gauss series.

$$\texttt{HYPGEO}(a, b, c, x) = {}_2F_1(a, b, c, x) = \frac{\Gamma(c)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^\infty \frac{\Gamma(a+n)\Gamma(b+n)}{\Gamma(c+n)} \frac{x^n}{n!}$$

Logarithmic confluent hypergeometric function

$$\texttt{CHLOGU}(a, b, x) = U(a, b, x) = x^{-a} {}_2F_0(a, 1+a-b; ; -1/x)$$

$$= \frac{\pi}{\sin(\pi b)} \left[\frac{M(a, b, x)}{\Gamma(1+a-b)\Gamma(b)} - x^{1-b} \frac{M(1+a-b, 2-b, x)}{\Gamma(a)\Gamma(2-b)} \right]$$

Where $M(a,b,x)$ is Kummer's Function or the regular confluent hypergeometric function

$$M(a,b,x) = {}_1F_1 = 1 + \frac{a}{b}x + \frac{(a)_2}{(b)_2}\frac{x^2}{2!} + \cdots + \frac{(a)_n}{(b)_n}\frac{x^n}{n!} + \cdots$$

$$\text{where} \quad (p)_n = p(p+1)(p+2)\cdots(p+n-1) = \Gamma(p+n) \quad \text{and} \quad (p)_0 = 1$$

Note: This function fails when $1+a-b$ is close to zero for small x.

Jacobi polynomials

Jacobi polynomials occur in studies of the rotation group, particularly in quantum mechanics. Legendre and Chebyshev are special cases of Jacobi polynomials.

$$\texttt{JACOBI}(a,b,n,x) = P_n^{(a,b)}(x) = \frac{(-1)^n}{2^n n!}(1-x)^{-a}(1+x)^{-b}\frac{d^n}{dx^n}[(1-x)^{a+n}(1+x)^{b+n}]$$

Kelvin functions

Kelvin functions occur in electrical engineering.

Kelvin functions of the first kind

The Kelvin functions of the first kind, order 0

$$\texttt{BER}(x) = ber_0 x \qquad \texttt{BEI(x)} = bei_0 x$$

$$\text{where} \quad ber_0 x + i\ bei_0 x = J_0(xe^{3\pi i/4})$$

and where J_0 is Bessel's function of the first kind, order 0.

Kelvin functions of the second kind

Kelvin functions of the second kind, order 0, for non-negative x.

$$\texttt{KER}(x) = ker_0 x \quad KEI(x) = kei_0 x$$

$$\text{where} \quad ker_0 x + i\ kei_0 x = e^{-\frac{\nu\pi i}{2}}|_{\nu=0}\ K_\nu(xe^{i\pi/4})|_{\nu=0} = K_0(xe^{i\pi/4})$$

and K_0 is the modified Bessel function of the second kind, of order 0.

Laguerre polynomials

Laguerre polynomials are related to hydrogen atom wave functions in quantum mechanics.

$$\texttt{LAGUERRE}(n,x) = e^x\frac{d^n}{dx^n}[x^n e^{-x}]$$

Legendre functions and polynomials

The most elementary of the Legendre functions, the Legendre polynomials, $P_n(x)$ can be defined by the generating function:

$$(1-2xr+r^2)^{-\frac{1}{2}} = \sum_{n=0}^{\infty} P_n(x)r^n$$

More explicit representations are:

$$\texttt{LEGENDRE}(n,x) = P_n(x) \equiv P_n^0(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2-1)^n = {}_2F_1(-n,n+1,1,(1-x)/2)$$

where ${}_2F_1$ is the hypergeometric function.

Unnormalized associated Legendre functions of degree n

$$\begin{aligned} \texttt{PLMU}(n,m,x) &= P_n^m(x) \\ &= (1-x^2)^{m/2}\frac{d^m}{dx^m}P_n(x) \quad \text{for} \quad m \geq 0 \\ &= \frac{(n+m)!}{(n-m)!}P_n^{-m}(x) \quad \text{for} \quad m < 0 \end{aligned}$$

Normalized associated Legendre functions

$$\texttt{PLM}(n,m,x) = \texttt{PLMN}(n,m,x) = \overline{P_n^m}(x) = \sqrt{\frac{2n+1}{2}\frac{(n-m)!}{(n+m)!}}P_n^m(x)$$

where $-1 < x < 1$, n is a non-negative integer and $m = -n, \ldots, -1, 0, 1, \ldots, n$ satisfy the differential equation:

$$(1-x^2)\frac{d^2y}{dx^2} - 2x\frac{dy}{dx} + [n(n+1) - \frac{m^2}{1-x^2}]y = 0$$

Note: $\texttt{PLMU}(n,0,x) = \texttt{LEGENDRE}(n,x)$.

Poisson-Charlier polynomial

$$\texttt{POICA}(a,n,x) = \rho_\eta^\alpha(x) = a^{n/2}(n!)^{-1/2}\sum_{k=0}^{n}(-1)^{n-k}\begin{pmatrix} n \\ k \end{pmatrix} k!a^{-k}\begin{pmatrix} x \\ k \end{pmatrix}$$

Rademacher function

$$\texttt{RADMAC}(k,x) = \Upsilon_k(x) = \text{sign}[\sin(2^{k+1}\pi x)]$$

Alternatively, find m such that $m \le 2^{k+1}x < (m+1)$; where $m = 0,\ \pm1,\ \pm2,\ \ldots$

$$\text{then} \quad \Upsilon_k(x) = \begin{cases} +1 & \text{if } m\text{=even} \\ -1 & \text{if } m\text{=odd} \end{cases}$$

Struve functions

First order

$$\texttt{STRUVE0}(x) = \frac{2}{\pi}[x - \frac{x^3}{(1\cdot3)^2} + \frac{x^5}{(1\cdot3\cdot5)^2} - \cdots]$$

Second order

$$\texttt{STRUVE1}(x) = \frac{2}{\pi}[\frac{x^2}{1^2\cdot3} - \frac{x^4}{(1\cdot3)^2\cdot5} + \frac{x^6}{(1\cdot3\cdot5)^2\cdot7} - \cdots]$$

Student's t-distribution

$$\texttt{STUDENT}(t,n) = P(\frac{t}{n}) = [\beta(\frac{1}{2},\frac{n}{2})\sqrt{n}]^{-1}\int_{-\infty}^{t}(1+\frac{x^2}{n})^{-(n+1)/2}dx$$

where β is the complete β function. Since $\Gamma(\frac{1}{2}) = \sqrt{\pi}$

$$\texttt{STUDENT}(t,n) = \frac{\Gamma(\frac{n+1}{2})}{\Gamma(\frac{n}{2})\sqrt{n\pi}}\int_{-\infty}^{t}(1+\frac{x^2}{n})^{-(n+1)/2}dx$$

Inverse

Given $y = P(\frac{t}{n})$ and n, then t is found: $\texttt{STUDENTI}(y,n) = t$.

Normalized tina resolution

$$\begin{aligned} \texttt{TINA}(x,a,b,c) &= \frac{1}{2ce^{b^2/4c^2}}e^{\frac{x-a}{c}}(1-\texttt{ERF}(\frac{x-a}{b}) \\ &= \frac{1}{2ce^{b^2/4c^2}}e^{(1-\frac{2}{\sqrt{\pi}})(x-a)/c}\int_0^{(x-a)/b} e^{-t^2}dt \end{aligned}$$

If $(x-a)/b > 80$ or $(x-a)/c > 80$ then $\texttt{TINA}(x,a,b,c)$ is returned as zero (0). This is done to circumvent floating overflow problems for large x.

Vector coupling coefficients

Clebsch-Gordan coefficients, Wigner's $3j$, $6j$, and $9j$ symbols, Jahn's U-function, and Racah coefficients are the vector coupling coefficients in the theory of angular momentum in quantum mechanics. For more information, please refer to:

Group Theory and its Application to the Quantum Mechanics of Atomic Spectra
by Eugene P. Wigner, Academic Press, 1959

Elementary Theory of Angular Momentum
by M.E. Rose, John Wiley & Sons, Inc., 1957

Angular Momentum in Quantum Mechanics
by A.R. Edmonds, Princeton University Press, 1960

The Clebsch-Gordan vector-addition coefficient, $(j_1j_2m_1m_2|jm)$ is defined as:

$$(j_1j_2m_1m_2|jm) = \delta(m, m_1+m_2)\sqrt{\frac{(j_1+j_2-j)!(j+j_1-j_2)!(j+j_2-j_1)!(2j+1)}{(j+j_1+j_2+1)!}}$$

$$\times\sum_k \frac{(-1)^k\sqrt{(j_1+m_1)!(j_1-m_1)!(j_2+m_2)!(j_2-m_2)!(j+m)!(j-m)!}}{k!(j_1+j_2-j-k)!(j_1-m_1-k)!(j_2+m_2-k)!(j-j_2+m_1+k)!(j-j_1-m_2+k)!}$$

$$\text{where} \quad \delta(i,k) = \begin{cases} 1, & i = k \\ 0, & i \ne k \end{cases}$$

and with the following restrictions:

$$j_1, j_2, j = +n \quad \text{or} \quad +\frac{n}{2} \quad (n = \text{integer})$$

$$j_1 + j_2 + j = n$$

$$\left.\begin{array}{l} +j_1 + j_2 - j \\ +j_1 - j_2 + j \\ -j_1 + j_2 + j \end{array}\right\} \geq 0$$

$$m_1, m_2, m = \pm n \quad \text{or} \quad \pm\frac{n}{2}$$

$$|m_1| \leq j_1, \quad |m_2| \leq j_2, \quad |m| \leq j$$

$$(j_1 j_2 m_1 m2 | j_1 j_2 j m) = 0 \quad \text{for} \quad m_1 + m_2 \neq m$$

Clebsch-Gordan coefficient function

$$\texttt{CLEBSG}(j1, j2, j, m1, m2, m) = (j_1 j_2 m_1 m_2 | j m)$$

and

$$\texttt{CLEBSG}(j1, j2, j) = (j_1 j_2 00 | j0)$$

Note: $(j_1 j_2 00 | j0) = 0$ when $j_1 + j_2 + j = 2n + 1$.

Wigner $3-j$ function

$$\texttt{WIGN3J}(j1, j2, j, m1, m2, m) = \begin{pmatrix} j_1 & j_2 & j \\ m_1 & m_2 & m \end{pmatrix} = \frac{(-1)^{j_1 - j_2 - m}}{\sqrt{2j+1}} (j_1 j_2 m_1 m_2 | j\, -m)$$

and

$$\texttt{WIGN3J}(j1, j2, j) = \begin{pmatrix} j_1 & j_2 & j \\ 0 & 0 & 0 \end{pmatrix} = \frac{(-1)^{j_1 - j_2}}{\sqrt{2j+1}} (j_1 j_2 00 | j0)$$

Racah coefficients

To define the Racah coefficients we first define the "triangle" coefficient:

$$\Delta(abc) = \sqrt{\frac{(a+b-c)!(a-b+c)!(-a+b+c)!}{(a+b+c+1)!}}$$

Racah's W-function is defined as:

$$\begin{aligned} &\texttt{RACAHC}(a, b, c, d, e, f) = W(abcd; ef) = \\ &\quad \Delta(abe)\Delta(cde)\Delta(acf)\Delta(bdf) \\ &\quad \times \sum_k \frac{(-1)^{k+a+b+c+d}(k+1)!}{(k-a-b-e)!(k-c-d-e)!(k-a-c-f)!(k-b-d-f)!} \\ &\quad \times \frac{1}{(a+b+c+d-k)!(a+d+e+f-k)!(b+c+e+f-k)!} \end{aligned}$$

Wigner's $6-j$ function

$$\texttt{WIGN6J}(j1, j2, j, m1, m2, m) = \begin{Bmatrix} j_1 & j_2 & j \\ m_1 & m_2 & m \end{Bmatrix} = (-1)^{j_1 + j_2 + m_1 + m_2} W(j_1 j_2 m_2 m_1; jm)$$

Wigner's $9-j$ function

$$\texttt{WIGN9J}(j1, j2, j, m1, m2, m, n1, n2, n) = \begin{Bmatrix} j_1 & j_2 & j \\ m_1 & m_2 & m \\ n_1 & n_2 & n \end{Bmatrix}$$

please refer to: Group Theory and its Application to the Quantum Mechanics of Atomic Spectra by Eugene P. Wigner, Academic Press, 1959

Jahn's U function

$$\texttt{JAHNUF}(j1, j2, m2, m1, j, m) = U(j_1 j_2 m_2 m_1; jm)$$

$$= (-1)^{j_1 + j_2 + m_1 + m_2} \sqrt{(2j+1)(2m+1)} \begin{Bmatrix} j_1 & j_2 & j \\ m_1 & m_2 & m \end{Bmatrix}$$

Walsh functions

The Walsh functions form a complete set of orthogonal, normalized, rectangular periodic functions with a period of one.

Let m be found so that k can be written as a binary number:

$$k = \sum_{i=0}^{m} a_i 2^i; \quad \text{where} \quad a_i = 0 \quad \text{or} \quad 1 \quad \text{then} \quad \mathtt{WALSH}(k,x) = \prod_{i=0}^{m} \Upsilon_i^{a_i}(x)$$

where Υ_i is the Rademacher function. WALSH can assume the values of 1 or -1 only.

Voigt profile

The Voigt profile function is the convolution integral of a Gaussian and a Lorentzian function. We wish to evaluate

$$G(E) = \int_{-\infty}^{\infty} f(E-E')g(E'-E_1)dE'$$

where $f(E)$ is the Gaussian function

$$f(E) = e^{-E^2/2\sigma^2}$$

and $g(E)$ is the Lorentzian

$$g(E) = \frac{1}{E^2+\Gamma^2}$$

Putting

$$\begin{aligned} x &= (E'-E)/\sqrt{2}\sigma \\ a &= (E_1-E)/\sqrt{2}\sigma \\ b &= |\Gamma/\sqrt{2}\sigma \end{aligned}$$

we obtain

$$G = \frac{1}{\sqrt{2}\sigma}\int_{-\infty}^{\infty}\frac{e^{-x^2}}{b^2+(a-x)^2}dx$$

This integral has been evaluated for $b>0$ in terms of the complex error function $\omega(z)$.

$$\begin{aligned} G &= \frac{1}{\sqrt{2}\sigma}\frac{\pi}{b}\Re[\omega(a+ib)] \\ &= \frac{1}{\sqrt{2}\sigma} - \Re[e^{b+ia}erfc(b-ia)] \end{aligned}$$

The Voigt function implemented here is: $\mathtt{VOIGT}(E, E_1, \Gamma/2, \sigma) = G$

Functions that return a string

DATE

Syntax `string = DATE`

The DATE function has no arguments. It returns a string which contains the current date with the format dd-mmm-yyyy.

Example

function	*result*
DATE	‘ 6-MAY-1993’

TIME

Syntax `string = TIME`

The TIME function has no arguments. It returns a string which contains the current time with the format hh:mm:ss.

Example

function	*result*
TIME	‘10:36:25’

UCASE

Syntax `string = UCASE(string)`

The UCASE function converts a string into upper case.

Example

function	*result*
UCASE(‘this is a string’)	‘THIS IS A STRING’

LCASE

Syntax `string = LCASE(string)`

The LCASE function converts a string into lower case.

Example

function	*result*
LCASE('THIS IS A STRING')	'this is a string'

TCASE

Syntax `string = TCASE(string)`

The `TCASE` function toggles the case for each character of a string.

Example

function	*result*
TCASE('ThIs iS A StRiNg')	'tHiS Is a sTrInG'

CHAR

Syntax `string = CHAR(scalar)`
`string = CHAR(vector)`

The `CHAR` function accepts either a numeric vector or a numeric scalar as argument. It converts these ASCII decimal codes to the equivalent characters and returns these characters as a string. The inverse of this function is the `ICHAR` function.

EXPAND

Syntax `string = EXPAND(string)`

The `EXPAND` function accepts a string as argument. The result is also a character string. It parses the argument, expanding any expression variables it finds. If an expression variable, contained in the argument, also contains expression variables then they are also expanded, and so on until all such expression variables have been expanded. Syntax checking is done during the expansion.

The maximum length of a completely expanded expression is two thousand five hundred (2500) characters.

Example


```
A=2              ! define a scalar A
B=3              ! define a scalar B
FC1='(A+B)/A'    ! define a string variable FC1
FC2='SQRT(A/B)'  ! define a string variable FC2
FC3='FC1*FC2'    ! define a string variable FC3
FC4='FC3+4*FC2'  ! define a string variable FC4
=FC4             ! displays 'FC3+4*FC2'
=EXPAND(FC4)     ! displays '(((A+B)/A)*(SQRT(A/B)))+4*(SQRT(A/B))'
=EVAL(FC4)       ! displays 5.307228
```

VARNAME

Syntax `string = VARNAME(variable)`

The `VARNAME` function accepts a variable, either string or numeric, as its argument, and converts that variable name into a string. This function can be useful in scripts where a variable is passed to the script as one of the generalized `?` parameter. You could then convert the name to a string for display or manipulation.

Examples

The following script shows one way in which the `VARNAME` function could be used.

```
t1=varname(?1)   ! 1st variable name passed to the script converted to a string
t2=varname(?2)   ! 2nd variable name passed to the script converted to a string
graph ?1 ?2      ! plot graph of 2nd variable versus 1st variable
text 'graph of '//t1//' vs '//t2  ! lable the plot with the command
```

VARTYPE

Syntax `string = VARTYPE(name)`

The `VARTYPE` function returns the type of the argument as a character string. If the argument is undefined, `VARTYPE` returns the string `unknown`. For example, if `S` is a scalar, `X` is a vector, `M` is a matrix, `T` is a string variable, and `TA` is a string array variable, then:

	returned string
VARTYPE(1.0)	‘number’
VARTYPE(S)	‘scalar’
VARTYPE(X)	‘vector’
VARTYPE(M)	‘matrix’
VARTYPE(T)	‘string’
VARTYPE(TA)	‘string array’
VARTYPE(‘abc’)	‘literal string’

STRING

Syntax `quote_string = STRING(some_string)`

The `STRING` function is useful in command scripts, where you want to pass a string to the script as a parameter, without enclosing it in quotes. For example, `@script test` where `script.pcm` is as follows:

```
x = string(?1)//‘ is a quote string’
display x
```

would display

```
test is a quote string
```

The key element here is that the user does not have to supply the quotes around the parameter `test`.

Examples

STRING(test case)	produces	‘test case’
STRING(test)	produces	‘test’
STRING(‘test’)	produces	‘test’

RCHAR

Syntax `string = RCHAR(scalar)`
`string = RCHAR(scalar,string)`

The `RCHAR` function accepts a numeric scalar as first argument and returns a string. It converts the numeric value to a string using PHYSICA’s own format. You can specify your own format string by including it as the optional second argument, a string.

Examples

function	*result*
RCHAR(-1.234)	‘-1.234’
RCHAR(PI)	‘3.141593’
RCHAR(2*PI,‘E10.4’)	‘0.6283E+01’

Strings may be appended together using the append operator, `//`. Scalar *values* can also be appended to strings using the `RCHAR` function. For example, suppose `A = -1.234`, and `T` is a string variable with `T = ‘ units’`.

function	*result*
‘The value of A is ’//RCHAR(A)//T	‘The value of A is -1.234 units’
The value of A is ’//RCHAR(A,‘F4.1’)//T	‘The value of A is -1.2 units’

TRANSLATE

Syntax `out_string = TRANSLATE(in_string)`

The `TRANSLATE` function accepts a string as argument, and returns a string which is:

VMS: the translation of the logical name `‘in_string’`

UNIX: the translation of the environment variable `‘in_string’`

If `‘in_string’` has no translation, `TRANSLATE` simply returns it unchanged.

Examples

The following shows one way in which the `TRANSLATE` function could be used.

```
read translate(‘data_dir’)//‘file.dat’ x y z ! pre-define data_dir
```

Numeric functions with string arguments

CLEN

Syntax `scalar = CLEN(string)`

The `CLEN` function accepts only a string as argument. It returns the number of characters in the string. The string can be a literal quote string or a string variable. It cannot be a string array variable.

