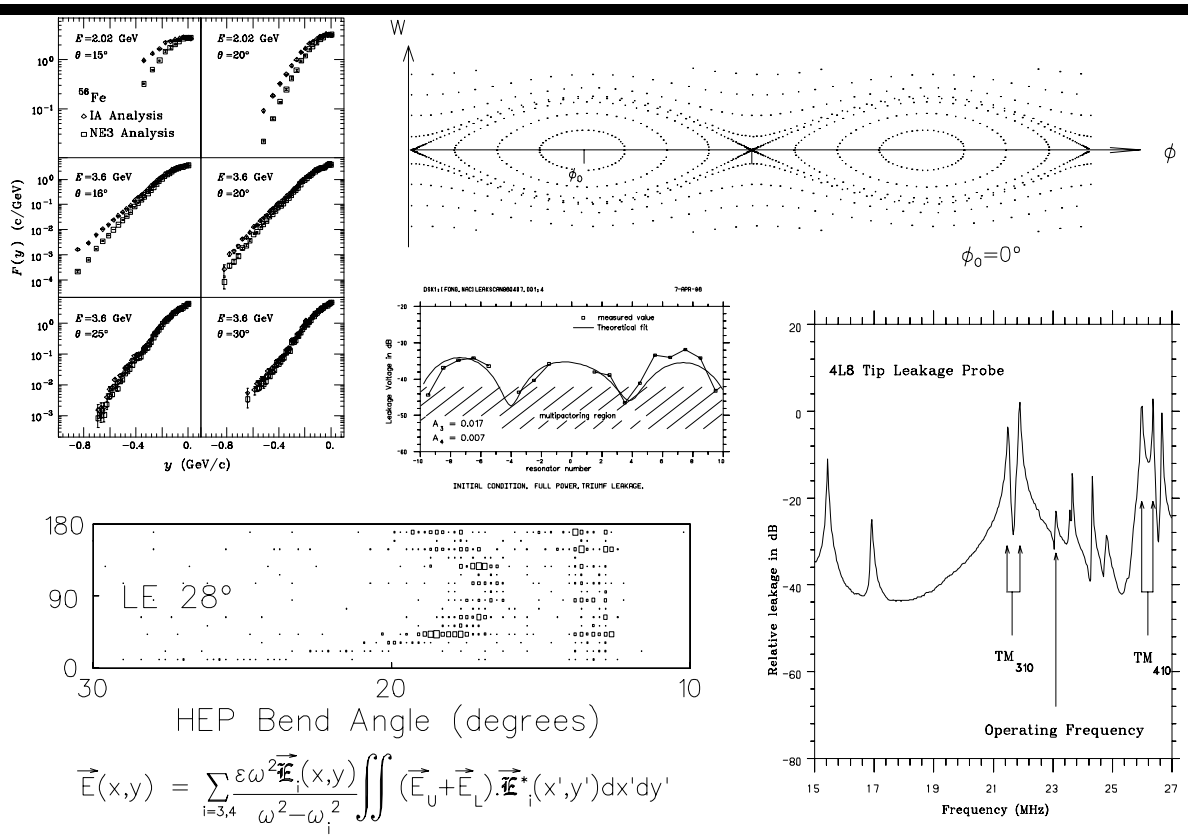
TRIUMF	4004 Wesbrook Mall, Vancouver, B.C., Canada V6T 2A3			
Computing Document	J.L. Chuma	February 1998	TRI-CD-98-01	v1.0

PHYSICA ©

USER'S GUIDE

Mathematical Analysis and Data Visualization Software



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.

PHYSICA commands are simple and forgiving of input errors. The principle uses of PHYSICA are to input and manipulate data and then to produce graphical representations of this data. Data can be read from files, input interactively via the keyboard, or generated internally in various ways.

Introduction

1.1 How to run PHYSICA

VMS: If PHYSICA has been properly installed on your system, you can run it by just typing PHYSICA at the DCL prompt. The PHYSICA command can be defined for all users by the System Manager, or you can place the following in your LOGIN.COM file:

```
DEFINE PHYSICA$DIR <physica-directory>
DEFINE TRIUMF$FONTS <font-directory>
PHYSICA :== $PHYSICA$DIR:PHYSICA
```

where <physica-directory> is the place where the program and help files are kept, and <font-directory> is the place where the VAXFONT.DAT file is kept.

UNIX: If PHYSICA is installed, you can type physica at your shell prompt to run it. The best way to run PHYSICA is from a shell script, as follows:

```
#!/bin/csh
setenv PHYSICA_DIR <physica-directory>
setenv TRIUMF_FONTS <font-directory>
<physica-directory>/physica
```

where <physica-directory> is the place where the program and help files are kept, and <font-directory> is the place where the vaxfont.dat file is kept.