ICHAR

Syntax `vector = ICHAR(string)`

The ICHAR function accepts a string as argument. The string cannot be a complete array string variable, but it can be an element of an array string variable. It returns a numeric vector whose elements are the equivalent ASCII decimal codes for the characters. If the string is only one character, the resultant vector will be of length one. The inverse of this function is the CHAR function.

EQS

Syntax `scalar = EQS(string1,string2)`

The EQS function accepts two strings as arguments. If the two strings are exactly equal it returns one (1), otherwise it returns zero (0).

NES

Syntax `scalar = NES(string1,string2)`

The NES function accepts two strings as arguments. If the two strings are exactly equal it returns zero (0), otherwise it returns one (1).

SUB

Syntax `scalar = SUB(string1,string2)`

The SUB function accepts two strings as arguments. If string1 is a subset of string2 it returns one (1), otherwise it returns zero (0).

SUP

Syntax `scalar = SUP(string1,string2)`

The SUP function accepts two strings as arguments. If string2 is a subset of string1 it returns one (1), otherwise it returns zero (0).

INDEX

Syntax `scalar = INDEX(string1,string2)`

The INDEX function accepts two strings as arguments and returns a scalar value. If string2

is a subset of string1 it returns the substring's starting position. If string2 occurs more than once in string1, the starting position of the first (leftmost) occurrence is returned. If string2 does not occur in string1, the value zero (0) is returned.

Example

function	*result*
INDEX('abc','abc')	1
INDEX('abcd','abc')	1
INDEX('abc','abcd')	0
INDEX('xxabcabc','abc')	3

EVAL

Syntax `numeric = EVAL(string)`

string variables can be used in numeric expressions, as so called expression variables, to shorten or to simplify an expression. Parentheses around the expression variable are assumed during numeric evaluation. For example:

```
T='A+B'
Y=X*T      ! this is equivalent to Y=X*(A+B)
```

A string variable will be numerically evaluated if it is a numeric operand or the argument of a numeric function. Otherwise, a string variable is treated as a string. Use the EVAL function to force numeric evaluation. The type of result, that is, scalar, vector, or matrix, depends on the evaluated expression.

Example

Suppose that string variable `T='3+2'`.

input	*result*
=T	the string '3+2'
=EVAL(T)	the numeric value 5

Numeric analysis functions

AREA

Syntax `scalar = AREA(vector1,vector2)`

The AREA function calculates the area enclosed in the polygon specified by the vertex coordinates given in the vector arguments. Both vector arguments must be of the same length. The polygon need not be closed, that is, the last point will be assumed to connect to the first point.

DERIV

Syntax	`vector = DERIV(vector1,vector2)` `vector = DERIV(vector1,vector2,'keyword')`
Keywords	`SMOOTH, INTERP, FC, LAGRANGEn (n=3,5,7,9)`
Default	`SMOOTH`

The `DERIV` function evaluates the first derivatives of the vector `y`, the dependent variable, with respect to the vector `x`, the independent variable, at the `x` locations. The vector `x` must be strictly monotonically increasing. The result of this function is a vector with the same length as the vectors `x` and `y`. The algorithm that is employed depends on the `keyword` that is used. By default, the derivatives are calculated using smoothing cubic splines.

Smoothing splines

Syntax	`vector = DERIV(vector1,vector2)` `vector = DERIV(vector1,vector2,'SMOOTH')`

By default, or if the `SMOOTH` keyword is used, then smoothing cubic splines, which may not pass through the data points, are used. The spline tension will be the current value of `TENSION`, which may be changed with the `SET` command. A tension of zero gives the loosest splines, while a large tension gives linear interpolation. The default tension is 1.

Interpolating splines

Syntax	`vector = DERIV(vector1,vector2,'INTERP')`

Cubic splines, which always pass through the data points, are used. The spline tension will be the current value of `TENSION`, which may be changed with the `SET` command. A tension of zero gives the loosest splines, while a large tension gives linear interpolation. The default tension is 1.

Lagrange polynomials

Syntax	`vector = DERIV(vector1,vector2,'LAGRANGE3')` `vector = DERIV(vector1,vector2,'LAGRANGE5')` `vector = DERIV(vector1,vector2,'LAGRANGE7')` `vector = DERIV(vector1,vector2,'LAGRANGE9')`

If the `LAGRANGEn` keyword is used, where `n` can be `3, 5, 7`, or `9`, the derivatives are calculated using the method of Lagrange interpolating polynomials.

Monotone piecewise cubic interpolation

Syntax	`vector = DERIV(vector1,vector2,'FC')`

If the `FC` keyword is used, the derivatives are calculated using the Fritsch and Carlson method of monotone piecewise cubic interpolation. This algorithm produces a visually pleasing interpolant, that is, the interpolating curve has no extraneous "bumps" or "wiggles". For an explanation of this method, see:
SIAM Journal of Numerical Analysis, volume 17, number 2, April 1980.

Example

Suppose that you want to see the derivatives, from 0 to π, of $cos(x)^3 + sin(x)^4$. The following commands produce Figure 4.33.

```
LEGEND ON
LEGEND TRANSPARENCY OFF
LEGEND\GRAPH FRAME .25 -2.25 2 -1
SET PCHAR 1
X=[0:PI:.1]
GRAPH 'cos(x)<^>3<_>+sin(x)<^>4<_>' X COS(X)^3+SIN(X)^4
SET PCHAR 2
GRAPH\NOAXES 'derivative' X DERIV(X,COS(X)^3+SIN(X)^4,'LAGRANGE5')
REPLOT
```

INTEGRAL

Syntax	`vector = INTEGRAL(vector1,vector2)` `vector = INTEGRAL(vector1,vector2,'keyword')`
Keywords	`SMOOTH`
Default	`SMOOTH`

The `INTEGRAL` function integrates the vector, `vector2`, the dependent variable, with respect to vector `vector1`, the independent variable. `vector1` must be strictly monotonically increasing. The output of this function is a vector with the same length as `vector1` and `vector2`. The last element of the output vector is the integral over the full range of `vector1`. The method used depends on the keyword. Currently, there is only one type of integration available.

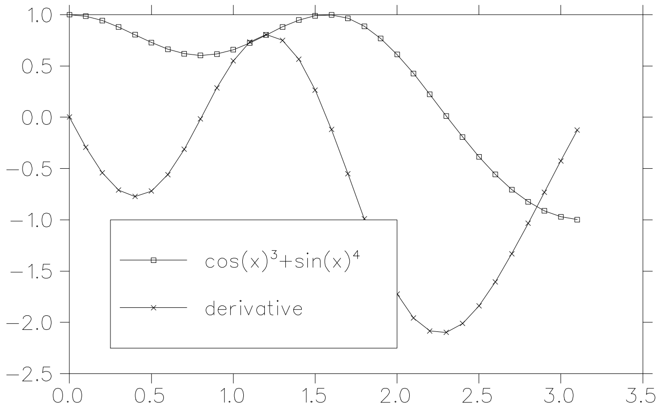


Figure 4.33: An example illustrating the `DERIV` function

Smoothing splines

Syntax	`vector = INTEGRAL(vector1,vector2)`
	`vector = INTEGRAL(vector1,vector2,‘SMOOTH’)`

By default, the integration method utilizes an interpolating spline under tension. The spline tension used is the current value of `TENSION`, which may be changed with the `SET` command. The nature of the interpolating curve varies continuously from pure cubic splines, for `TENSION` $= 0$, to a piecewise linear curve, that is, points joined by straight line segments, for large `TENSION`.

Example

Suppose you need to see the integral values, from 0 to π, of $\cos(x)^3 + \sin(x)^4$. The following sequence of commands could be used. See Figure 4.34


```
LEGEND ON
LEGEND TRANSPARENCY OFF
LEGEND\GRAPH FRAME .25 -.75 2 .25
SET PCHAR 1
X=[0:PI:.1]
GRAPH ‘cos(x)<^>3<_>+sin(x)<^>4<_>’ X COS(X)^3+SIN(X)^4
SET PCHAR 2
GRAPH\NOAXES ‘integral’ X INTEGRAL(X,COS(X)^3+SIN(X)^4)
REPLOT
```

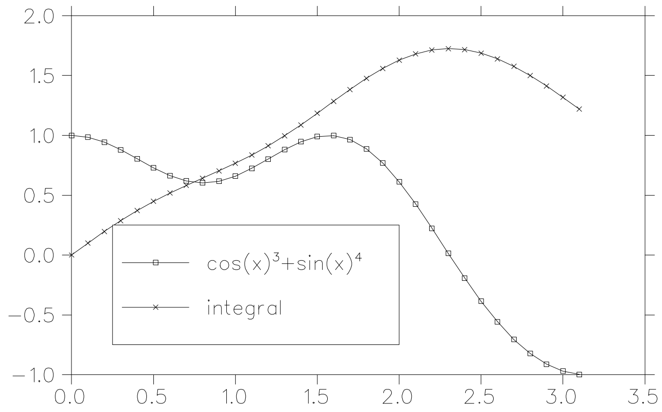


Figure 4.34: An example illustrating the `INTEGRAL` function

GAUSSJ

Syntax	`vector = GAUSSJ(matrix,vector)`

The `GAUSSJ` function solves the system of equations `m<>x = b`, where `m` is a square matrix and `b` is a vector, and returns `x`, the vector of solutions. This function uses the Gauss-Jordan method of elimination with full pivoting. The matrix must be square. The length of the input vector must be the same as the row dimension of the matrix. The function returns a vector with the same length.

Example 1

To solve the following three equations for the three unknowns x_1, x_2, and x_3:

$$
\begin{aligned}
3x_1 + 4x_2 + 15x_3 &= 26 \\
10x_1 + 2x_2 + 3x_3 &= 14 \\
5x_1 - 4x_2 + 3x_3 &= 22
\end{aligned}
$$

You could use the script:

```
m=[[3;1;5];[4;2;4];[5;3;3]]
b=[26;14;22]
soln=gaussj(m,b)
```

The answer: `soln=[1.238095;-2.365079;2.116402]` can be checked by using the outer product operator: `m<>soln`
and comparing this result with the original `b` vector.

Example 2

The following script, `INVERSE.PCM`, will find the inverse of a square matrix. If you have a square matrix `M`, you could find it's inverse, `INV_M`, with the command: `@INVERSE M`

```
if ( vlen(?1)[1] "NE" vlen(?1)[2] ) then
  display 'input matrix must be square'
  return
endif
n = vlen(?1)[1]
! identity(n) is the identity matrix of order n
do k = [1:n]
 inv_m[1:n,k] = gaussj(?1,identity(n)[1:n,k])
enddo
```

INVERSE

Syntax `matrix = INVERSE(matrix)`

The function `INVERSE(m)` returns the inverse of the matrix `m`, which *must* be a square matrix. The output is a matrix with the same shape as the argument. The answer can be checked by using the inner product operator, for example:

```
invm=INVERSE(m) ! find the inverse of m
=m<>invm        ! this should be close to the identity matrix
```

Method

Suppose that the matrix A has n rows and n columns. Let X represent the inverse of A, and let I be the identity matrix:

$$I_{i,j} = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{otherwise} \end{cases}$$

The LU decomposition method[2] is used for finding the inverse matrix X. Write A as the product of two matrices: $A = L<>U$ where L is lower triangular and U is upper triangular. A lower triangular matrix has elements only on the diagonal and below, while an upper triangular matrix has elements only on the diagonal and above. This decomposition is used to solve n sets of n linear equations. The matrix subscript $*, j$ represents the entire j_{th} column of that matrix.

$$A<>X_{*,j} = (L<>U)<>X_{*,j} = L<>(U<>X_{*,j}) = I_{*,j} \quad \text{for each} \quad j = 1, 2, \ldots, n$$

Solve for the y vectors, there will be n of them, such that $L<>y = I_{*,j}$ and then solve for the j_{th} column of X: $U<>X_{*,j} = y$ for each $j = 1, 2, \ldots, n$

Since L and U are triangular

$$
\begin{aligned}
y_1 &= I_{1,j}/L_{1,1} \\
y_i &= \left[I_{i,j} - \textstyle\sum_{k=1}^{i-1} L_{i,k} y_k \right] / L_{i,i} \quad \text{for} \quad i = 2, 3, \ldots, n
\end{aligned}
$$

and

$$
\begin{aligned}
X_{n,j} &= y_n/U_{n,n} \\
X_{i,j} &= \left[y_{i,j} - \textstyle\sum_{k=i+1}^{n} U_{i,k} X_{k,j} \right] / U_{i,i} \quad \text{for} \quad i = n-1, n-2, \ldots, 1
\end{aligned}
$$

DET

Syntax `scalar = DET(matrix)`

The function `DET(m)` returns the determinant of the matrix `m`, which *must* be a square matrix. The output is a scalar.

Beware: The determinant of a reasonably sized matrix can get very large, or very small, leading to over/underflows.

The method used to find the determinant uses the LU decomposition of the matrix argument. Please refer to the discussion on LU decomposition of a matrix in the `INVERSE` function section, page 321.

[2]The definitions used here are taken from "Numerical Recipes – The Art of Scientific Computing" by W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, Cambridge University Press, 1986.

IDENTITY

Syntax `matrix = IDENTITY(scalar)`

The function `IDENTITY(n)` returns the identity matrix of order n. That is, the $n \times n$ matrix with 1's on the diagonal and zeros elsewhere. For example:

$$\texttt{IDENTITY(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

EIGEN

Syntax `matrix = EIGEN(matrix)`

If matrix `m` is an $n \times n$ symmetric matrix, then `EIGEN(m)` returns a matrix with n rows and $n+1$ columns. Column $n+1$ contains the eigenvalues, while columns 1 to n are the eigenvectors of the symmetric matrix `m`. The eigenvector `x` and the eigenvalue `s` of matrix `m` satisfy the equation: `m<>x = s*x`.

One way to check a result is with the following script:

```
e=EIGEN(m)
n=vlen(m)[1]
DO j = [1:n]
! these should be all zero (or close to zero)
  =m<>e[*,j]-e[j,n+1]*e[*,j]
ENDDO
```

Example

The following script:

```
m=[[2;-1;0;0];[-1;2;-1;0];[0;-1;2;-1];[0;0;-1;2]]
e=eigen(m)
display ‘matrix m’
write\matrix\format sys$output (4f9.5) m
display ‘matrix e’
write\matrix\format sys$output (5f9.5) e
```

produces:

Functions

```
matrix m
 2.00000 -1.00000  0.00000  0.00000
-1.00000  2.00000 -1.00000  0.00000
 0.00000 -1.00000  2.00000 -1.00000
 0.00000  0.00000 -1.00000  2.00000
matrix e
 0.37175  0.60150  0.60150 -0.37175  0.38197
 0.60150  0.37175 -0.37175  0.60150  1.38197
 0.60150 -0.37175 -0.37175 -0.60150  2.61803
 0.37175 -0.60150  0.60150  0.37175  3.61803
```

The eigenvalues are `e[*,5] = [0.38197;1.38197;2.61803;3.61803]`

The eigenvectors are `e[*,j]` for `j = [1:4]`

PFACTORS

Syntax `vector = PFACTORS(scalar)`

The `PFACTORS(n)` function returns a vector containing the prime factors of the scalar `n`. For example, if you enter `X=PFACTORS(420)` then `X=[2;2;3;5;7]`.

This function can be usefull in determining a good length for the input vector for the `FFT` function.

FFT

Syntax	`matrix = FFT(vector)`
	`matrix = FFT(vector,‘keyword’)`
Keywords	`AMP&PHASE, COS&SIN`
Default	`AMP&PHASE`

The `FFT` function calculates the discrete fast Fourier transform of the input variable, `vector`. By default, `FFT` returns the amplitudes and the phases, where the phases are in degrees. If the `COS&SIN` keyword is used, `FFT` returns the Fourier coefficients.

Note: The reason that the amplitudes and phases are returned by default is historical. Actually, the Fourier coefficients, that is, the cosine and sine coefficients, are calculated and the amplitudes and phases are just derived from them, as described below. It is a simple matter for the user to request the cosine and sine coefficients, and then to calculate the amplitudes and phases him/herself.

Suppose that the length of the input vector is $2N$. The output of this function is a matrix with $N+1$ rows and 2 columns. The first column contains the amplitudes (or the cosine coefficients), and the second column contains the phases (or the sine coefficients).

The `IFFT` function calculates the inverse fast Fourier transform.

Fourier coefficients

Syntax	`matrix = FFT(vector,'COS&SIN')`

If the `COS&SIN` keyword is used, then the `FFT` function returns the actual Fourier coefficients. Let that the cosine coefficients be called $\mathcal{H}$'s and the sine coefficients be called $\mathcal{G}$'s. $\mathcal{H}_0/2$ is the mean value of the input data.

As shown in the example subsection below, these coefficients can be used for smooth interpolation. Suppose x_i is the interpolation location, and $2N$ is the number of original data points.

$$y_i = \frac{\mathcal{H}_0}{2} + \sum_{k=1}^{N} (\mathcal{H}_k \cos(kx_i) + \mathcal{G}_k \sin(kx_i))$$

Discrete Fourier series

Given $2N$ samples of real data y_j (where $j = 0, 1, 2, \ldots, 2N-1$) taken at equally spaced intervals $\Delta t = T/(2N)$, where T is the period, the corresponding Fourier series is:

$$y(t) = \frac{\mathcal{H}_0}{2} + \sum_{k=1}^{N} [\mathcal{H}_k \cos(2\pi kt/T) + \mathcal{G}_k \sin(2\pi kt/T)]$$

where $\mathcal{G}_0 = \mathcal{G}_N = 0$ and $\mathcal{H}_0/2$ is the mean value of y.

From the original $2N$ data points, we have exactly $2N$ calculated coefficients, that is, we have $(N+1)$ $\mathcal{H}$'s and $(N-1)$ $\mathcal{G}$'s with "real" information.

If $t = 0$ at y_0, then for each y_j we have $t = jT/N$, for $j = 0, 1, \ldots, 2N-1$, and so

$$y_j = \frac{\mathcal{H}_0}{2} + \sum_{k=1}^{N} [\mathcal{H}_k \cos(jk\pi/N) + \mathcal{G}_k \sin(jk\pi/N)]$$

The amplitude, A, and the phase, P, are calculated as follows:

$$A_0 = \mathcal{H}_0 \quad A_N = \mathcal{H}_N$$

$$A_j = \sqrt{\mathcal{H}_j^2 + \mathcal{G}_j^2} \quad \text{for} \quad j = 1, 2, \ldots, N-1$$

$$P_0 = P_N = 0$$

$$P_j = \frac{180}{\pi}\arctan(\mathcal{G}_j/\mathcal{H}_j) \quad \text{for} \quad j = 1, 2, \ldots, N-1 \quad \text{and if} \quad P < 0^\circ \quad \text{then} \quad P = P + 360^\circ$$

Restrictions

The input vector y is assumed to be periodic with an *even* number of points, that is,

$$\texttt{LEN}(y) = 2N > 2, \quad \text{and} \quad y_0 = y_{2N} = \cdots$$

Prime factors

Suppose that $\texttt{LEN}(y) = 2N$. The largest prime factor of $2N$ must be ≤ 23, and there must be no more than 11 distinct prime factors. The product of the square-free prime factors of $2N$ must be ≤ 210. The calculation speed is enhanced by using a value of $2N$ with small prime factors, particularly powers of 2. For reference, the prime numbers less than 1000 are listed below.

2	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53	59	61	67	71	73	79
83	89	97	101	103	107	109	113	127	131	137	139	149	151	157	163	167	173	179	181	191	193
197	199	211	223	227	229	233	239	241	251	257	263	269	271	277	281	283	293	307	311	313	317
331	337	347	349	353	359	367	373	379	383	389	397	401	409	419	421	431	433	439	443	449	457
461	463	467	479	487	491	499	503	509	521	523	541	547	557	563	569	571	577	587	593	599	601
607	613	617	619	631	641	643	647	653	659	661	673	677	683	691	701	709	719	727	733	739	743
751	757	761	769	773	787	797	809	811	821	823	827	829	839	853	857	859	863	877	881	883	887
907	911	919	929	937	941	947	953	967	971	977	983	991	997								

For example, $2N = 202 = 2 \times 101$ is not allowed, and $2N = 402 = 2 \times 3 \times 67$ is also not allowed. If $2N$ does not satisfy the above restrictions, the input vector can be *padded out*, usually with zeroes, to an even length whose prime factors do not exceed 23. The `PFACTORS` function, page 324, returns the prime factors of a constant or scalar.

Example

The following script demonstrates how you can use the `FFT` function to smooth data. Note that when all the Fourier coefficients are used, the smoothed curve must pass through the original data points. See Figure 4.35.

```
N = 16                ! even number of points
X = [0:N-1]*2*PI/N    ! generate some "data"
Y = SIN(X)+5*RAN(X)   !
M = FFT(Y,'COS&SIN')  ! calculate Fourier coefficients
H = M[*,1]            ! extract column 1 as a vector
G = M[*,2]            ! extract column 2 as a vector
Z = [0:2*PI:.05]      !
SCALAR\DUMMY K        ! define K to be dummy variable for SUM function
SCALE 0 6 0 -2 7 0    ! set axis scales
SET %XLABSZ 5         ! increase the size of the x-axis label
DO J = [3:N/2+1:2]
  WINDOW (J-3)/2+15   ! choose a graphics window
  SET PCHAR -1        ! choose plotting symbol
  LABEL\XAXIS 'Number of harmonics used = '//RCHAR(J)
  GRAPH X Y           ! plot original data
  SET PCHAR  0        ! choose no plotting symbol
  GRAPH\NOAXES Z H[1]/2+SUM(H[K]*COS((K-1)*Z)+G[K]*SIN((K-1)*Z),K,2:J)
ENDDO
GRAPH\NOAXES [0;2*PI] [H[1]/2;H[1]/2] ! overlay the mean value
```

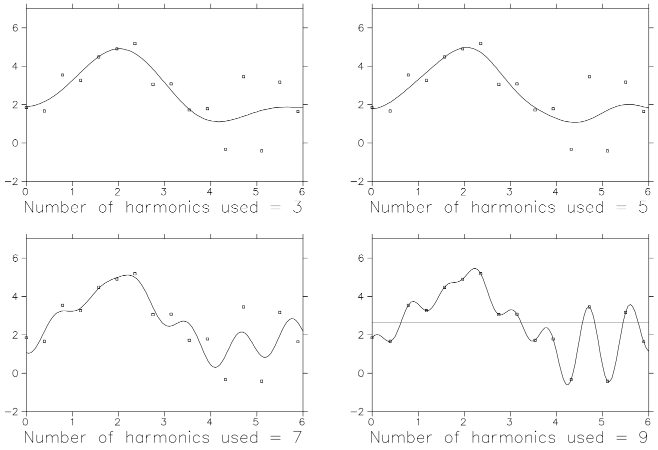


Figure 4.35: An FFT example showing data smoothing

IFFT

Syntax	`vector = IFFT(matrix)`
	`vector = IFFT(matrix,'keyword')`
Keywords	`AMP&PHASE, COS&SIN`
Default	`AMP&PHASE`

The IFFT function calculates the inverse discrete Fourier transform of the two column input matrix. This matrix is usually calculated by the FFT function, thus reconstructing the original data.

By default, IFFT expects amplitudes and phases, where the phases are in degrees. The first column of the matrix should contain the amplitudes and the second column the phases. If the COS&SIN keyword is used, IFFT expects the Fourier coefficients, that is, the cosine coefficients in the first column and the sine coefficients in the second column. If the input matrix has N rows, the function returns a vector with length $2(N-1)$.

The principle usage of the IFFT function is to modify some of the amplitudes returned from the FFT function and note their effect on the original data. A typical application would be one of data smoothing, in which the user would zero out the amplitudes of the higher order harmonics.

CONVOL

Syntax	`vector = CONVOL(vector1,vector2,scalar)`

The CONVOL function accepts vectors as the first two arguments. It convolutes or deconvolutes the input vector, `vector1`, with the specified blurring vector, `vector2`. The result is a vector the same length as `vector1`. The third argument should be a scalar, and indicates which operation is to be performed.

Convolution of an odd number of points

Syntax	`vector = CONVOL(x,b,0)`

The blurring vector, b, must contain an odd number of points. Suppose that N is the length of b and M is the length of x. The convolution of x with b is:

$$y_i = \sum_{j=1}^{N} x_{i-\frac{N}{2}+j-1} \cdot b_{N-j+1}$$

$$= x_{i-\frac{N}{2}} b_N + x_{i-\frac{N}{2}+1} b_{N-1} + \cdots + x_{i+\frac{N}{2}-1} b_2 + x_{i+\frac{N}{2}} b_1$$

for $i = 1, 2, 3, \ldots, M$. References to subscripts out of the range of x are not summed. The blurring vector is normalized to 1 to insure that the integrals of the y and x are identical. This normalization is internal, the blurring vector is returned unchanged. To ensure proper convolution, x should be padded at it's upper and lower ends with zeros so it's length is at least the minimum of:

the non-zero length of b; and $\frac{1}{2}$ the length of b

Note: The lengths of b and x can differ. To avoid centroid shifts in the output, centre the blurring vector properly. For example, suppose that b has $2N-1$ elements containing a gaussian, then it's peak should be at N.

Convolution of an even number of points

Syntax	`vector = CONVOL(x,b,1)`

The blurring vector, b, should contain an even number of points. The preferred lengths are powers of 2. The convolution is done using fast Fourier transforms. The following restrictions apply:

- x must be padded at it's lower end with zeros with the number of elements which are non-zero in b, for example, if x and b are of length 128, and b is zero in the range 30 – 128, then x must contain zeros in locations 1 – 29
- x and b must have the same length
- the end points of b must not be equal. A difference of less than 0.0001 produces oscillations in the deconvoluted result. The usual way is to shift b to the left so that the first point has a non-zero value. Together with the first restriction, this ensures that the right point has the value zero, leaving the ends unequal.

Noise in b produces a change in the output, which, due to averaging, has a small effect. Noise effects depend on the shape of the deconvoluted peak.

Convolution noise effects

The narrower this peak, the more effect the noise in b has. This occurs because each noisy point becomes a greater percentage of the total number in the convoluting or deconvoluting function, thus reducing the average effect. In many applications, the noise in the measured data is statistical in nature and so, to reduce the sensitivity to this noise on the deconvolution, apply smoothing filters on the measured data before deconvolution.

Deconvolution of an even number of points

Syntax	`vector = CONVOL(x,b,-1)`

The blurring vector, b, should contain an even number of points. The preferred lengths are powers of 2. The deconvolution is done using fast Fourier transforms. The following restrictions apply:

1. x must be padded at it's lower end with zeros with the number of elements which are non-zero in b, for example, if x and b are of length 128, and b is zero in the range 30 – 128, then x must contain zeros in locations 1 – 29
2. x and b must have the same length
3. the end points of b must not be equal. A difference of less than 0.0001 produces oscillations in the deconvoluted result. The usual way is to shift b to the left so that the first point has a non-zero value. Together with the first restriction, this ensures that the right point has the value zero, leaving the ends unequal.

Noise in b produces a change in the output, which, due to averaging, has a small effect. Noise effects depend on the shape of the deconvoluted peak. The narrower this peak, the more effect the noise in b has. This occurs because each noisy point becomes a greater percentage of the total number in the convoluting or deconvoluting function, thus reducing the average effect. In many applications, the noise in the measured data is statistical in nature and so, to reduce the sensitivity to this noise on the deconvolution, apply smoothing filters on the measured data before deconvolution.

INTERP

Syntax	`vector = INTERP(vector1,vector2,vector3)` `vector = INTERP(vector1,vector2,vector3,'keyword')`
Keywords	`SPLINE, LINEAR, FC, LAGRANGE`
Default	`SPLINE`

The `INTERP` function interpolates the data contained in `vector1`, the independent variable, and `vector2`, the dependent variable. `vector1` must be strictly monotonically increasing. The interpolant locations are given in `vector3`. The `INTERP` function will return the interpolated values as a vector with the same length as `vector3`. The algorithm that is employed depends on the keyword that is used. The default is use interpolating splines.

An interpolated curve will always pass through the original data points. If it is not important that the curve pass through the original data, use the `SMOOTH` function. If your independent variable is not monotonically increasing, use the `SPLINTERP` function.

Spline interpolation

Syntax `vector = INTERP(x,y,xi)`
`vector = INTERP(x,y,xi,'SPLINE')`

By default, or if the SPLINE keyword is used, the interpolant is calculated by the method of cubic splines under tension. The tension factor corresponds to the "curviness", and must be greater than zero. If it is close to zero, each interpolated function is almost a cubic spline and the resulting curve is quite "loose". If the tension is large, then the resultant is almost linear. The tension used is the current value of TENSION, which may be changed with the SET command.

Linear interpolation

Syntax `vector = INTERP(x,y,xi,'LINEAR')`

If the LINEAR keyword is used, the interpolating method used is linear interpolation.

Lagrange interpolation

Syntax `vector = INTERP(x,y,xi,'LAGRANGE')`

If the LAGRANGE keyword is used, the interpolating method used is general Lagrange interpolation. The degree of the interpolating polynomial depends on the number of input data points. Suppose that N is the number of points, equal to the lengths of x and y. If $N = 2$, then a simple straight line is used for interpolating. If $N = 3$ or $N = 4$, then a quadratic is used. If $N \geq 5$, then a polynomial of degree 4 is used for the interpolation.

Interpolation by monotone piecewise cubic polynomials

Syntax `vector = INTERP(x,y,xi,'FC')`

If the FC keyword is entered, the interpolant is calculated using the Fritsch and Carlson method of monotone piecewise cubic interpolation. This algorithm produces a visually pleasing interpolant, that is, the interpolating curve has no extraneous "bumps" or "wiggles".

For an explanation of this method, see:
SIAM Journal of Numerical Analysis, volume 17, number 2, April 1980.

SPLINTERP

Syntax `vector = SPLINTERP(vector1,vector2,scalar)`

The SPLINTERP function interpolates the data contained in vector1, the independent variable, and vector2, the dependent variable. vector1 need not be monotonically increasing. The interpolated curve will always pass through the original data points. The number of output interpolant locations is given in scalar. Suppose scalar $= n$. The output of this function is a matrix with n rows and 2 columns. The first column will contain the output locations and the second column the interpolated values.

The points are first parameterized in terms of normalized arc length. The normalized length of x is the real length divided by the range of x, that is, the maximum value minus the minimum value. The arclength at a point is approximated by the sum of the lengths of straight line segments connecting all points up to that point. A spline under tension is calculated for vector1 versus arc length and vector2 versus arc length. The vector1 and vector2 values are interpolated separately and then combined to form the output interpolant.

The tension factor corresponds to the "curviness", and must be greater than zero. If it is close to zero, each interpolated function is almost a cubic spline and the resulting curve is quite "loose". If the tension is large, then the resultant is almost linear. The tension used is the current value of TENSION, which may be changed with the SET command.

For monotonically increasing data, use the INTERP function.

SMOOTH

Syntax `vector = SMOOTH(vector1,vector2,vector3)`
`vector = SMOOTH(vector1,vector2,vector3,vector4)`

The SMOOTH function calculates a smooth curve through the data contained in vector1, the independent variable, and vector2, the dependent variable. vector1 *must* be strictly monotonically increasing. The output locations are given in vector3, which must also be monotonically increasing. This function returns the smoothed values as a vector with the same length as vector3. The smooth curve is calculated by the method of cubic splines under tension.

For another smoothing method, using Savitzky-Golay filters, use the SAVGOL function. Depending on the tension, a smoothed curve may not pass through the original data points. If you want the curve to always pass through the original data, use the INTERP function. If your data is not monotonically increasing, use the SPLSMOOTH function.

Weights

Syntax `vector = SMOOTH(x,y,xout,w)`

If no weights, w, are entered, the weight at each data point defaults to 1. The weights control the amount of smoothing at each data point. As the weight at a point decreases, the spline fits that data point more closely.

Spline tension

The tension factor corresponds to the "curviness", and must be greater than zero. The tension used is the current value of `TENSION`, which may be changed with the `SET` command.

If the tension is set to zero, the result will be an interpolating cubic spline. If the tension is large, the result will be the least-squares line through the data. As the tension decreases, the amount of smoothing decreases and the data points are fit more exactly. As the tension increases, the fit straightens and has less curvature at peaks, valleys and endpoints.

Suppose that N is the length of y and that the weights are the standard deviations of y. Values of tension t in the range $N - \sqrt{2N} \le t \le N + \sqrt{2N}$ give the most natural looking results. To obtain the most suitable fit, the user may wish to do several runs with different values of t. By observing the spline fits plotted on a graph, the fit with the most suitable amount of smoothing can be selected.

Method

Given a set of abscissae: $x_1 < x - 2 < \ldots < x_N$,

a cubic spline function over the region (x_1, x_N) is composed of cubic parabolas

$$\mathcal{G}(x) = a_i + b_i \cdot (x - x_i) + c_i \cdot (x - x_i)^2 + d_i \cdot (x - x_i)^3$$

where $x_i \le x < x_{i+1}$, which join at the endpoints x_i such that $\mathcal{G}$, $\mathcal{G}'$, and $\mathcal{G}''$ are continuous.

The smoothing function is constructed by minimizing $\int_{x_1}^{x_N} \mathcal{G}''(x)^2 dx$
subject to the constraint $\sum_{i=1}^{N} \left((\mathcal{G}(x_i - y_i)/w_i\right)^2 \le t$
where $w_i > 0$ are the weights and $t \ge 0$ is the spline tension.

The solution proceeds by the standard methods of minimizing the functional

$$\int_{x_1}^{x_N} \mathcal{G}''(x)^2 dx + p \cdot \left\{ \sum_{i=1}^{N} ((\mathcal{G}(x_i) - y_i)/w_i)^2 + z^2 - t \right\}$$

where z and p are auxiliary parameters. The functional is minimized with respect to z and p by setting the partial derivatives with respect to z and p equal to zero.

SPLSMOOTH

Syntax `matrix = SPLSMOOTH(vector1,vector2,scalar)`
`matrix = SPLSMOOTH(vector1,vector2,scalar,vector3)`

The `SPLSMOOTH` function calculates a smooth curve through the data contained in `vector1`, the independent variable, and `vector2`, the dependent variable. `vector1` need not be monotonically increasing. The number of output locations is given in `scalar`. Suppose `scalar` $= n$. The output of this function is a matrix with n rows and 2 columns. The first column will contain the output locations and the second column the smoothed values.

The points are first parameterized in terms of normalized arc length. The normalized length of x is the real length divided by the range of x, that is, the maximum value minus the minimum value. The arclength at a point is approximated by the sum of the lengths of straight line segments connecting all points up to that point. A spline under tension is calculated for `vector1` versus arc length and `vector2` versus arc length. The `vector1` and `vector2` values are interpolated separately and then combined to form the output interpolant.

The tension factor corresponds to the "curviness", and must be greater than zero. If it is close to zero, each interpolated function is almost a cubic spline and the resulting curve is quite "loose". If the tension is large, then the resultant is almost linear. The tension used is the current value of `TENSION`, which may be changed with the `SET` command.

For monotonically increasing data, use the `SMOOTH` function.

Weights

Syntax `matrix = SPLSMOOTH(x,y,n,w)`

If m is the length of w, x and y, the goal is to achieve: $\sum_{i=1}^{m}((G_i - y_i)/w_i)^2 = s$
where G_i is the cubic spline function at x_i and s is the spline tension. As w_i decreases, the spline fits the data point (x_i, y_i) more closely.

SAVGOL

Syntax `vector = SAVGOL(scalar1,scalar2,vector)`

The `SAVGOL` function calculates a smooth curve through the data contained in `vector`, the dependent variable, using the Savitzky-Golay smoothing filter method. The order of the filter is given in `scalar1`, and can be 2 or 4. The filter width is given in `scalar2`. These filters preserve the area under the data, the zeroth moment, but also the higher moments.

There is no input independent variable, as the data is assumed to be equally spaced.

As a rough guideline, best results are obtained when the filter width, m, of the order 4 Savitzky-Golay filter is between one and two times the full width half maximum (FWHM) of desired features in the data.

The Savitzky-Golay smoothing method

A nonrecursive filter is defined by the convolution formula: $yout_i = \sum_{j=-N_L}^{N_U} c_j y_{i+j}$

The c_j's are the coefficients of the filter, the y's are the input data, and the $yout$'s are the outputs. The set of points y_{i-N_L} to y_{i+N_U} define the moving window of the filter, where $N_L = m/2$ and $N_U = m - N_L - 1$. The Savitzky-Golay smoothing method finds filter coefficients that preserve the 0^{th}, the 1^{st}, the 2^{nd}, and higher moments. The idea is to approximate the underlying function within a moving filter window by a polynomial of order 2, or of order 4. For each data point, y_i, least-squares fit a polynomial to all m points within the filter window, and then set $yout_i$ to be the value of that polynomial at position i. No use is made of the value of that polynomial at any other point. For the next point, y_{i+1}, a whole new least-squares fit is done using the shifted window.

Since the process of least-squares fitting involves only a linear matrix inversion, the coefficients of a fitted polynomial are themselves linear in the values of the data. All the least-squares fitting can be done in advance, for fictitious data consisting of all zeros except for a single 1, and then the fits on the real data are done by taking linear combinations. Thus, there are particular sets of filter coefficients, c_j, for which the convolution formula automatically accomplishes the process of polynomial least-squares fitting inside a moving window.

The Savitzky-Golay filters provide smoothing without loss of resolution when the underlying function can be locally well fitted by a polynomial. This is true for smooth line profiles not too much narrower than the filter width. When this is not true, these filters have no advantage over other smoothing methods.

If the data is irregularly sampled, that is, the y values are not equally spaced, one can pretend that the data points *are* equally spaced. This amounts to virtually shifting, within each moving window, the data points to equally spaced positions. Such a shift introduces the equivalent of an additional source of noise into the function values. In those cases where smoothing is useful, this noise will often be much smaller than the noise already present. Specifically, if the location of the points is approximately random within the window, then a rough criterion is this: If the change in y across the full width of the window, m, is less than $\sqrt{m}/2$ times the measurement noise on a single point, then this implementation of the Savitzky-Golay filter can be used.

For more information on Savitzky-Golay filters, see:
Computers in Physics, volume 4, number 6, November/December 1990, pages 669 – 672.

JOIN

Syntax `scalar = JOIN(vector1,vector2)`

The arguments of the `JOIN` function must both be vectors. `JOIN` produces a matrix with 3 columns. The first column is the intersection of `vector1` and `vector2`, that is, if you enter `m=join(x,y)` then `m[*,1]` is the same as `x/&y`. `m[i,2]` is the index of `x` from which `m[i,1]` was taken, and `m[i,3]` is the index of `y` from which `m[i,1]` was taken. If the vector arguments are ordered, the `JOIN` function will proceed much faster than if they are unordered.

Example

Suppose that you have two vectors:

`X = [0;1;2;3;4;5;6;7;8;9;10], Y = [1;3;5;7;9]`

then `JOIN(X,Y)` $= \begin{pmatrix} 1 & 2 & 1 \\ 3 & 4 & 2 \\ 5 & 6 & 3 \\ 7 & 8 & 4 \\ 9 & 10 & 5 \end{pmatrix}$

Functions that return a variable's characteristics

EXIST

Syntax `scalar = EXIST(string)`

The `EXIST` function accepts one string as its argument and returns a one (1) if the string is an existing variable name, and zero (0) is it is not.

Example

If you enter `X2=3`, the function `EXIST('X2')` would have the value 1.

You can even enter: `N=2` and `T='X'//RCHAR(N)` and the function `EXIST(T)` would have the value 1.

LEN

Syntax `scalar = LEN(vector)`

The `LEN` function only accepts a vector as argument. It returns the length of the vector as a scalar.

VLEN

Syntax `vector = VLEN(vector)`
`vector = VLEN(matrix)`

The `VLEN` function accepts either a vector or a matrix as argument. It returns the length of each dimension of the argument. If the argument is a vector, the result is a vector of length one. If the argument is a matrix, the result is a vector of length two, with the first element being the number of rows and the second the number of columns.

FIRST

Syntax `scalar = FIRST(vector)`

The `FIRST` function returns the starting index for `vector`. The result of the `FIRST` function is a scalar. The length of vector `x` is `LAST(x)-FIRST(x)+1`, which is equal to `LEN(x)`.