1.2 A first example

One can immediately start using PHYSICA by learning just a few basic commands. A straight-forward use for PHYSICA is to read two lists of numbers from a file, draw a graph of one list versus the other list, label the graph with the date and some text string, and, finally, obtain a hardcopy of the final picture. For example:

```
READ FILE.DAT X Y
GRAPH X Y
TEXT DATE
TEXT 'Some text label'
HARDCOPY
```

Note: The program informs you that it is ready to accept input when the PHYSICA: prompt appears.

PHYSICA is very useful when a drawing needs to be tailored to exact specifications. For example, many publications require tic marks on graph axes to point inward. This is accomplished with the SET command. For example:

```
PHYSICA: SET
 Enter: name { value } >> XTICA  90
 Enter: name { value } >> YTICA -90
 Enter: name { value } >>
PHYSICA:
```

PHYSICA makes full use of the TRIUMF graph plotting package, GPLOT.

A necessary supplement to this manual is the PHYSICA Reference Manual which discusses the PHYSICA commands, functions and operators in detail.

1.3 What is in this manual

This manual starts with the introduction and an explanation of how to run the PHYSICA program, followed by a description of how the graphics display device type is chosen. This is followed by a general discussion of the graphics plotting units, the program's own graphics windows, and the two graphics orientations which are available.

The next chapter discusses program instruction sources and instruction types. Instructions can be entered interactively from the keyboard, or can be read from script files. A terminal interface is used which closely mimics the DCL command recall facility. Script files are discussed, including the branching and looping capabilities. The four types of program instruction are discussed: script file comments, operating system commands, assignments, evaluations, and program commands.

The next chapter explains how variables are used in the program, including the maximum allowable number and size of variables. Variable indices are a major feature of the program, and are described in some detail. The program allows for string variables as well as numeric variables, and some novel uses for string variables are discussed.

The bulk of this manual is a collection of examples, roughly divided up into categories by chapter.

Users are referred to the PHYSICA Reference Manual for detailed descriptions of the PHYSICA commands, functions and operators. Those who are familiar with the predecessor program, PLOTDATA, are referred to the PLOTDATA to PHYSICA Conversion Manual for tips on converting PLOTDATA scripts to PHYSICA.

1.4 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`.

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.

1.5 Questions or comments?

Contact Joseph Chuma, if you have problems, questions or comments.

e-mail: chuma@triumf.ca ⟵ *preferred method of contact*
telephone: (604) 222-1047 extension 6310

2 GETTING STARTED

2.1 Graphics display device type

Device	monitor_name
Digital VT100	VT100
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 2.1: Supported graphics display device types

PHYSICA needs to know your graphics display device type. This information can be passed to the program by a logical name, on VMS systems, by an environment variable, on UNIX systems, or by interactively answering a question.

VMS: `DEFINE TRIUMF_TERMINAL_TYPE monitor_name`

UNIX: `setenv TRIUMF_TERMINAL_TYPE monitor_name`

If TRIUMF_TERMINAL_TYPE is assigned to one of the valid monitor type names, as listed in the second column of Table 2.1, before running PHYSICA, then no initial question will be asked. If PHYSICA cannot translate TRIUMF_TERMINAL_TYPE into a valid monitor type name, the monitor type will be interactively requested.

The graphics hardcopy device type is chosen while running PHYSICA. The default graphics hardcopy device type is HPLASER, a bitmap device, at 150 dots per inch. The graphics hardcopy device type may be changed at any time by using the DEVICE command.

2.1.1 X Window displays

Selecting monitor type X indicates that you are using a workstation or terminal that supports the X Window System. In this case, PHYSICA can be run on the local workstation or on a remote computer that also has the X Window System. After logging in to a remote machine, and before running PHYSICA, the appropriate command must be issued to enable your X display to be used.

If the remote system is a VMS system, the following command should be used:

```
SET DISPLAY/CREATE/NODE=node_name/TRANSPORT=transport_name
```

The node_name parameter is the name of your local workstation or X terminal, as it is known to the remote host. The transport_name parameter specifies the type of communications link to be used, typically, TCPIP or DECNET.

If the remote system is a UNIX system, the following command should be used. For a TCP/IP connection:

```
setenv DISPLAY node_name:0
```

and for a DECNET connection, if supported on the local and the remote systems:

```
setenv DISPLAY node_name::0
```

Once the X network connection is established, all graphics functions, including keyboard and mouse input, will be performed on the local workstation, even though PHYSICA is running on the remote host.

2.2 Plotting units

The plotting units horizontal and vertical ranges, that is, the size of the world coordinate system, is determined by the graphics hardcopy device type and the orientation, either landscape or portrait.

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 2.2: Plotting units for HPLaserJet 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 2.3: Plotting units for InkJet devices

If you change hardcopy devices, with the DEVICE command, or if you change the orientation, with the ORIENTATION command, the plotting units will change.

By default, the plotting units are expressed in centimeters. The units type can be changed to inches, or back to centimeters, with the SET UNITS command.