LAST

Syntax `scalar = LAST(vector)`

The `LAST` function returns the final index for `vector`. The result of the `LAST` function is a scalar. The length of vector `x` is `LAST(x)-FIRST(x)+1`, which is equal to `LEN(x)`.

ICLOSE

Syntax `scalar = ICLOSE(vector,scalar)`

The `ICLOSE` function returns the index of `vector` which corresponds to where `vector` is closest to the value of `scalar`, that is, `ICLOSE` returns the *first* `j` where `|vector[j] - scalar|` is a minimum.

IEQUAL

Syntax `scalar = IEQUAL(vector,scalar)`

The `IEQUAL` function returns the index of `vector` which corresponds to where `vector` is equal to the value of `scalar`, that is, `IEQUAL` returns the *first* `j` where `|vector[j] = scalar|`. If `scalar` is not in the vector, `IEQUAL` returns zero (0).

WHERE

Syntax `vector = WHERE(vector)`

The `WHERE` function only accepts a vector as its argument. It returns the indices where this vector is not equal to zero.

Examples

function	*result*
`WHERE([-5:5]>0)`	`[7;8;9;10;11]`
`WHERE([-5:5]<=0)`	`[1;2;3;4;5;6]`

Suppose you have two vectors `X` and `Y` and you want to graph only those points that lie within the unit circle, that is, that satisfy `SQRT(X^2+Y^2)<=1`

```
IDX=WHERE(SQRT(X^2+Y^2)<=1)
GRAPH X(IDX) Y(IDX)
```

Shape changing functions

FOLD

Syntax `matrix = FOLD(vector,scalar)`

The `FOLD` function has two arguments. The first must be a vector, and the second a scalar. The result is a matrix formed by folding the data in the vector into the columns of a matrix. Suppose that vector x has m elements. $FOLD(x,n)_{i,j} = x_{i+(j-1)m}$ for $i = 1,2,\dots,n$ and $j = 1,2,\dots,m/n$. Note that m *must* be divisible by n.

Example

function	*result*
`FOLD([1:12],3)`	$\begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix}$
`FOLD([1:12],4)`	$\begin{pmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{pmatrix}$

UNFOLD

Syntax `vector = UNFOLD(matrix)`

The `UNFOLD` function has one argument, which must be a matrix. The result is a vector formed by unfolding the data in the rows of matrix. Suppose that matrix m has n_c columns and n_r rows. Then $UNFOLD(m)_{j+(i-1)n_c} = m_{i,j}$ for $i = 1,2,\dots,n_r$; and $j = 1,2,\dots,n_c$.

Example

Suppose that matrix M = $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$

function	*result*
UNFOLD(M)	$[1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12]$

ROLL

Syntax `vector = ROLL(vector,scalar)`
`matrix = ROLL(matrix,scalar)`

The `ROLL` function accepts either a vector or a matrix as its first argument. It shifts the elements of a vector or the rows of a matrix by the specified step size, `scalar`.

- If `scalar` $= n > 0$, the last n elements of the vector or the last n rows of the matrix are rolled around to the beginning.
- If `scalar` $= n = 0$, the vector or matrix is returned unchanged.
- If `scalar` $= n < 0$, the first n elements of the vector or the first n rows of the matrix are rolled around to the end.

If `scalar` is non-integral, then linear interpolation is used to generate new values.

Examples

function	*result*
`ROLL([1:10],2)`	`[9;10;1;2;3;4;5;6;7;8]`
`ROLL([1:10],-2)`	`[3;4;5;6;7;8;9;10;1;2]`
`ROLL([1:10],1.7)`	`[9.3;7.3;1.3;2.3;3.3;4.3;5.3;6.3;7.3;8.3]`

Suppose matrix M $= \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$

function	*result*
`ROLL(M,2)`	$\begin{pmatrix} 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 1 & 2 & 3 & 4 \end{pmatrix}$
`ROLL(M,-1)`	$\begin{pmatrix} 9 & 10 & 11 & 12 \\ 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$
`ROLL(M,1.5)`	$\begin{pmatrix} 7 & 8 & 9 & 10 \\ 5 & 6 & 7 & 8 \\ 3 & 4 & 5 & 6 \end{pmatrix}$

STEP

Syntax `vector = STEP(vector,scalar)`
`matrix = STEP(matrix,scalar)`

The `STEP` function accepts either a vector or a matrix as its first argument. It shifts the elements of a vector or the rows of a matrix by the specified step size, `scalar`.

- If `scalar` $= n > 0$, the last n elements of the vector or the last n rows are lost.
- If `scalar` $= n = 0$, the vector or matrix is returned unchanged.
- If `scalar` $= n < 0$, the first n elements of the vector or the first n rows are lost.

If `scalar` is non-integral, then linear interpolation is used to generate new values.

Examples

function	*result*
`STEP([1:10],2)`	`[1;1;1;2;3;4;5;6;7;8]`
`STEP([1:10],-2)`	`[3;4;5;6;7;8;9;10;10;10]`
`STEP([1:10],1.7)`	`[1;1;1.3;2.3;3.3;4.3;5.3;6.3;7.3;8.3]`

Suppose matrix M $= \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$

function	*result*
`STEP(M,2)`	$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}$
`STEP(M,-1)`	$\begin{pmatrix} 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 9 & 10 & 11 & 12 \end{pmatrix}$
`STEP(M,1.5)`	$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 3 & 4 & 5 & 6 \end{pmatrix}$

WRAP

Syntax `matrix = WRAP(matrix,scalar)`

The `WRAP` function accepts a matrix as its first argument. It wraps the column elements of a matrix by the specified step size, `scalar`.

- If `scalar` $= n > 0$, the column elements of the matrix are shifted down. The first n elements in the first column are zero filled, then the last n elements of each column are brought into the next column. The last n elements in the last column are lost.
- If `scalar` $= n = 0$, the matrix is returned unchanged.
- If `scalar` $= n < 0$, the column elements of the matrix are shifted up. The first n elements in the first column are lost. The first n elements of each column are brought into the preceding column. The last n elements in the last column are zero filled.

If `scalar` is non-integral, it is truncated to an integer.

Examples

Suppose matrix `M` $= \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$

function	*result*
`WRAP(M,2)`	$\begin{pmatrix} 0 & 5 & 6 & 7 \\ 0 & 9 & 10 & 11 \\ 1 & 2 & 3 & 4 \end{pmatrix}$
`WRAP(M,-2)`	$\begin{pmatrix} 9 & 10 & 11 & 12 \\ 2 & 3 & 4 & 0 \\ 6 & 7 & 8 & 0 \end{pmatrix}$

Looping functions

Looping functions mimic standard mathematical notation, for example, the sum:

$$\mathtt{SUM}(f(j), j, 1:N) \equiv \sum_{j=1}^{N} f(j)$$

Where j is the dummy variable.

The looping functions, `SUM`, `PROD`, `RSUM`, `RPROD`, and `LOOP`, require a previously declared scalar dummy variable as second argument. A dummy variable is declared with the `SCALAR\DUMMY` command. A dummy variable is different from other scalar variables in that its value is only defined while inside the looping function. The third argument of a looping function is always the range of this dummy variable, and must be a vector. The first argument of a looping function would normally be some function of the dummy variable, but it is not necessary that the dummy variable appear in the first argument.

SUM

Syntax
```
scalar = SUM(scalar,dummy,scalar)
vector = SUM(vector,dummy,vector)
matrix = SUM(matrix,dummy,matrix)
```

The `SUM` function requires a previously declared scalar dummy variable as second argument. A dummy variable is declared with the `SCALAR\DUMMY` command. The third argument is always the range of this dummy variable, and must be a vector. The first argument would normally be some function of the dummy variable, but it is not necessary that the dummy variable appear in the first argument.

The `SUM` function accepts a scalar, a vector, or a matrix as its first argument. The `SUM` function is equivalent to $\sum_{i\in\mathcal{R}} \mathcal{F}$ where i is the dummy variable, $\mathcal{R}$ is the range of the dummy variable, and $\mathcal{F}$ is some function of the dummy variable.

If $\mathcal{F}$ is a scalar, the result is a scalar. If $\mathcal{F}$ is a vector, the result is a vector. If $\mathcal{F}$ is a matrix, the result is a matrix.

Examples

Suppose X=[1;2;3;4;5], Y=[2;3;4;5;6], M= $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$ and I and J have been declared to be dummy variables with SCALAR\DUMMY I J.

function	*result*
SUM(X[I],I,1:5)	15
SUM(X,I,1:5)	[5;10;15;20;25]
SUM((X^2+Y^2)[I],I,1:5)	145
SUM(SUM(M[I,J],I,1:3),J,1:4)	78

PROD

Syntax

```
scalar = PROD(scalar,dummy,scalar)
vector = PROD(vector,dummy,vector)
matrix = PROD(matrix,dummy,matrix)
```

The PROD function requires a previously declared scalar dummy variable as second argument. A dummy variable is declared with the SCALAR\DUMMY command. The third argument is always the range of this dummy variable, and must be a vector. The first argument would normally be some function of the dummy variable, but it is not necessary that the dummy variable appear in the first argument.

The PROD function accepts a scalar, a vector, or a matrix as its first argument. The PROD function is equivalent to $\prod_{i\in\mathcal{R}}\mathcal{F}$ where i is the dummy variable, $\mathcal{R}$ is the range of the dummy variable, and $\mathcal{F}$ is some function of the dummy variable.

If $\mathcal{F}$ is a scalar, the result is a scalar. If $\mathcal{F}$ is a vector, the result is a vector. If $\mathcal{F}$ is a matrix, the result is a matrix.

Examples

Suppose X=[1;2;3;4;5], Y=[2;3;4;5;6], M= $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$ and I and J have been declared to be dummy variables with SCALAR\DUMMY I J.

function	*result*
PROD(X[I],I,1:5)	120
PROD(X,I,1:5)	[1;32;243;1024;3125]
PROD((X^2+Y^2)[I],I,1:5)	4064125
PROD(PROD(M[I,J],I,1:3),J,1:4)	479001600

RSUM

Syntax

```
vector = RSUM(scalar,dummy,vector)
matrix = RSUM(vector,dummy,matrix)
```

The RSUM function requires a previously declared scalar dummy variable as second argument. A dummy variable is declared with the SCALAR\DUMMY command. The third argument is always the range of this dummy variable, and must be a vector. The first argument would normally be some function of the dummy variable, but it is not necessary that the dummy variable appear in the first argument.

The RSUM function accepts either a scalar or a vector as its first argument. The RSUM function calculates the running sums. If the first argument is a scalar, the last element of the RSUM output vector is the total sum. If the first argument is a vector, the last column of the RSUM output matrix is the total sum.

Examples

Suppose X=[1;2;3;4;5], Y=[2;3;4;5;6], M= $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$ and I and J have been declared to be dummy variables with SCALAR\DUMMY I J.

function	*result*
RSUM(X[I],I,1:5)	[1;3;6;10;15]
RSUM(X,I,1:3)	$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \\ 4 & 8 & 12 \\ 5 & 10 & 15 \end{pmatrix}$
RSUM((X^2+Y^2)[I],I,1:5)	[5;18;43;84;145]
RSUM(RSUM(M[I,J],I,1:3),J,1:4)	$\begin{pmatrix} 1 & 3 & 6 & 10 \\ 6 & 14 & 24 & 36 \\ 15 & 33 & 54 & 78 \end{pmatrix}$

RPROD

Syntax
```
vector = RPROD(scalar,dummy,vector)
matrix = RPROD(vector,dummy,matrix)
```

The `RPROD` function requires a previously declared scalar dummy variable as second argument. A dummy variable is declared with the `SCALAR\DUMMY` command. The third argument is always the range of this dummy variable, and must be a vector. The first argument would normally be some function of the dummy variable, but it is not necessary that the dummy variable appear in the first argument.

The `RPROD` function accepts either a scalar or a vector as its first argument. The `RPROD` function calculates the running products. If the first argument is a scalar, the last element of the `RPROD` output vector is the total product. If the first argument is a vector, the last column of the `RPROD` output matrix is the total product.

Examples

Suppose `X=[1;2;3;4;5]`, `Y=[2;3;4;5;6]`, `M=` $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$ and `I` and `J` have been declared to be dummy variables with `SCALAR\DUMMY I J`.

function	*result*
`RPROD(X[I],I,1:5)`	`[1;2;6;24;120]`
`RPROD(X,I,1:3)`	$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 4 & 8 \\ 3 & 9 & 27 \\ 4 & 16 & 64 \\ 5 & 25 & 125 \end{pmatrix}$
`RPROD((X^2+Y^2)[I],I,1:5)`	`[5;65;1625;66625;4064125]`
`RPROD(RPROD(M[I,J],I,1:3),J,1:4)`	$\begin{pmatrix} 1 & 2 & 6 & 24 \\ 5 & 60 & 1260 & 40320 \\ 45 & 5400 & 1247400 & 479001600 \end{pmatrix}$

LOOP

Syntax
```
vector = LOOP(scalar,dummy,vector)
matrix = LOOP(vector,dummy,matrix)
```

The `LOOP` function requires a previously declared scalar dummy variable as second argument. A dummy variable is declared with the `SCALAR\DUMMY` command. The third argument

is always the range of this dummy variable, and must be a vector. The first argument would normally be some function of the dummy variable, but it is not necessary that the dummy variable appear in the first argument.

The `LOOP` function accepts either a scalar or a vector as its first argument. The `LOOP` function simply loops over the range of the dummy variable, filling the output with the appropriate value from the first argument. The output of the `LOOP` function will always have one dimension more than the first argument. If the first argument is a scalar, the output will be the elements of a vector. If the first argument is a vector, the output will be the columns of a matrix.

Examples

Suppose `X=[1;2;3;4;5]`, `Y=[2;3;4;5;6]`, `M=` $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}$ and `I` and `J` have been declared to be dummy variables with `SCALAR\DUMMY I J`.

function	*result*
`LOOP(X[I]*Y[(6-I)],I,1:5)`	`[6;10;12;12;10]`
`LOOP(X,I,1:3)`	$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \\ 5 & 5 & 5 \end{pmatrix}$
`LOOP(M[I,I],I,1:4)`	`[1;6;11;16]`
`LOOP(LOOP(M[I,J],I,1:J),J,1:4)`	$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 6 & 7 & 8 \\ 0 & 0 & 11 & 12 \\ 0 & 0 & 0 & 16 \end{pmatrix}$
`LOOP(LOOP(M[J,I],I,1:J),J,1:4)`	$\begin{pmatrix} 1 & 5 & 9 & 13 \\ 0 & 6 & 10 & 14 \\ 0 & 0 & 11 & 15 \\ 0 & 0 & 0 & 16 \end{pmatrix}$

Suppose `M=` $\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{pmatrix}$ and `J` has been declared to be a dummy variable with `SCALAR\DUMMY J`.
To invert the matrix, that is, to flip the matrix upside down, use:

function	*result*
`<-LOOP(M[I,*],I,3:1:-1)`	$\begin{pmatrix} 31 & 32 & 33 \\ 21 & 22 & 23 \\ 11 & 12 & 13 \end{pmatrix}$

You must use the transpose operator, since `M[I,*]` is a vector, and each time `LOOP` iterates on `I`, it produces a column of the output matrix.

GPLOT KEYWORDS

This chapter contains detailed descriptions of the GPLOT graph and text plot characteristic keywords, which are controlled by the `SET` and `GET` commands.

The values associated with every name are converted internally to floating point real, and all angles are in degrees.

A.1 Summary

Following is a list of plot characteristic keywords, with very terse descriptions and default values.

General

name	*description*	*default value*	
`PTYPE`	controls whether pixels are turned on, off, or complemented	0	
`LINTYP`	line type	1	
`LINTHK`	line thickness	1	
`COLOUR`	colour	1	
`NUMBLD`	fill number for the axis numbers	0	
`CLIP`	controls whether data curves are clipped at the box edge	1	
`HISTYP`	histogram type	0	
`CHARA`	plotting symbol angle	horizontal	
`CHARSZ`	plotting symbol size	1	%

Text

name	*description*	*default value*	
`CURSOR`	text justification	1	
`TXTANG`	text angle	0	
`TXTHIT`	text height	3	%
`XLOC`	horizontal reference location for text positioning	50	%
`YLOC`	vertical reference location for text positioning	50	%

x-axis

name	*description*	*default value*	
XAXIS	controls whether or not to draw the x-axis	1	
XLABSZ	height of the x-axis text label	3	%
XLOG	base of the x-axis numbers	0	
NXGRID	number of grid lines to draw parallel to the y-axis	0	
XCROSS	controls where the y-axis will cross the x-axis	0	
XZERO	controls whether zero is forced to appear on the x-axis	0	
XTICTP	type of tic marks to place on the x-axis	1	
XTICA	angle of the x-axis tic marks	$\parallel$ y-axis	
NLXINC	number of long x-axis tic marks	2	
XTICL	length of the long tic marks on the x-axis	2	%
NSXINC	number of short x-axis tic marks	1	
XTICS	length of the short tic marks on the x-axis	1	%
XMAX	maximum value for the x-axis	10	
XVMAX	virtual maximum for the x-axis	10	
XMIN	minimum value for the x-axis	0	
XVMIN	virtual minimum for the x-axis	0	
XMOD	base of the modulus for x-axis numbering	0	
XOFF	offset added to the numbers labeling the x-axis	0	
XLEADZ	controls whether x-axis leading zeros are displayed	0	
XPAUTO	controls the automatic x-axis scale factor	1	
XPOW	x-axis numbers scale factor	0	
NXDIG	number of digits to display in the x-axis numbers	5	
NXDEC	number of decimal places to display in the x-axis numbers	-1	
XNUMSZ	height of the numbers labeling the x-axis	3	%
XNUMA	angle of the numbers labeling the x-axis	horizontal	
XITICA	angle at which to position the x-axis numbers	$\parallel$ y-axis	
XITICL	distance from the x-axis to the numbers labeling the x-axis	3	%

y-axis

name	*description*	*default value*	
YAXIS	controls whether or not to draw the y-axis	1	
YLABSZ	height of the y-axis text label	3	%
YLOG	controls whether the y-axis is to be linear or logarithmic	0	
NYGRID	number of grid lines to draw parallel to the x-axis	0	
YCROSS	controls where the x-axis will cross the y-axis	0	
YZERO	controls whether zero is forced to appear on the y-axis	0	
YTICTP	type of tic marks to place on the y-axis	1	
YTICA	angle of the y-axis tic marks	$\parallel$ x-axis	
NLYINC	number of long y-axis tic marks	2	
YTICL	length of the long tic marks on the y-axis	2	%
NSYINC	number of short y-axis tic marks	1	
YTICS	length of the short tic marks on the y-axis	1	%
YMAX	maximum value for the y-axis	10	
YVMAX	virtual maximum for the y-axis	10	
YMIN	minimum value for the y-axis	0	
YVMIN	virtual minimum value for the y-axis	0	
YMOD	base of the modulus for the y-axis numbering	0	
YOFF	offset added to the numbers labeling the y-axis	0	
YLEADZ	controls whether y-axis leading zeros are displayed	0	
YPAUTO	controls the automatic y-axis scale factor	1	
YPOW	y-axis numbers scale factor	0	
NYDIG	number of digits to display in the y-axis numbers	5	
NYDEC	number of decimal places to display in the y-axis numbers	-1	
YNUMSZ	height of the numbers labeling the y-axis	3	%
YNUMA	angle of the numbers labeling the y-axis	horizontal	
YITICA	angle at which to position the y-axis numbers	$\parallel$ y-axis	
YITICL	distance from the y-axis to the numbers labeling the y-axis	3	%

Axis Box

name	*description*	*default value*	
XLWIND	the left edge of the window	0	%
XUWIND	the right edge of the window	100	%
YLWIND	the bottom edge of the window	0	%
YUWIND	the top edge of the window	100	%
BOX	controls whether or not to draw an axis box around the graph	1	
XLAXIS	location of the lower end of the x-axis	15	%
XUAXIS	location of the upper end of the x-axis	95	%
XAXISA	angle of the x-axis	0	
YLAXIS	location of the bottom of the y-axis	15	%
YUAXIS	location of the top of the y-axis	90	%
YAXISA	angle of the y-axis	90	
BOTNUM	controls the height of the numbers on the bottom of the box	0	
BOTTIC	controls the length of the tic marks on the bottom of the box	1	
RITNUM	controls the height of the numbers on the right side of the box	0	
RITTIC	controls the length of the tic marks on the right side of the box	-1	
TOPNUM	controls the height of the numbers on the top of the box	0	
TOPTIC	controls the length of the tic marks on the top of the box	-1	
LEFNUM	controls the height of the numbers on the left side of the box	0	
LEFTIC	length of the tic marks on the left side of the box	1	

A.2 General Characteristics

PTYPE

Default: PTYPE $= 0$

PTYPE controls whether graphics pixels are turned on, off, or toggled. This can be used to selectively erase graphics, by drawing with PTYPE $= 0$ and redrawing with PTYPE $= 1$; or by drawing and redrawing with PTYPE $= 2$. This only works on the monitor screen and on bitmap hardcopy output. Values other than 0, 1, 2 are ignored.

0	pixels turned on (draw)
1	pixels turned off (erase)
2	pixels complemented

Complemented pixels means that a pixel is turned off if it is on, or turned off if it is on.

LINTYP

Default: LINTYP $= 1$

LINTYP controls the type of line to use when drawing a data curve. LINTYP should be between 1 and 10, inclusive. Table 2.8 on page 30 shows the specifications for the default line types. The length of the dashes in the dashed line types, the default types 3 to 10, are constant and do not depend on the separation between data points. The definition of a line type can be changed via the SET LINE command.

LINTHK

Default: LINTHK $= 1$

LINTHK controls the line thickness for bitmap hardcopy output and for PostScript hardcopy output. LINTHK has no affect on monitor screen output or on pen plotter hardcopy output.

COLOUR

Default: COLOUR $= 1$

COLOUR sets a colour code which is used to control the monitor screen colour and the hardcopy colour. See Table 2.6 on page 20 for a list of the colour names and associated colour numbers.

NUMBLD

Default: NUMBLD $= 0$

NUMBLD is the fill number for the axis numbers.

0	no filling
$1 \leq$ NUMBLD ≤ 10	use hatch pattern
$11 \leq$ NUMBLD ≤ 99	use dot fill pattern

CLIP

Default: CLIP $= 1$

CLIP controls whether or not data curves that are plotted are clipped at the boundaries of the axis box.

0	do not clip
$\neq 0$	clip at the boundaries of the axis box

HISTYP

Default: HISTYP $= 0$

HISTYP controls whether a normal line graph or a histogram is drawn. Histograms can have tails or no tails and the profile can be along the x-axis or along the y-axis. See Figure 2.16 on page 123.

0	normal line graph, not a histogram
1	histogram with no tails and profile along the x-axis. Can control the width and colour of each individual bar
2	histogram with tails to $y = 0$ and profile along the x-axis. Can control the filling pattern, width and colour of each individual bar
3	histogram without tails and profile along the y-axis. Can control the height and colour of each individual bar
4	histogram with tails to $x = 0$ and profile along the y-axis. Can control the filling pattern, height and colour of each individual bar

No plotting symbol will be drawn at the data points when HISTYP > 0. The fill pattern, width and colour of each histogram bar can be controlled by using the optional arrays in the SET PCHAR command.

CHARA

Default: %CHARA = XAXISA

CHARA is the angle, in degrees, of the plotting symbols that are drawn at the data points, measured counterclockwise between a horizontal line and a base line through the plotting symbol. CHARA will not be used when plotting if the optional angle array is included with the SET PCHAR command.

If CHARA is set as a percentage, the actual value to which %CHARA has been set is ignored and the angle is set to XAXISA. This allows the user to change the angle of the x-axis and keep the plotting symbols base line parallel to the x-axis. By default, CHARA is set as a percentage.

CHARSZ

Default: %CHARSZ $= 1$

CHARSZ is the size of the plotting symbols that are drawn at the data points. %CHARSZ is a percentage of the height of the window, that is, CHARSZ = %CHARSZ $\times$ (YUWIND $-$ YLWIND) $\div 100$ CHARSZ will be the base size of the plotting symbols if the optional size array is included in the SET PCHAR command.

A.3 Text

CURSOR

Default: CURSOR $= 1$

CURSOR controls the justification of the text that is drawn when using TEXT command. The origin of the text is always the lower left corner of the string. The justification determines where this origin is placed with respect to a reference point. Refer to Figure 2.28 on page 258 and to Table 2.64 on page 259.

≤ 0	then XLOC and YLOC will be used to determine the reference point for the text, and the justification will be determined by |CURSOR|.
> 0	the user selects a reference point with the graphics cursor. The justification of the text is selected interactively. Refer to Table 2.65 on page 260.

The value of CURSOR will be updated to the value corresponding to the justification that was chosen. The values of %XLOC and %YLOC will be updated to the new reference point location.

 GPLOT Keywords

TXTANG

Default: $\texttt{TXTANG} = 0$

TXTANG controls the angle at which text will be drawn, when using the TEXT command. TXTANG is the angle, in degrees, measured counterclockwise, between the base line of the text and a horizontal line. The value of TXTANG will be ignored if $|\texttt{CURSOR}| = 4,\ 5,$ or 6.

TXTHIT

Default: $\texttt{\%TXTHIT} = 3$

TXTHIT is the height of text to be drawn, when using the TEXT command. %TXTHIT is a percentage of the height of the window, that is, $\texttt{TXTHIT} = \texttt{\%TXTHIT} \times (\texttt{YUWIND} - \texttt{YLWIND}) \div 100$

XLOC

Default: $\texttt{\%XLOC} = 50$

XLOC is the horizontal reference position of a text drawn using the TEXT command. This reference point is used for text justification. See the explanation of CURSOR for how XLOC will be used. XLOC will be used if $\texttt{CURSOR} < 0$, or if the X key is typed when drawing text in interactive mode. The value of %XLOC is updated after each string is drawn with the TEXT command. %XLOC is a percentage of the width of the window, that is, $\texttt{XLOC} = \texttt{XLWIND} + \texttt{\%XLOC} \times (\texttt{XUWIND} - \texttt{XLWIND}) \div 100$

YLOC

Default: $\texttt{\%YLOC} = 50$

YLOC controls the vertical reference position of a text string drawn with the TEXT command. This reference point is used for text justification. See the explanation of CURSOR for how YLOC will be used. YLOC will be used if $\texttt{CURSOR} < 0$, or if the Y key is typed when drawing text in interactive mode. The value of %YLOC is updated after each string is drawn with the TEXT command. %YLOC is a percentage of the height of the window, that is, $\texttt{YLOC} = \texttt{YLWIND} + \texttt{\%YLOC} \times (\texttt{YUWIND} - \texttt{YLWIND}) \div 100$

A.4 x-axis

Figure A.36 illustrates the definitions of some of the x-axis characteristics.

XAXIS

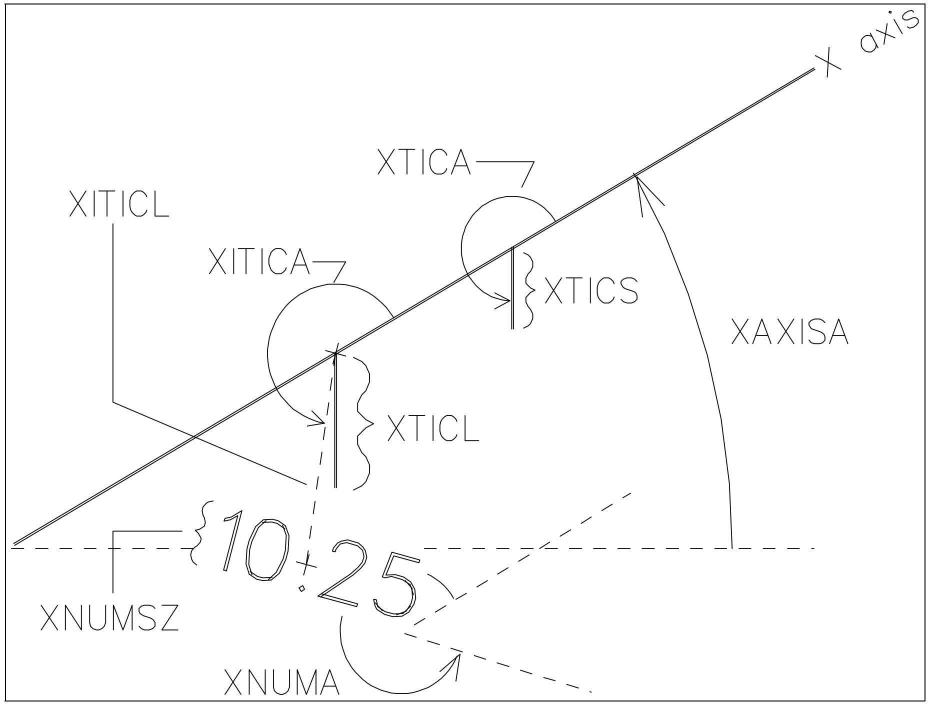


Figure A.36: Some x-axis characteristics

Default: XAXIS $= 1$

XAXIS controls whether or not the x-axis is drawn.

0	do not draw the x-axis
$\neq 0$	draw the x-axis

If BOX $\neq 0$, then the bottom of the axis box will be drawn, even if XAXIS $= 0$.

XLABSZ

Default: %XLABSZ $= 3$

XLABSZ controls the height of the automatic text label for the x-axis, set by the LABEL\XAXIS command. %XLABSZ is a percentage of the height of the window, that is, XLABSZ $=$ %XLABSZ $\times$ (YUWIND $-$ YLWIND) $\div 100$

XLOG

Default: XLOG $= 0$

XLOG determines whether the x-axis is to be linear or logarithmic. See Figure A.37 for examples of various logarithmic axes.

> 1	the x-axis will have a logarithmic scale. The numbers labeling the x-axis will be in exponent form, for example, $10^1\ 10^2\ 10^3\ 10^4$.
$-1 \leq$ XLOG ≤ 1	the x-axis will have a linear scale
< -1	the x-axis will have a logarithmic scale. The numbers labeling the x-axis will be in full digit form, for example, $10\ \ 100\ \ 1000\ \ 10000$.

Suppose that $|$XLOG$| > 1$. The base will be the *integer part* of $|$XLOG$|$, except for the special case: $1.05 \times e >$ XLOG $> 0.95 \times e$, where e is the base of the natural logarithms, $e \approx 2.718281828$, in which case the base will be e. XMIN and XMAX will be the *exponents* for the minimum and maximum values on the x-axis. The maximum value displayed on the x-axis is $|\mathtt{XLOG}|^{\mathtt{XMAX}}$ and the minimum value displayed on the x-axis is $|\mathtt{XLOG}|^{\mathtt{XMIN}}$. Integer exponents are always used for the axis numbering, so the x-axis may not begin and/or end on a large tic mark.

If $|$XLOG$| = 10$, the small tic marks on the x-axis can be specified exactly with NSXINC. If small tic marks are desired at the locations $j_m \times 10^n$, where $2 \leq j_m \leq 9$, then set NSXINC $= -j_1 \ldots j_m$. For example, if you want small tic marks at 2×10^n, 4×10^n, 5×10^n, and 8×10^n, then set NSXINC $= -2458$.

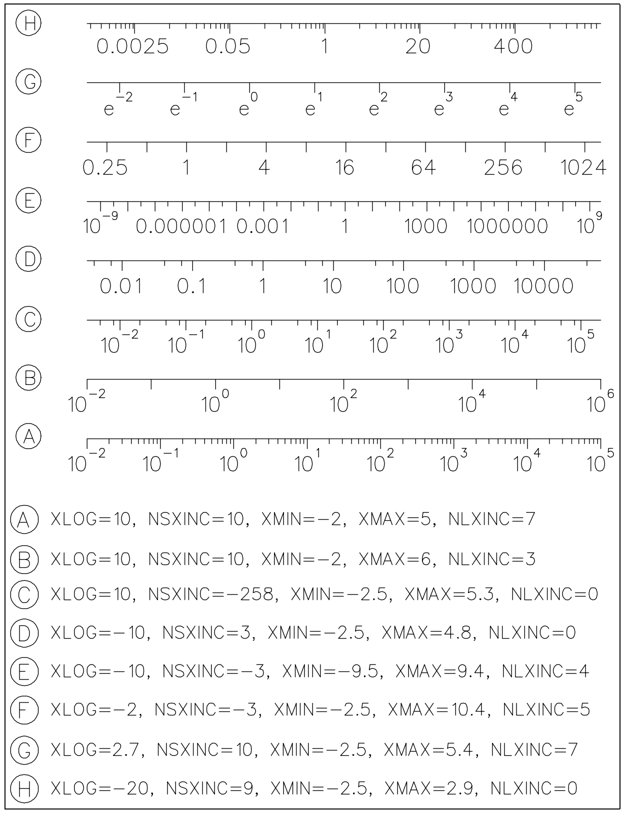


Figure A.37: Logarithmic x-axis examples

NXGRID

Default: NXGRID $= 0$

NXGRID controls the number of grid lines parallel to the y-axis. NXGRID specifies the number of large x-axis tic marks between each grid line.

> 0	grid lines parallel to the y-axis at every NXGRID large tic mark, starting with the first
0	suppress all grid lines parallel to the y-axis
-1	grid lines parallel to the y-axis at every x-axis tic mark, large and small. This option applies only if the axes are orthogonal.

XCROSS

Default: XCROSS $= 0$

XCROSS controls where the y-axis is to cross the x-axis. The axes always cross at a large tic mark.

0	the y-axis will cross the x-axis at XMIN
$\neq 0$	the y-axis will cross the x-axis at the large tic mark closest to $x = 0$

XZERO

Default: XZERO $= 0$

XZERO controls whether or not to force zero to be displayed on the x-axis. XZERO is set to zero after each graph is drawn.

$\neq 0$	if XMIN ≥ 0 then set XMIN to zero, or, if XMAX ≤ 0 then set XMAX to zero
0	do not force zero to be displayed on the x-axis

XTICTP

Default: XTICTP $= 1$

XTICTP controls the type of tic marks to place on the x-axis.

2	tic marks on both sides of the x-axis
$\neq 2$	tic marks on only one side of the x-axis. XTICA controls the side on which the tic marks will appear

XTICA

Default: XTICA $=$ YAXISA $-$ XAXISA $+ 180°$

XTICA is the angle, in degrees, measured counterclockwise, between a horizontal line and a tic mark, both large and small, on the x-axis. See Figure A.36. By default, XTICA is set as a percentage. If XTICA is set as a percentage, then XTICA is set to YAXISA $-$ XAXISA $+ 180$. The actual value of %XTICA will be ignored. This allows the user to change the angle of the axes and keep the x-axis tic marks parallel to the y-axis.

NLXINC

Default: NLXINC $= 2$

NLXINC controls the number of large tic marks to be displayed on the x-axis.

-1000	drop the first and the last numbers on the x-axis and determine the number of large x-axis tic marks
< 0	drop the first and the last numbers on the x-axis. The number of large tic marks on the x-axis will be $\lvert$NLXINC$\rvert + 1$, unless $\lvert$XLOG$\rvert > 1$, in which case the number of tic marks will be determined to avoid fractional powers
0	the number of large tic marks on the x-axis will be automatically determined
> 0	the number of large tic marks on the x-axis will be NLXINC $+ 1$, unless $\lvert$XLOG$\rvert > 1$, in which case NLXINC will be determined to avoid fractional powers

XTICL

Default: %XTICL $= 2$

XTICL is the length of the long tic marks on the x-axis. See Figure A.36 on page 356. %XTICL is a percentage of the height of the window, that is, XTICL $=$ %XTICL $\times$ (YUWIND $-$ YLWIND) $\div 100$

NSXINC

Default: NSXINC $= 1$

NSXINC controls the number of small tic marks to be displayed on the x-axis. The small tic marks appear between the large tic marks.

≤ 1	no small tic marks on the x-axis
≥ 2	the number of small tic marks, on the x-axis, between each pair of large tic marks, will be NSXINC $- 1$

If $|\texttt{XLOG}| = 10$, the small tic marks on the x-axis can be specified exactly with NSXINC. If small tic marks are desired at the locations $j_m \times 10^n$, where $2 \leq j_m \leq 9$, then set NSXINC $= -j_1 \ldots j_m$. For example, if you want small tic marks at 2×10^n, 4×10^n, 5×10^n, and 8×10^n, then set NSXINC $= -2458$.

XTICS

Default: %XTICS $= 1$

XTICS controls the length of the short tic marks on the x-axis. See Figure A.36 on page 356. %XTICS is a percentage of the height of the window, that is, XTICS $=$ %XTICS $\times$ (YUWIND $-$ YLWIND) $\div 100$

XMAX

Default: XMAX $= 10$

XMAX controls the maximum value for the x-axis.

If $|\texttt{XLOG}| > 1$, then XMAX is assumed to be the exponent for the maximum value on the x-axis. The maximum value displayed on the x-axis is $|\texttt{XLOG}|^{\texttt{XMAX}}$. Integer exponents are always used for the axis numbering, so the x-axis may not end on a large tic mark.

If $|\texttt{XLOG}| \leq 1$, then XMAX is the actual maximum value for the x-axis. If XVMAX is equal to XMAX then the x-axis always ends on a large, labeled, tic mark. If XVMAX is not equal to XMAX then the x-axis will not end at a large, labeled, tic mark.

When the value of XMAX is changed, the value of XVMAX is simultaneously changed to the same value.

XVMAX

Default: XVMAX $= 10$

XVMAX controls the virtual maximum value for the x-axis. See Figure A.38 on page 363.

If $|\texttt{XLOG}| > 1$, the virtual maximum is ignored.

If $|\texttt{XLOG}| \leq 1$, then XVMAX is the virtual maximum value of the labels for the x-axis. This value will not be displayed if XVMAX is greater than XMAX, but it will be displayed if XVMAX is less than or equal to XMAX. This virtual maximum is used to determine "nice" numbers for labeling the large tic marks on the x-axis. If XVMAX is not equal to XMAX then the x-axis will not end on a large, labeled, tic mark.

The value of XVMAX is changed to the value of XMAX when the value of XMAX is changed. So, if you want to make XVMAX different from XMAX, it must be changed *after* XMAX is changed.

If NLXINC is set to zero, then the x-axis will begin at XMIN and end at XMAX, but if these are not 'nice' numbers, the virtual minimum and maximum will be set to values outside the actual minimum and maximum so that the large tic marks can be labeled with "nice" numbers.

XMIN

Default: XMIN $= 0$

XMIN controls the minimum value for the x-axis.

If $|\texttt{XLOG}| > 1$, then XMIN is assumed to be the exponent for the minimum value on the x-axis. The minimum value displayed on the x-axis is $|\texttt{XLOG}|^{\texttt{XMIN}}$. Integer exponents are always be used for the axis numbering, so the x-axis may not begin on a large tic mark.

If $|\texttt{XLOG}| \leq 1$, XMIN is the actual minimum value for the x-axis. If XVMIN is equal to XMIN then the x-axis always begins on a large, labeled, tic mark. If XVMIN is not equal to XMIN then the x-axis will not begin at a large, labeled, tic mark.

When the value of XMIN is changed, the value of XVMIN is simultaneously changed to the same value.

XVMIN

Default: XVMIN $= 0$

XVMIN controls the virtual minimum value for the x-axis. See Figure A.38 on page 363.

If $|\texttt{XLOG}| > 1$, the virtual minimum is ignored.

If $|\texttt{XLOG}| \leq 1$, then XVMIN is the virtual minimum value of the labels for the x-axis. This value will not be displayed if XVMIN is less than XMIN, but it will be displayed if XVMIN is greater than or equal to XMIN. This virtual minimum is used to determine "nice" numbers for labeling the large tic marks on the x-axis. If XVMIN is not equal to XMIN then the x-axis will not begin on a large, labeled, tic mark.

The value of XVMIN is changed to the value of XMIN when the value of XMIN is changed. So, if you want to make XVMIN different from XMIN, it must be changed *after* XMIN is changed.

If NLXINC is set to zero, then the x-axis will begin at XMIN and end at XMAX, but if these are not "nice" numbers, the virtual minimum and maximum will be set to values outside the actual minimum and maximum so that the large tic marks can be labeled with "nice" numbers.

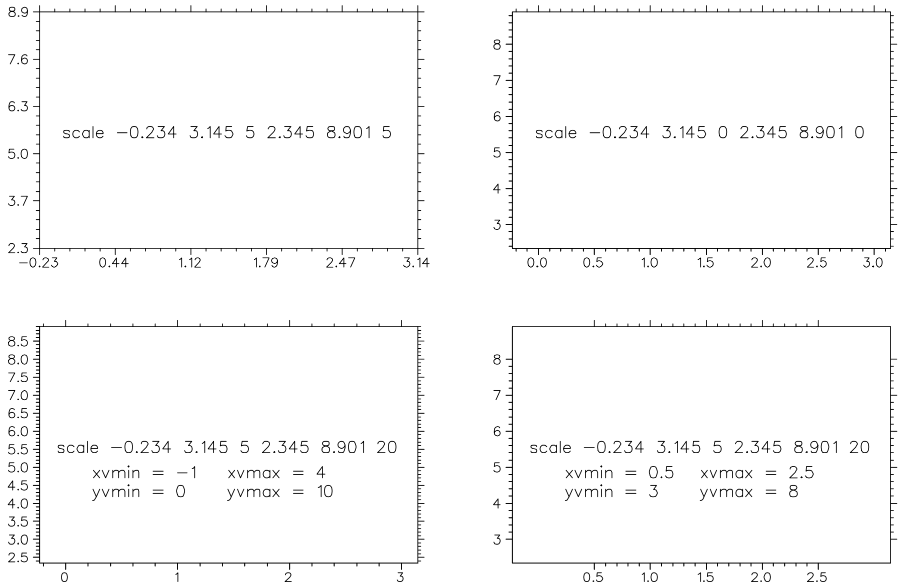


Figure A.38: Virtual axes examples

XMOD

Default: XMOD $= 0$

XMOD is another control on the numbering for the x-axis.

GPLOT Keywords

> 1	the numbers displayed on the x-axis are modulo XMOD and positive. If the number to be displayed on the axis is x then the number that is actually displayed is $\lvert x - \texttt{XMOD} \times (x/\texttt{XMOD}) \rvert + \texttt{XOFF}$
$-1 \le$ XMOD ≤ 1	the numbers will be displayed normally
< -1	the numbers will be modulo $\lvert\texttt{XMOD}\rvert$ and may be negative

XOFF

Default: XOFF $= 0$

XOFF is another control on the numbering for the x-axis. If $\lvert\texttt{XMOD}\rvert > 1$, then XOFF is added to the numbers to be displayed on the x-axis. If $\lvert\texttt{XMOD}\rvert \le 1$, then XOFF is ignored.

XLEADZ

Default: XLEADZ $= 0$

XLEADZ controls whether leading zeros are displayed on the x-axis numbers.

1	numbers will have leading zeros if they are between 0 and 1
0	numbers will *not* have leading zeros if they are between 0 and 1

The default is to not display these leading zeros. Numbers that are between -1 and 0 always have the leading zero displayed.

XPAUTO

Default value: XPAUTO $= 1$

XPAUTO, along with XPOW, controls the x-axis scale factor that is appended to the x-axis text label. This scale factor is needed when there are not enough digits allowed for the numbers labeling the x-axis. XPOW is only appended to the text label if XPOW $\neq 0$.

2	determine XPOW, but do *not* append the scale factor to the text label
1	determine XPOW, and append the scale factor to the text label
0	use the present value of XPOW

If the user wishes to completely specify the appearance of the x-axis, XPAUTO must be set to 0, otherwise the number of digits and decimal places, NXDIG and NXDEC, may be changed.

XPOW

Default: XPOW $= 0$

XPOW controls the x-axis scale factor that is appended to the x-axis text label. This scale factor is a power of ten, that is, $10^{\texttt{XPOW}}$, and the numbers labeling the x-axis should be multiplied by this scale factor to get the correct graph units. XPOW is only appended to the text label if XPOW $\neq 0$.

NXDIG

Default: NXDIG $= 5$

NXDIG controls the total number of digits to be displayed in each of the numbers labeling the x-axis. If NXDIG is smaller than required to display the x-axis numbers, NXDIG will *not* be increased, but a scale factor, $(\times 10^n)$, will be appended to the x-axis text label. If NXDIG is larger than required, NXDIG will be reduced to the minimum value required. The value of NXDIG is updated after each graph is drawn.

NXDEC

Default: NXDEC $= -1$

NXDEC controls the number of digits to be displayed in the fractional parts of the numbers labeling the x-axis. The value of NXDEC is updated after each graph is drawn.

-1	display the numbers labeling the x-axis as integers, with no decimal point
0	display the numbers labeling the x-axis with no fractional part, but with a decimal point
> 0	display the numbers labeling the x-axis with NXDEC digits in the fractional part

XNUMSZ

Default: %XNUMSZ $= 3$

XNUMSZ controls the size of the numbers labeling the x-axis. See Figure A.36. %XNUMSZ is a percentage of the height of the window, that is, XNUMSZ $=$ %XNUMSZ $\times$ (YUWIND $-$ YLWIND) $\div 100$

XNUMA

Default: XNUMA $= -$XAXISA

XNUMA controls the angle of the base line for the numbers labeling the x-axis. XNUMA is the angle, in degrees, measured counterclockwise, between a line parallel to the x-axis and the base line of a number. Refer to Figure A.36 on page 356. By default, XNUMA is set as a percentage. If XNUMA is set as a percentage, then XNUMA is set to $-$XAXISA. The actual value of %XNUMA will be ignored. This allows the user to change the angle of the x-axis and keep the base line of the x-axis numbers horizontal.

XITICA

Default: XITICA $=$ YAXISA $-$ XAXISA $+ 180$

XITICA, along with XITICL, controls the location of the numbers labeling the x-axis at the large tic marks. XITICA is the angle, in degrees, measured counterclockwise, between the x-axis and a line joining the base of each large tic mark on the x-axis to the centre of the number labeling that tic mark. See Figure A.36. By default, XITICA is set as a percentage. If XITICA is set as a percentage, then XITICA is set to YAXISA $-$ XAXISA $+ 180$. The actual value of %XITICA will be ignored. This allows the user to change the angle of the axes and keep the relative locations of the x-axis numbers the same.

XITICL

Default: %XITICL $= 3$

XITICL, along with XITICA, controls the location of the numbers labeling the x-axis at the large tic marks. XITICL is the distance from the base of each large tic mark on the x-axis, to the centre of the number labeling that tic mark. See Figure A.36 on page 356. %XITICL is a percentage of the height of the window, that is, XITICL $=$ %XITICL $\times$ (YUWIND $-$ YLWIND) $\div 100$

A.5 y-axis

Figure A.39 illustrates the definitions of some of the y-axis characteristics.

YAXIS

Default: YAXIS $= 1$

YAXIS controls whether or not the y-axis is drawn.

$\neq 0$	draw the y-axis
0	do not draw the y-axis

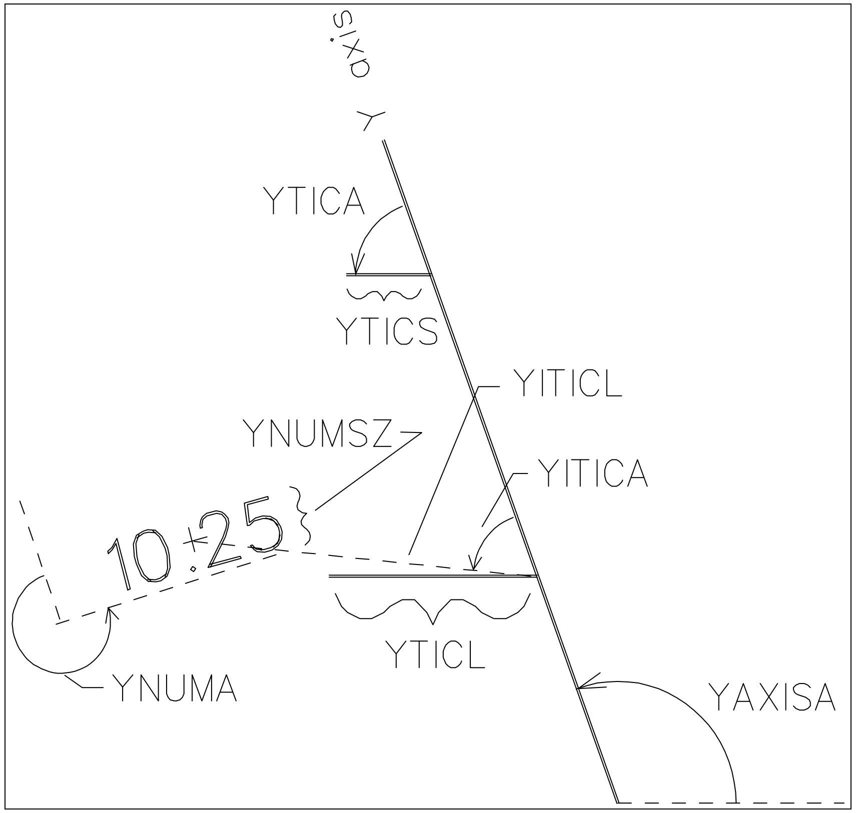


Figure A.39: Some y-axis characteristics

If BOX $\neq 0$, then the left side of the axis box will be drawn, even if YAXIS $= 0$.

YLABSZ

Default: %YLABSZ $= 3$

YLABSZ controls the size of the automatic text label for the y-axis, set by the LABEL\YAXIS command. %YLABSZ is a percentage of the height of the window, that is, YLABSZ $=$ %YLABSZ $\times$ (YUWIND $-$ YLWIND) $\div 100$

YLOG

Default: YLOG $= 0$

YLOG determines whether the y-axis is to be linear or logarithmic. See Figure A.40 for examples of various logarithmic axes.

> 1	the y-axis will have a logarithmic scale. The numbers labeling the y-axis will be in exponent form, for example, 10^1 10^2 10^3 10^4.
$-1 \leq$ YLOG ≤ 1	the y-axis will have a linear scale
< -1	the y-axis will have a logarithmic scale. The numbers labeling the y-axis will be in full digit form, for example, $10\ \ 100\ \ 1000\ \ 10000$.

Suppose that |YLOG| > 1. The base will be the *integer part* of |YLOG|, except for the special case: $1.05 \times e >$ YLOG $> 0.95 \times e$, where e is the base of the natural logarithms, $e \approx 2.718281828$, in which case the base will be e. YMIN and YMAX will be the *exponents* for the minimum and maximum values on the y-axis. The maximum value displayed on the y-axis is $|\mathtt{YLOG}|^{\mathtt{YMAX}}$ and the minimum value displayed on the y-axis is $|\mathtt{YLOG}|^{\mathtt{YMIN}}$. Integer exponents are always used for the axis numbering, so the y-axis may not begin and/or end on a large tic mark.

If |YLOG| $= 10$, the small tic marks on the y-axis can be specified exactly with NSYINC. If small tic marks are desired at the locations $j_m \times 10^n$, where $2 \leq j_m \leq 9$, then set NSYINC $= -j_1 \ldots j_m$. For example, if you want small tic marks at 2×10^n, 4×10^n, 5×10^n, and 8×10^n, then set NSYINC $= -2458$.

NYGRID

Default: NYGRID $= 0$

NYGRID controls the number of grid lines parallel to the x-axis. NYGRID specifies the number of large y-axis tic marks between each grid line.

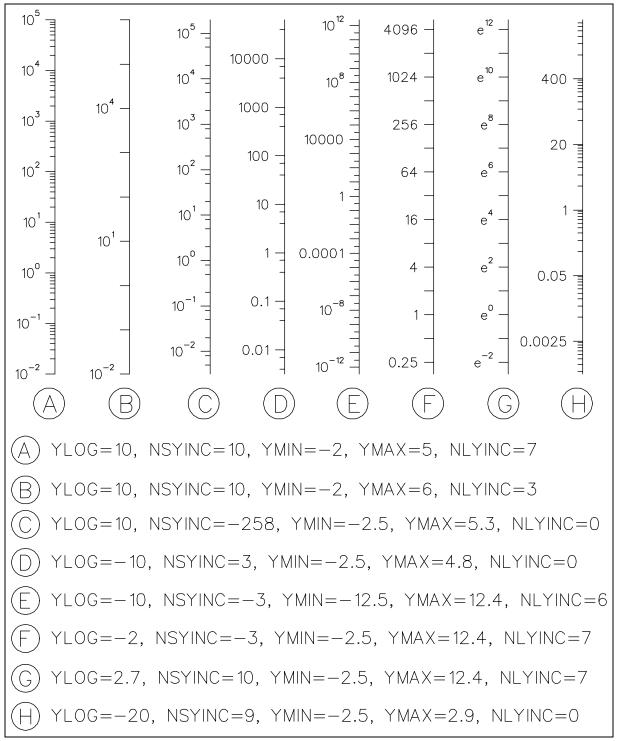


Figure A.40: Logarithmic y-axis examples

> 0	draw a grid line parallel to the x-axis at every `NYGRID` large tic mark, starting with the first
0	suppress all grid lines parallel to the x-axis
-1	draw a grid line parallel to the x-axis at every y-axis tic mark, large and small. This option applies only if the axes are orthogonal.

YCROSS

Default: `YCROSS` $= 0$

`YCROSS` controls where the x-axis is to cross the y-axis. The axes always cross at a large tic mark.

0	the x-axis will cross the y-axis at `YMIN`
$\neq 0$	the x-axis will cross the y-axis at $y = 0$, or at the large tic mark closest to $y = 0$

YZERO

Default: `YZERO` $= 0$

`YZERO` controls whether or not to force zero to be displayed on the y-axis. `YZERO` is set to zero after each graph is drawn.

$\neq 0$	if `YMIN` ≥ 0 then set `YMIN` to zero, or, if `YMAX` ≤ 0 then set `YMAX` to zero
0	do not force zero to be displayed on the y-axis

YTICTP

Default: `YTICTP` $= 1$

`YTICTP` controls the type of tic marks to place on the y-axis.

$\neq 2$	tic marks are drawn on only one side of the y-axis. `YTICA` controls the side on which the tic marks will appear
2	tic marks are drawn on both sides of the y-axis

YTICA

Default: `YTICA` $=$ `XAXISA` $-$ `YAXISA` $+ 180$

YTICA controls the angle of the tic marks, both large and small, on the y-axis. YTICA is the angle, in degrees, measured counterclockwise, between the y-axis and a tic mark. See Figure A.39 on page 367. By default, YTICA is set as a percentage. If YTICA is set as a percentage, for example, SET %YTICA 0, then YTICA is set to XAXISA $-$ YAXISA $+ 180$. The actual value of %YTICA will be ignored. This allows the user to change the angle of the axes and keep the y-axis tic marks parallel to the x-axis.

NLYINC

Default: NLYINC $= 2$

NLYINC controls the number of large tic marks to be displayed on the y-axis.

-1000	drop the first and the last numbers on the y-axis and determine the number of large y-axis tic marks
< 0	drop the first and the last numbers on the y-axis. The number of large tic marks on the y-axis will be $\vert$NLYINC$\vert + 1$, unless YLOG > 1, in which case the number of tic marks will be determined to avoid fractional powers
0	the number of large tic marks on the y-axis will be automatically determined
> 0	the number of large tic marks on the y-axis will be NLYINC$+1$, unless YLOG > 1, in which case NLYINC will be determined to avoid fractional powers

YTICL

Default: %YTICL $= 2$

YTICL is the length of the long tic marks on the y-axis. See Figure A.39 on page 367. %YTICL is a percentage of the height of the window, that is, YTICL $=$ %YTICL $\times$ (YUWIND $-$ YLWIND) $\div 100$

NSYINC

Default: NSYINC $= 1$

NSYINC controls the number of small tic marks to be displayed on the y-axis. The small tic marks appear between the large tic marks. See Figure A.39 on page 367.

≤ 1	no small tic marks on the y-axis
≥ 2	the number of small tic marks, on the y-axis, between each pair of large tic marks, will be NSYINC -1

If $\vert$YLOG$\vert = 10$, the small tic marks on the y-axis can be specified exactly with NSYINC. If small tic marks are desired at the locations $j_m \times 10^n$, where $2 \leq j_m \leq 9$, then set NSYINC $= -j_1 \ldots j_m$. For example, if you want small tic marks at 2×10^n, 4×10^n, 5×10^n, and 8×10^n, then set NSYINC $= -2458$.

YTICS

Default: %YTICS $= 1$

YTICS is the length of the short tic marks on the y-axis. See Figure A.39 on page 367. %YTICS is a percentage of the height of the window, that is, YTICS $=$ %YTICS $\times$ (YUWIND $-$ YLWIND) $\div 100$

YMAX

Default: YMAX $= 10$

YMAX controls the maximum value for the y-axis.

If $\vert$YLOG$\vert > 1$, then YMAX is assumed to be the *exponent* for the maximum value on the y-axis. The maximum value displayed on the y-axis is $\vert\mathtt{YLOG}\vert^{\mathtt{YMAX}}$. Integer exponents are always used for the axis numbering, so the y-axis may not end on a large tic mark.

If $\vert$YLOG$\vert \leq 1$, then YMAX is the actual maximum value for the y-axis. If YVMAX is equal to YMAX then the y-axis always ends on a large, labeled, tic mark. If YVMAX is not equal to YMAX then the y-axis will not end at a large, labeled, tic mark.

When the value of YMAX is changed, the value of YVMAX is simultaneously changed to the same value.

YVMAX

Default: YVMAX $= 10$

YVMAX controls the virtual maximum value for the y-axis. See Figure A.38 on page 363.

If $\vert$YLOG$\vert > 1$, the virtual maximum is ignored.

If $\vert$YLOG$\vert \leq 1$, then YVMAX is the virtual maximum value of the labels for the y-axis. This value will not be displayed if YVMAX is greater than YMAX, but it will be displayed if YVMAX is less than or equal to YMAX. This virtual maximum is used to determine "nice" numbers for labeling the large tic marks on the y-axis. If YVMAX is not equal to YMAX then the y-axis will not end on a large, labeled, tic mark.

The value of YVMAX is changed to the value of YMAX when the value of YMAX is changed. So, if you want to make YVMAX different from YMAX, it must be changed *after* YMAX is changed.

If NLYINC is set to zero, then the y-axis will begin at YMIN and end at YMAX, but if these are not "nice" numbers, the virtual minimum and maximum will be set to values outside the actual minimum and maximum so that the large tic marks can be labeled with "nice" numbers.

YMIN

Default: $\texttt{YMIN} = 0$

YMIN controls the minimum value for the y-axis.

If $|\texttt{YLOG}| > 1$, then YMIN is assumed to be the *exponent* for the minimum value on the y-axis. The minimum value displayed on the y-axis is $|\texttt{YLOG}|^{\texttt{YMIN}}$. Integer exponents are always be used for the axis numbering, so the y-axis may not begin on a large tic mark.

If $|\texttt{YLOG}| \le 1$, YMIN is the actual minimum value for the y-axis. If YVMIN is equal to YMIN then the y-axis always begins on a large, labeled, tic mark. If YVMIN is not equal to YMIN then the y-axis will not begin at a large, labeled, tic mark.

When the value of YMIN is changed, the value of YVMIN is simultaneously changed to the same value.

YVMIN

Default: $\texttt{YVMIN} = 0$

YVMIN controls the virtual minimum value for the y-axis. See Figure A.38 on page 363.

If $|\texttt{YLOG}| > 1$, the virtual minimum is ignored.

If $|\texttt{YLOG}| \le 1$, then YVMIN is the virtual minimum value of the labels for the y-axis. This value will not be displayed if YVMIN is less than YMIN, but it will be displayed if YVMIN is greater than or equal to YMIN. This virtual minimum is used to determine "nice" numbers for labeling the large tic marks on the y-axis. If YVMIN is not equal to YMIN then the y-axis will not begin on a large, labeled, tic mark.

The value of YVMIN is changed to the value of YMIN when the value of YMIN is changed. So, if you want to make YVMIN different from YMIN, it must be changed *after* YMIN is changed.

If NLYINC is set to zero, then the y-axis will begin at YMIN and end at YMAX, but if these are not

"nice" numbers, the virtual minimum and maximum will be set to values outside the actual minimum and maximum so that the large tic marks can be labeled with "nice" numbers.

YMOD

Default: $\texttt{YMOD} = 0$

YMOD is another control on the numbering for the y-axis.

> 1	the numbers displayed on the y-axis will be modulo YMOD and positive. If the number to be displayed on the axis is x then the number that is actually displayed is $|x - \texttt{YMOD} \times (x/\texttt{YMOD})| + \texttt{YOFF}$
$-1 \le \texttt{YMOD} \le 1$	the numbers will be displayed normally
< -1	the numbers will be modulo $|\texttt{YMOD}|$ and may be negative

YOFF

Default: $\texttt{YOFF} = 0$

YOFF is another control on the numbering for the y-axis. If $|\texttt{YMOD}| > 1$, then YOFF is added to the numbers to be displayed on the y-axis. If $|\texttt{YMOD}| \le 1$, then YOFF is ignored.

YLEADZ

Default: $\texttt{YLEADZ} = 0$

YLEADZ controls whether leading zeros are displayed on the y-axis numbering.

$\neq 0$	the numbers displayed on the will have leading zeros if they are between 0 and 1
0	the numbers displayed on the will *not* have leading zeros if they are between 0 and 1

The default is to not display these leading zeros. Numbers that are between -1 and 0 always have the leading zero displayed.

YPAUTO

Default value: $\texttt{YPAUTO} = 1$

YPAUTO, along with YPOW, controls the y-axis scale factor that is appended to the y-axis text label. This scale factor is needed when there are not enough digits allowed for the numbers labeling the y-axis.

- 2 determine `YPOW`, but do not append the scale factor to the text label even if it is needed
- 1 determine `YPOW`, and append the scale factor to the text label
- 0 use the present value of `YPOW` and if `YPOW` $\neq 0$ append the scale factor to the text label

If the user wishes to completely specify the appearance of the y-axis, `YPAUTO` must be set to 0, otherwise the number of digits and decimal places, `NYDIG` and `NYDEC`, may be changed.

YPOW

Default: `YPOW` $= 0$

`YPOW` controls the y-axis scale factor that is appended to the y-axis text label. This scale factor is a power of ten, that is, $10^{\mathtt{YPOW}}$, and the numbers labeling the y-axis should be multiplied by this scale factor to get the correct graph units.

NYDIG

Default: `NYDIG` $= 5$

`NYDIG` controls the total number of digits to be displayed in each of the numbers labeling the y-axis. If `NYDIG` is smaller than required to display the y-axis numbers, `NYDIG` will *not* be increased but a scale factor, $(\times 10^n)$, will be appended to the y-axis text label. If `NYDIG` is larger than required, `NYDIG` will be reduced to the minimum value required. The value of `NYDIG` is updated after each graph is drawn.

NYDEC

Default: `NYDEC` $= -1$

`NYDEC` controls the number of digits to be displayed in the fractional parts of the numbers labeling the y-axis. The value of `NYDEC` is updated after each graph is drawn.

- -1 display the numbers labeling the y-axis as integers, with no decimal point
- 0 display the numbers labeling the y-axis with no fractional part, but with a decimal point
- > 0 display the numbers labeling the y-axis with `NYDEC` digits in the fractional part

YNUMSZ

Default: `%YNUMSZ` $= 3$

`YNUMSZ` controls the size of the numbers labeling the y-axis. See Figure A.39 on page 367. `%YNUMSZ` is a percentage of the height of the window, that is, `YNUMSZ` $=$ `%YNUMSZ` $\times$ (`YUWIND` $-$ `YLWIND`) $\div 100$

YNUMA

Default: `YNUMA` $= -$`YAXISA`

`YNUMA` controls the angle of the base line for the numbers labeling the y-axis. `YNUMA` is the angle, in degrees, measured counterclockwise, between a line parallel to the y-axis and the base line of a number. See Figure A.39 on page 367. By default, `YNUMA` is set as a percentage. If `YNUMA` is set as a percentage, for example, `SET %YNUMA 0`, then `YNUMA` is set to $-$`YAXISA`. The actual value of `%YNUMA` will be ignored. This allows the user to change the angle of the y-axis and keep the base line of the y-axis numbers horizontal.

YITICA

Default: `YITICA` $=$ `XAXISA` $-$ `YAXISA` $+ 180$

`YITICA`, along with `YITICL`, controls the location of the numbers labeling the y-axis at the large tic marks. `YITICA` is the angle, in degrees, measured counterclockwise, between the y-axis and a line joining the base of each large tic mark on the y-axis to the center of the number labeling that tic mark. See Figure A.39 on page 367. By default, `YITICA` is set as a percentage. If `YITICA` is set as a percentage, then `YITICA` is set to `XAXISA` $-$ `YAXISA` $+ 180$. The actual value of `%YITICA` will be ignored. This allows the user to change the angle of the axes and keep the relative location of the y-axis numbers the same.

YITICL

Default: `%YITICL` $= 3$

YITICL, along with YITICA, controls the location of the numbers labeling the y-axis at the large tic marks. YITICL is the distance from the base of each large tic mark on the y-axis to the lower left hand corner of the number labeling that tic mark. See Figure A.39 on page 367. %YITICL is a percentage of the height of the window, that is, $\mathtt{YITICL} = \mathtt{\%YITICL} \times (\mathtt{YUWIND} - \mathtt{YLWIND}) \div 100$

A.6 Axis Box Characteristics

Figure A.41 shows the relation of the world coordinate system to the window and to the graph axis corner locations.

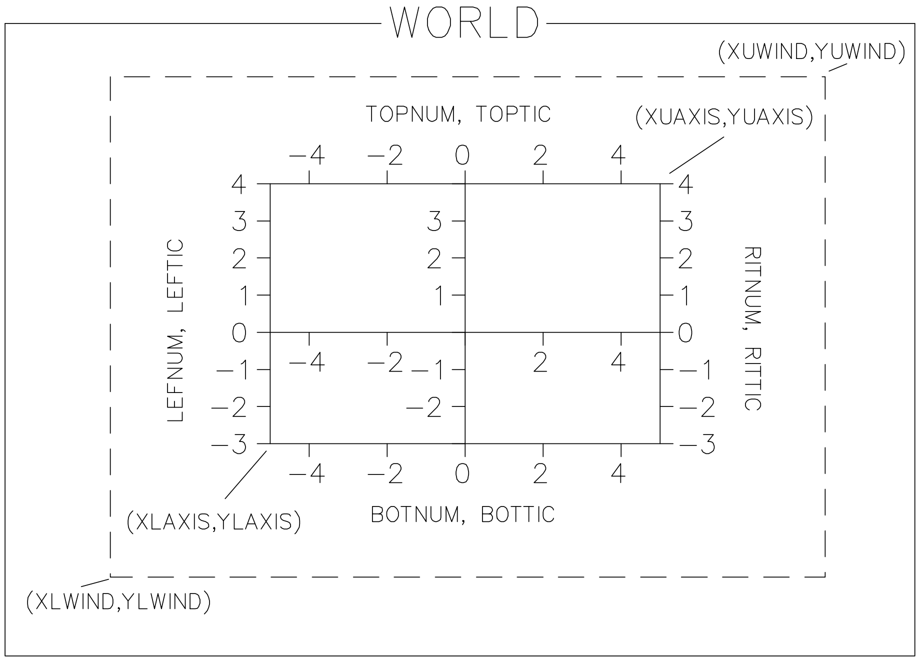


Figure A.41: The window and axis locations

XLWIND

Default: $\mathtt{\%XLWIND} = 0$

XLWIND is the left edge of the window box. See Figure A.41 on page 377. %XLWIND is a

percentage of the entire width of the world coordinate system.

XUWIND

Default: $\mathtt{\%XUWIND} = 0$

XUWIND is the right edge of the window box. See Figure A.41 on page 377. %XUWIND is a percentage of the entire width of the world coordinate system.

YLWIND

Default: $\mathtt{\%YLWIND} = 0$

YLWIND is the bottom edge of the window box. See Figure A.41 on page 377. %YLWIND is a percentage of the entire height of the world coordinate system.

YUWIND

Default: $\mathtt{\%YUWIND} = 0$

YUWIND is the top edge of the window box. See Figure A.41 on page 377. %YUWIND is a percentage of the entire height of the world coordinate system.

BOX

Default: $\mathtt{BOX} = 1$

BOX controls whether or not an axis box is placed around the graph.

0	do not draw an axis box
$\neq 0$	draw an axis box

XLAXIS

Default: $\mathtt{\%XLAXIS} = 15$

XLAXIS controls the position of the left, or lower, end of the x-axis, see Figure A.41 on page 377. This is also the horizontal coordinate of the lower left hand corner of the graph box, if $\mathtt{BOX} = 1$. %XLAXIS is a percentage of the width of the window, that is, $\mathtt{XLAXIS} = \mathtt{XLWIND} + \mathtt{\%XLAXIS} \times (\mathtt{XUWIND} - \mathtt{XLWIND}) \div 100$

XUAXIS

Default: $\texttt{\%XUAXIS} = 95$

XUAXIS controls the position of the right, or upper, end of the x-axis, see Figure A.41 on page 377. This is also the horizontal coordinate of the upper right hand corner of the graph box, if $\texttt{BOX} = 1$. %XUAXIS is a percentage of the width of the window, that is, $\texttt{XUAXIS} = \texttt{XLWIND} + \texttt{\%XUAXIS} \times (\texttt{XUWIND} - \texttt{XLWIND}) \div 100$

XAXISA

Default: $\texttt{XAXISA} = 0$

XAXISA is the angle, in degrees, measured counterclockwise, between a horizontal line and the x-axis. See Figure A.36 on page 356.

YLAXIS

Default: $\texttt{\%YLAXIS} = 15$

YLAXIS is the position of the bottom, or lower, end of the y-axis, see Figure A.41 on page 377. This is also the vertical coordinate of the lower left hand corner of the graph box, if $\texttt{BOX} = 1$. %YLAXIS is a percentage of the height of the window, that is, $\texttt{YLAXIS} = \texttt{\%YLAXIS} \times (\texttt{YUWIND} - \texttt{YLWIND}) \div 100$

YUAXIS

Default: $\texttt{\%YUAXIS} = 90$

YUAXIS is the position of the upper end of the y-axis, see Figure A.41 on page 377. This is also the vertical coordinate of the upper right hand corner of the graph box, if $\texttt{BOX} = 1$. %YUAXIS is a percentage of the height of the window, that is, $\texttt{YUAXIS} = \texttt{\%YUAXIS} \times (\texttt{YUWIND} - \texttt{YLWIND}) \div 100$

YAXISA

Default: $\texttt{YAXISA} = 90$

YAXISA is the angle, in degrees, measured counterclockwise, between a horizontal line and the y-axis. See Figure A.39 on page 367.

BOTNUM

Default: $\texttt{BOTNUM} = 0$

BOTNUM controls the height of the numbers on the bottom edge of the box. $|\texttt{BOTNUM}|$ is the ratio of the height of the numbers on the bottom edge of the box to XNUMSZ, the height of the numbers on the x-axis. The height of these numbers will be $|\texttt{BOTNUM}| \times \texttt{XNUMSZ}$.

BOTNUM is ignored if $\texttt{BOX} = 0$ or if the bottom edge of the axis box overlaps the x-axis.

0	no numbers on the bottom edge of the box
> 0	numbers on bottom edge of the box on same side as x-axis numbers
< 0	numbers on bottom edge of the box on opposite side as x-axis numbers

BOTTIC

Default: $\texttt{BOTTIC} = 1$

BOTTIC controls the lengths of the large and short tic marks on the bottom edge of the box. $|\texttt{BOTTIC}|$ is the ratio of the lengths of the tic marks on the bottom edge of the box to the lengths of the tic marks on the x-axis, where XTICL is the length of the large x-axis tic marks and XTICS is the length of the short x-axis tic marks. The large tic marks on the bottom of the box will have a length of $|\texttt{BOTTIC}| \times \texttt{XTICL}$ and the short tic mark length will be $|\texttt{BOTTIC}| \times \texttt{XTICS}$.

BOTTIC is ignored if $\texttt{BOX} = 0$ or if the bottom edge of the axis box overlaps the x-axis.

0	no tic marks on the bottom edge of the box
> 0	tic marks on bottom edge of the box on the same side as the x-axis tic marks
< 0	tic marks on bottom edge of the box on the opposite side as the x-axis tic marks

RITNUM

Default: $\texttt{RITNUM} = 0$

RITNUM controls the height of the numbers on the right edge of the box. $|\texttt{RITNUM}|$ is the ratio of the height of the numbers on the right edge of the box to YNUMSZ, the height of the numbers on the y-axis. This number height will be $|\texttt{RITNUM}| \times \texttt{YNUMSZ}$.

RITNUM is ignored if $\texttt{BOX} = 0$ or if the right edge of the axis box overlaps the y-axis.

0	no numbers on the right edge of the box
> 0	numbers on right edge of the box on the same side as y-axis numbers
< 0	numbers on right edge of the box on the opposite side as y-axis numbers

RITTIC

Default: RITTIC $= -1$

RITTIC controls the lengths of the large and short tic marks on the right edge of the box. |RITTIC| is the ratio of the lengths of the tic marks on the right edge of the box to the lengths of the tic marks on the y-axis. YTICL is the length of the large y-axis tic marks and YTICS is the length of the short y-axis tic marks. The large tic marks on the right edge of the box will have a length of |RITTIC|×YTICL and the short tic mark length will be |RITTIC|×YTICS.

RITTIC is ignored if BOX $= 0$ or if the right edge of the axis box overlaps the y-axis.

0	no tic marks on the right edge of the box
> 0	tic marks on right edge of the box on the same side as y-axis tic marks
< 0	tic marks on right edge of the box on the opposite side as y-axis large tic marks

TOPNUM

Default: TOPNUM $= 0$

TOPNUM controls the height of the numbers on the top edge of the box. |TOPNUM| is the ratio of the height of the numbers on the top edge of the box to XNUMSZ, the height of the numbers on the x-axis. The height of these numbers will be |TOPNUM|×XNUMSZ.

TOPNUM is ignored if BOX $= 0$ or if the top edge of the axis box overlaps the x-axis.

0	no numbers on the top edge of the box
> 0	numbers on top edge of the box on the same side as x-axis numbers
< 0	numbers on top edge of the box on the opposite side as x-axis numbers

TOPTIC

Default: TOPTIC $= -1$

TOPTIC controls the lengths of the large and short tic marks on the top edge of the box. |TOPTIC| is the ratio of the lengths of the tic marks on the top edge of the box to the lengths of the tic marks on the x-axis, where XTICL is the length of the large x-axis tic marks and XTICS is the length of the short x-axis tic marks. The large tic marks on the top edge of the box will have a length of |TOPTIC|×XTICL and the short tic mark length will be |TOPTIC|×XTICS.

TOPTIC is ignored if BOX $= 0$ or if the top edge of the axis box overlaps the x-axis.

0	no tic marks on the top edge of the box
> 0	tic marks on top edge of the box on the same side as x-axis large tic marks
< 0	tic marks on top edge of the box on the opposite side as x-axis large tic marks

LEFNUM

Default: LEFNUM $= 0$

LEFNUM controls the height of the numbers on the left edge of the box. |LEFNUM| is the ratio of the height of the numbers on the left edge of the box to YNUMSZ, the height of the numbers on the y-axis. The height of these numbers will be |LEFNUM|×YNUMSZ.

LEFNUM is ignored if BOX $= 0$ or if the left edge of the axis box overlaps the y-axis.

0	no numbers on the left edge of the box
> 0	numbers on left edge of the box on the same side as y-axis numbers
< 0	numbers on left edge of the box on the opposite side as y-axis numbers

LEFTIC

Default: LEFTIC $= 1$

LEFTIC controls the lengths of the large and short tic marks on the bottom edge of the box. |LEFTIC| is the ratio of the lengths of the tic marks on the left edge of the box to the lengths of the tic marks on the y-axis, where YTICL is the length of the large y-axis tic marks and YTICS is the length of the short y-axis tic marks. The large tic marks on the left edge of the box will have a length of |LEFTIC|×YTICL and the short tic mark length will be |LEFTIC|×YTICS.

LEFTIC is ignored if BOX $= 0$ or if the left edge of the axis box overlaps the y-axis.

0	no tic marks on the left edge of the box
>0	tic marks on left edge of the box on the same side as y-axis tic marks
<0	tic marks on left edge of the box on the opposite side as y-axis tic marks

VAX/VMS COMMAND PROCEDURE

This command procedure must be executed *before* you invoke PHYSICA. This command procedure can be found in:

`PHYSICA$DIR:PHYSICA_USER_FUNCTIONS.COM`

If the define command:

```
 $ define PHYSICA_USER_FUNCTIONS your_source_file
```

is placed in your `LOGIN.COM` file, this command procedure need not be executed more than once. The shareable image created with this procedure will automatically be found when you run PHYSICA.

```
$SET VERIFY
$! This command procedure creates a shareable image for the 8 PHYSICA
$! user functions AND subroutines.  Normally you need not do this again
$! since the shareable image should still be around from the last use.
$!
$! The FORTRAN sources should be installed in your own directory in a file,
$! named, for example:   disk:[directory]TEST.FOR
$! If this is not present then the defaults installed in
$! PHYSICA$DIR:PHYSICA_USER_FUNCTIONS.FOR will be used.
$!
$! The following DEFINE command should have been set by your system manager
$! define PHYSICA_USER_FUNCTIONS PHYSICA$DIR:PHYSICA_USER_FUNCTIONS
$!
$! If PHYSICA_USER_FUNCTIONS is not defined, the image activator cannot find
$! the shareable image. To activate your own set of functions and subroutines,
$! the following DEFINE command should be in effect. If this is not defined
$! the image activator will then find the default shareable image.
$!
$ define PHYSICA_USER_FUNCTIONS disk:[directory]TEST
$!
$ fortran  PHYSICA_USER_FUNCTIONS
$!
$! The following scripts are needed since transfer vectors may only be
$! created in VAX MACRO.
$!
$macro/object=xfruser1 sys$input
      .title  xfruser1  - Transfer vector for USER1
      .ident  /v01-001/
      .psect  $$xfrvectors,exe,nowrt
```

```
      .transfer user1
      .mask     user1
      jmp       l^user1+2
      .end
$eod
$macro/object=xfruser2 sys$input
      .title  xfruser2  - Transfer vector for USER2
      .ident  /v01-001/
      .psect  $$xfrvectors,exe,nowrt
      .transfer user2
      .mask     user2
      jmp       l^user2+2
      .end
$eod
$macro/object=xfruser3 sys$input
      .title  xfruser3  - Transfer vector for USER3
      .ident  /v01-001/
      .psect  $$xfrvectors,exe,nowrt
      .transfer user3
      .mask     user3
      jmp       l^user3+2
      .end
$eod
$macro/object=xfruser4 sys$input
      .title  xfruser4  - Transfer vector for USER4
      .ident  /v01-001/
      .psect  $$xfrvectors,exe,nowrt
      .transfer user4
      .mask     user4
      jmp       l^user4+2
      .end
$eod
$macro/object=xfruser5 sys$input
      .title  xfruser5  - Transfer vector for USER5
      .ident  /v01-001/
      .psect  $$xfrvectors,exe,nowrt
      .transfer user5
      .mask     user5
      jmp       l^user5+2
      .end
$eod
$macro/object=xfruser6 sys$input
      .title  xfruser6  - Transfer vector for USER6
      .ident  /v01-001/
      .psect  $$xfrvectors,exe,nowrt
      .transfer user6
```



```
      .mask     user6
      jmp       l^user6+2
      .end
$eod
$macro/object=xfruser7 sys$input
      .title  xfruser7  - Transfer vector for USER7
      .ident  /v01-001/
      .psect  $$xfrvectors,exe,nowrt
      .transfer user7
      .mask     user7
      jmp       l^user7+2
      .end
$eod
$macro/object=xfruser8 sys$input
      .title  xfruser8  - Transfer vector for USER8
      .ident  /v01-001/
      .psect  $$xfrvectors,exe,nowrt
      .transfer user8
      .mask     user8
      jmp       l^user8+2
      .end
$eod
$macro/object=xfrsub1 sys$input
      .title  xfrsub1  - Transfer vector for SUB1
      .ident  /v01-001/
      .psect  $$xfrvectors,exe,nowrt
      .transfer sub1
      .mask     sub1
      jmp       l^sub1+2
      .end
$eod
$macro/object=xfrsub2 sys$input
      .title  xfrsub2  - Transfer vector for SUB2
      .ident  /v01-001/
      .psect  $$xfrvectors,exe,nowrt
      .transfer sub2
      .mask     sub2
      jmp       l^sub2+2
      .end
$eod
$macro/object=xfrsub3 sys$input
      .title  xfrsub3  - Transfer vector for SUB3
      .ident  /v01-001/
      .psect  $$xfrvectors,exe,nowrt
      .transfer sub3
      .mask     sub3
```

```
      jmp       l^sub3+2
      .end
$eod
$macro/object=xfrsub4 sys$input
      .title  xfrsub4  - Transfer vector for SUB4
      .ident  /v01-001/
      .psect  $$xfrvectors,exe,nowrt
      .transfer sub4
      .mask     sub4
      jmp       l^sub4+2
      .end
$eod
$macro/object=xfrsub5 sys$input
      .title  xfrsub5  - Transfer vector for SUB5
      .ident  /v01-001/
      .psect  $$xfrvectors,exe,nowrt
      .transfer sub5
      .mask     sub5
      jmp       l^sub5+2
      .end
$eod
$macro/object=xfrsub6 sys$input
      .title  xfrsub6  - Transfer vector for SUB6
      .ident  /v01-001/
      .psect  $$xfrvectors,exe,nowrt
      .transfer sub6
      .mask     sub6
      jmp       l^sub6+2
      .end
$eod
$macro/object=xfrsub7 sys$input
      .title  xfrsub7  - Transfer vector for SUB7
      .ident  /v01-001/
      .psect  $$xfrvectors,exe,nowrt
      .transfer sub7
      .mask     sub7
      jmp       l^sub7+2
      .end
$eod
$macro/object=xfrsub8 sys$input
      .title  xfrsub8  - Transfer vector for SUB8
      .ident  /v01-001/
      .psect  $$xfrvectors,exe,nowrt
      .transfer sub8
      .mask     sub8
      jmp       l^sub8+2
```



```
      .end
$eod
$!
$! Link these 8 user functions and 8 subroutines as a shareable image,
$!       disk:[directory]test.exe
$! The names usern_vector and subn_vector are dummy but must be unique.
$!
$link/share PHYSICA_USER_FUNCTIONS,sys$input/opt
gsmatch=lequal,1,0
cluster=user1_vector,,,xfruser1
cluster=user2_vector,,,xfruser2
cluster=user3_vector,,,xfruser3
cluster=user4_vector,,,xfruser4
cluster=user5_vector,,,xfruser5
cluster=user6_vector,,,xfruser6
cluster=user7_vector,,,xfruser7
cluster=user8_vector,,,xfruser8
cluster=sub1_vector,,,xfrsub1
cluster=sub2_vector,,,xfrsub2
cluster=sub3_vector,,,xfrsub3
cluster=sub4_vector,,,xfrsub4
cluster=sub5_vector,,,xfrsub5
cluster=sub6_vector,,,xfrsub6
cluster=sub7_vector,,,xfrsub7
cluster=sub8_vector,,,xfrsub8
$eod
$delete/NOconfirm xfruser*.obj;*,xfrsub*.obj;*
$set noverify
```

AlphaVMS COMMAND PROCEDURE

This command procedure must be executed *before* you invoke PHYSICA. This command procedure can be found in:

PHYSICA$DIR:PHYSICA_USER_FUNCTIONS.COM

If the define command:

```
 $ define PHYSICA_USER_FUNCTIONS your_source_file
```

is placed in your LOGIN.COM file, this command procedure need not be executed more than once. The shareable image created with this procedure will automatically be found when you run PHYSICA.

```
$SET VERIFY
$! This command procedure creates a shareable image for the 8 user functions
$! and 8 subroutines. The FORTRAN sources should be installed in your own
$! directory in a file, named, for example:   disk:[directory]TEST.FOR
$! If this is not present then the defaults installed in
$! PHYSICA$DIR:PHYSICA_USER_FUNCTIONS.FOR will be used.
$!
$! Your system manager should have created the default logical name:
$! DEFINE PHYSICA_USER_FUNCTIONS PHYSICA$DIR:PHYSICA_USER_FUNCTIONS
$! If PHYSICA_USER_FUNCTIONS is not defined, the image activator cannot find
$! the shareable image. To activate your own set of functions and subroutines,
$! the following DEFINE command should be in effect. If this is not defined
$! the image activator will find the default shareable image.
$!
$! define PHYSICA_USER_FUNCTIONS disk:[directory]test
$!
$! The logical name PHYSICA_USER_FUNCTIONS will need to be re-defined in each
$! new process where you want to make use of your own routines.
$!
$ fortran  PHYSICA_USER_FUNCTIONS.for
$!
$! Link these 8 user functions and 8 subroutines as a sharerable image,
$!       disk:[directory]test.exe
$! The names usern_vector and subn_vector are dummy but must be unique.
$!
$link/shareable PHYSICA_USER_FUNCTIONS.obj, sys$input/opt
 GSMATCH=lequal,1,1000
 SYMBOL_VECTOR=(user1=PROCEDURE,-
               user2=PROCEDURE,-
```



```
               user3=PROCEDURE,-
               user4=PROCEDURE,-
               user5=PROCEDURE,-
               user6=PROCEDURE,-
               user7=PROCEDURE,-
               user8=PROCEDURE,-
                sub1=PROCEDURE,-
                sub2=PROCEDURE,-
                sub3=PROCEDURE,-
                sub4=PROCEDURE,-
                sub5=PROCEDURE,-
                sub6=PROCEDURE,-
                sub7=PROCEDURE,-
                sub8=PROCEDURE)
$EOD
$set noverify
```

USER ROUTINE SOURCE CODE

VMS: Following is the default source code for the eight functions and the eight subroutines that are linked to PHYSICA via the shareable image. This source code can be found in:

PHYSICA$DIR:PHYSICA_USER_FUNCTIONS.FOR

UNIX: Following is the source code for the eight user defined functions and the eight user defined subroutines that are linked into PHYSICA by default. This source code can be found in:

phys_user.f

The process to change the user defined routines in physica follows:

1) edit the file phys_user.f to put in your versions of user1, ..., user8, sub1, ..., sub8
2) compile it, e.g., f77 -c phys_user.f
3) archive it with ar -rsv physica.a phys_user.o
4) link the program with physica.link

For off-site users: phys_user.f can be found in the physica-link tar file, along with the necessary archives.

```
      REAL*8 FUNCTION USER1(A)
      IMPLICIT REAL*8 (A-H,O-Z)
      USER1=A
      RETURN
      END

      REAL*8 FUNCTION USER2(A,B)
      IMPLICIT REAL*8 (A-H,O-Z)
      USER2=A+B
      RETURN
      END

      REAL*8 FUNCTION USER3(A,B,C)
      IMPLICIT REAL*8 (A-H,O-Z)
      USER3=A+B+C
      RETURN
      END
```



```
      REAL*8 FUNCTION USER4(A,B,C,D)
      IMPLICIT REAL*8 (A-H,O-Z)
      USER4=A+B+C+D
      RETURN
      END

      REAL*8 FUNCTION USER5(A)
      IMPLICIT REAL*8 (A-H,O-Z)
      USER5=5.0
      RETURN
      END

      REAL*8 FUNCTION USER6(A)
      IMPLICIT REAL*8 (A-H,O-Z)
      USER6=6.0
      RETURN
      END

      REAL*8 FUNCTION USER7(A,B)
      IMPLICIT REAL*8 (A-H,O-Z)
      SUM = 0.0
      DO I = 1, INT(A)
        SUM = SUM+B
      END DO
      USER7 = SUM
      RETURN
      END

      REAL*8 FUNCTION USER8(A)
      IMPLICIT REAL*8 (A-H,O-Z)
      USER8=8.0
      RETURN
      END

      SUBROUTINE SUB1(IATYPE,ICODE,IUPDATE,IER,X1,X2,ADIFF)
      INTEGER*4 IATYPE(15),ICODE(3,15),IUPDATE(15),IER
      REAL*8    ADIFF, X1(1), X2(1)

C   Determine if vector X1 is different than vector X2
C   If there is a difference set ADIFF = 1, otherwise set ADIFF = 0
CCC
      IF( IATYPE(1) .NE. 1 )GO TO 991
      IF( IATYPE(2) .NE. 1 )GO TO 992
      IF( IATYPE(3) .NE. 0 )GO TO 993
      IF( ICODE(1,1) .NE. ICODE(1,2) )GO TO 994
```

```
      IUPDATE(3) = 1     ! third argument to be updated
      ADIFF = 0.0D0
      DO I = 1, ICODE(1,1)
        IF( X1(I) .NE. X2(I) )THEN
          ADIFF = 1.0D0
          RETURN
        END IF
      END DO
      RETURN
 991  WRITE(*,*)' error in SUB1:  first argument is not a vector'
      IER = -1
      RETURN
 992  WRITE(*,*)' error in SUB1:  second argument is not a vector'
      IER = -1
      RETURN
 993  WRITE(*,*)' error in SUB1:  third argument is not a scalar'
      IER = -1
      RETURN
 994  WRITE(*,*)' error in SUB1:  input vectors not the same length'
      IER = -1
      RETURN

C  Setting IER to -1 indicates to PHYSICA that an error has occured in
C  the subroutine.  This is like an alternate RETURN. If you call this
C  subroutine in a script command file, and the error occurs, the script
C  will stop execution

      END

      SUBROUTINE SUB2(IATYPE,ICODE,IUPDATE,IER,XIN,XOUT,A)
      INTEGER*4 IATYPE(15),ICODE(3,15),IUPDATE(15),IER
      REAL*8    A, XIN(1), XOUT(1)

C   Divide a vector by a scalar and put the output in another vector
C   e.g., CALL SUB2 XIN XOUT A  and  XOUT will be XIN/A
CCC
      IF( IATYPE(1) .NE. 1 )GO TO 991
      IF( IATYPE(2) .NE. 1 )GO TO 992
      IF( IATYPE(3) .NE. 0 )GO TO 993
      IUPDATE(2) = 1     ! second argument to be updated
      NPTS = MIN(ICODE(1,1),ICODE(1,2))  ! use minimum length
      DO I = 1, NPTS
        IF( XIN(I) .GT. 100.0D0 )GO TO 994
        XOUT(I) = XIN(I)/A
      END DO
      RETURN
```



```
 991  WRITE(*,*)' error in SUB2:  first argument is not a vector'
      IER = -1
      RETURN
 992  WRITE(*,*)' error in SUB2:  second argument is not a vector'
      IER = -1
      RETURN
 993  WRITE(*,*)' error in SUB2:  third argument is not a scalar'
      IER = -1
      RETURN
 994  WRITE(*,*)' error in SUB2:  XIN(I) > 100'
      IER = -1
      RETURN
      END

      SUBROUTINE SUB3(IATYPE,ICODE,IUPDATE,IER,MATRIX,DIAGONAL)
      INTEGER*4 IATYPE(15),ICODE(3,15),IUPDATE(15),IER
      REAL*8 MATRIX(1), DIAGONAL(1)

C   Extract the diagonal from a square matrix and put it into a vector
C   e.g., CALL SUB3 MATRIX DIAGONAL
C        and  DIAGONAL will be MATRIX[J,J] for J = 1, #
CCC
      IF( IATYPE(1) .NE. 2 )GO TO 991
      IF( IATYPE(2) .NE. 1 )GO TO 992
      IUPDATE(2) = 1     ! second argument to be updated
      NROWS = ICODE(1,1)
      NCOLS = ICODE(2,1)
      IF( NCOLS .NE. NROWS )GO TO 993
      DO J = 1, NCOLS
        DIAGONAL(J) = MATRIX(J+(J-1)*NCOLS)
      END DO
      RETURN
 991  WRITE(*,*)' error in SUB3:  first argument is not a matrix'
      IER = -1
      RETURN
 992  WRITE(*,*)' error in SUB3:  second argument is not a vector'
      IER = -1
      RETURN
 993  WRITE(*,*)' Use a square matrix for SUB3'
      IER = -1
      RETURN
      END

      SUBROUTINE SUB4(IATYPE,ICODE,IUPDATE,IER,X,M,XOUT)
      INTEGER*4 IATYPE(15),ICODE(3,15),IUPDATE(15),IER
      REAL*8    M(1), X(1), XOUT(1)
```

```
C   Calculate XOUT = X<>M  (inner product)
C   where X is a vector, M is a matrix, XOUT is the output vector
CCC
      IF( IATYPE(1) .NE. 1 )GO TO 991
      IF( IATYPE(2) .NE. 2 )GO TO 992
      IF( IATYPE(3) .NE. 1 )GO TO 993
      IUPDATE(3) = 1     ! third argument to be updated
      NROWS = ICODE(1,2)
      NCOLS = ICODE(2,2)
      NUMX  = ICODE(1,1)
      NUMXO = ICODE(1,3)
      IF( NUMX .NE. NROWS )GO TO 994
      IF( NUMXO .NE. NCOLS )GO TO 995
      DO I = 1, NCOLS
        XOUT(I) = 0.0D0
        DO J = 1, NROWS
          XOUT(I) = XOUT(I) + X(J)*M(I+(J-1)*NCOLS)
        END DO
      END DO
      RETURN
 991  WRITE(*,*)' *** error in SUB4'
      WRITE(*,*)'     first argument is not a vector'
      IER = -1
      RETURN
 992  WRITE(*,*)' *** error in SUB4'
      WRITE(*,*)'     second argument is not a matrix'
      IER = -1
      RETURN
 993  WRITE(*,*)' *** error in SUB4'
      WRITE(*,*)'     third argument is not a vector'
      IER = -1
      RETURN
 994  WRITE(*,*)' *** error in SUB4'
      WRITE(*,*)'     vector length not equal to row dimension of matrix'
      IER = -1
      RETURN
 995  WRITE(*,*)' *** error in SUB4'
      WRITE(*,*)
     # '     output vector length not equal to column dimension of matrix'
      IER = -1
      RETURN
      END

      SUBROUTINE SUB5(IATYPE,ICODE,IUPDATE,IER,LFILE,X,Y)
      INTEGER*4    IATYPE(15),ICODE(3,15),IUPDATE(15),IER
```



```
      INTEGER*4    LENF, IUNIT, I, NPTS
      REAL*8       X(1), Y(1), XDUM
      LOGICAL*4    LUOPEN
      LOGICAL*1    LFILE(1)
      CHARACTER*80 AFILE

C   Read a vector X from a file, multiply it by 5 and add it to vector Y
C   with the result put into X
C   e.g., CALL SUB5 'FILE.DAT' X Y
C         and X will be X*5+Y
CCC
      IF( IATYPE(1) .NE.-1 )GO TO 991
      IF( IATYPE(2) .NE. 1 )GO TO 992
      IF( IATYPE(3) .NE. 1 )GO TO 993
      IF( ICODE(1,1) .GT. 80 )GO TO 994
      IUPDATE(2) = 1     ! second argument to be updated
      LENF = ICODE(1,1)
      DO I = 1, LENF
        AFILE(I:I) = CHAR(LFILE(I))
      END DO
      DO 2 I = 30, 99
        INQUIRE(UNIT=I,OPENED=LUOPEN,ERR=2)
        IF( .NOT.LUOPEN )THEN
          IUNIT = I
          GO TO 4
        END IF
   2  CONTINUE
   4  OPEN(FILE=AFILE(1:LENF),UNIT=IUNIT,STATUS='OLD',READONLY,SHARED,ERR=995)
      I = 1
  10  READ(33,*,ERR=996,END=20)XDUM
      IF( I .GT. ICODE(1,2) )THEN
        WRITE(*,*)' message from SUB5'
        WRITE(*,*)' max. number of elements from the file has been read'
        WRITE(*,*)' but there is more that could be read'
        GO TO 20
      END IF
      X(I) = XDUM
      I = I+1
      GO TO 10
  20  NPTS = MIN(ICODE(1,2),ICODE(1,3))
      DO I = 1, NPTS
        X(I) = X(I)*5.0D0+Y(I)
      END DO
      ICODE(1,2) = NPTS  ! update the length of X
      RETURN
 991  WRITE(*,*)' *** error in SUB5'
```

```
      WRITE(*,*)'     first argument is not a string'
      IER = -1
      RETURN
  992 WRITE(*,*)' *** error in SUB5'
      WRITE(*,*)'     second argument is not a vector'
      IER = -1
      RETURN
  993 WRITE(*,*)' *** error in SUB5'
      WRITE(*,*)'     third argument is not a vector'
      IER = -1
      RETURN
  994 WRITE(*,*)' *** error in SUB5'
      WRITE(*,*)'     string is longer than 80 characters'
      IER = -1
      RETURN
  995 WRITE(*,*)' *** error in SUB5'
      WRITE(*,*)'     unable to open file: '//AFILE(1:LENF)
      IER = -1
      RETURN
  996 WRITE(*,*)' *** error in SUB5'
      WRITE(*,9961)I,AFILE(1:LENF)
 9961 FORMAT('     reading line#',I3,' from file: ',A)
      IER = -1
      RETURN
      END

      SUBROUTINE SUB6(IATYPE,ICODE,IUPDATE,IER,X,Y)
      INTEGER*4 IATYPE(15),ICODE(3,15),IUPDATE(15),IER
      REAL*8 X(1), Y(1)

C   Multiply a vector by 6 and add it to another vector
C   e.g., CALL SUB6 X Y
C         and X will be X+6*Y
CCC
      IF( IATYPE(1) .NE. 1 )GO TO 991
      IF( IATYPE(2) .NE. 1 )GO TO 992
      IUPDATE(2) = 1     ! second argument to be updated
      NPTS = MIN(ICODE(1,1),ICODE(1,2))
      DO I = 1, NPTS
        X(I) = X(I)+Y(I)*6.0D0
      END DO
      RETURN
  991 WRITE(*,*)' *** error in SUB6'
      WRITE(*,*)'     first argument is not a vector'
      IER = -1
      RETURN
```



```
  992 WRITE(*,*)' *** error in SUB6'
      WRITE(*,*)'     second argument is not a vector'
      IER = -1
      RETURN
      END

      SUBROUTINE SUB7(IATYPE,ICODE,IUPDATE,IER,X,Y)
      INTEGER*4 IATYPE(15),ICODE(3,15),IUPDATE(15),IER
      REAL*8 X(1), Y(1)

C   Multiply a vector by 7 and add it to another vector
C   e.g., CALL SUB7 X Y
C         and X will be X+7*Y
CCC
      IF( IATYPE(1) .NE. 1 )GO TO 991
      IF( IATYPE(2) .NE. 1 )GO TO 992
      IUPDATE(2) = 1     ! second argument to be updated
      NPTS = MIN(ICODE(1,1),ICODE(1,2))
      DO I = 1, NPTS
        X(I) = X(I)+Y(I)*7.0D0
      END DO
      RETURN
  991 WRITE(*,*)' *** error in SUB7'
      WRITE(*,*)'     first argument is not a vector'
      IER = -1
      RETURN
  992 WRITE(*,*)' *** error in SUB7'
      WRITE(*,*)'     second argument is not a vector'
      IER = -1
      RETURN
      END

      SUBROUTINE SUB8(IATYPE,ICODE,IUPDATE,IER,X,Y)
      INTEGER*4 IATYPE(15),ICODE(3,15),IUPDATE(15),IER
      REAL*8 X(1), Y(1)

C   Multiply a vector by 8 and add it to another vector
C   e.g., CALL SUB8 X Y
C         and X will be X+8*Y
CCC
      IF( IATYPE(1) .NE. 1 )GO TO 991
      IF( IATYPE(2) .NE. 1 )GO TO 992
      IUPDATE(2) = 1     ! second argument to be updated
      NPTS = MIN(ICODE(1,1),ICODE(1,2))
      DO I = 1, NPTS
        X(I) = X(I)+Y(I)*8.0D0
```

User Routine Source Code

```
      END DO
      RETURN
991   WRITE(*,*)' *** error in SUB8'
      WRITE(*,*)'     first argument is not a vector'
      IER = -1
      RETURN
992   WRITE(*,*)' *** error in SUB8'
      WRITE(*,*)'     second argument is not a vector'
      IER = -1
      RETURN
      END
```

Index