Since the plotting units are in actual device units, lengths and distances are not distorted on graphics hardcopy devices. Commensurateness is automatic. For example, no horizontal or vertical scaling

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 2.4: Plotting units for Printronix, LA100, and ThinkJet devices

paper		LANDSCAPE		PORTRAIT	
size	*units*	*horizontal*	*vertical*	*horizontal*	*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 2.5: Plotting units for PostScript devices

paper		LANDSCAPE		PORTRAIT	
size	*units*	*horizontal*	*vertical*	*horizontal*	*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 2.6: Plotting units for pen plotter 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 2.7: Plotting units for LN03+ and Imagen 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 2.8: Plotting units for GKS graphics metafiles

need be done to draw a circle.

Commensurate graphs are a different matter, and can be obtained by means of the SET AUTOSCALE or the SCALE command.

2.3 PHYSICA windows

By default, there is always at least one rectangle drawn on the graphics monitor screen. The largest rectangle represents the world boundary, that is, the edges of the hardcopy page. A smaller inner rectangle, drawn with a dashed line, represents a PHYSICA window within the page.

The window and page boundary rectangles are for the user's reference only and *will not appear on a hardcopy*. They can be turned off with the DISABLE BORDER command, and turned back on with the ENABLE BORDER command.

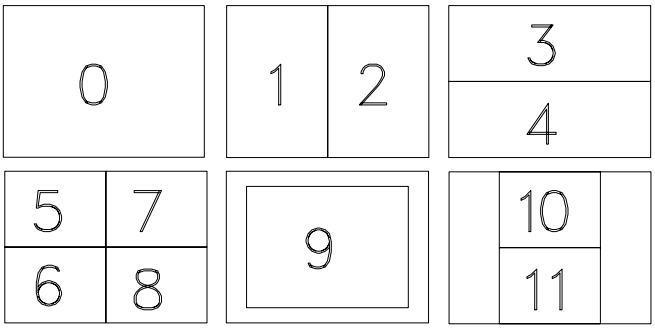


Figure 2.1: The pre-defined PHYSICA windows in landscape orientation

Windows are chosen with the WINDOW command. There are twelve (12) pre-defined windows, see Figures 2.1 and 2.2. Window zero is the default window and is the full page. It is a simple matter to

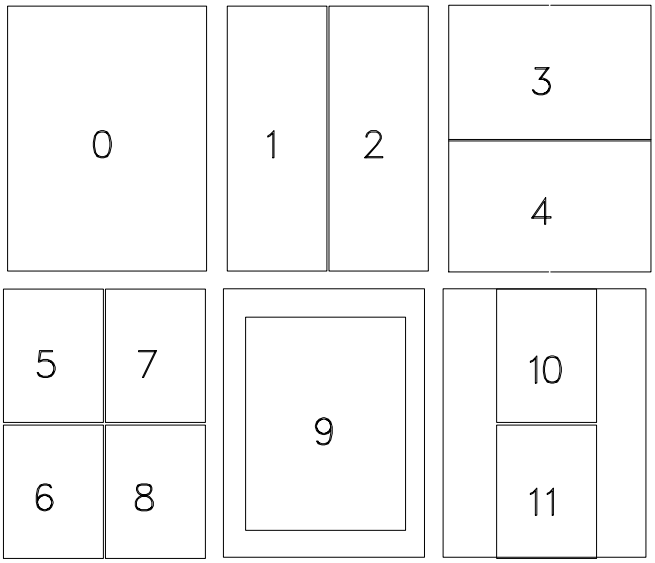


Figure 2.2: The pre-defined PHYSICA windows in portrait orientation

define your own special windows with the WINDOW command.

Windows are defined by four keywords: %XLWIND, %XUWIND, %YLWIND and %YUWIND which are used by GPLOT, the TRIUMF graph plotting package. These characteristics are explained in **Appendix A** of the PHYSICA Reference Manual.

2.4 Graphics orientation

There are two graphics orientations available in the PHYSICA program. Use the ORIENTATION command to change the graphics orientation.

In LANDSCAPE orientation the large dimension is horizontal. In PORTRAIT orientation the large dimension is vertical. The default orientation is LANDSCAPE. See Figure 2.3.

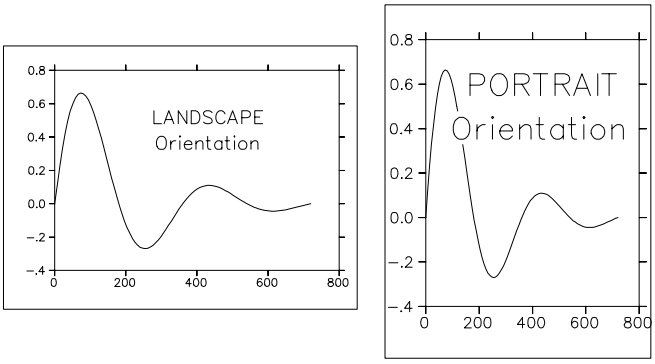


Figure 2.3: Examples of LANDSCAPE and PORTRAIT orientations

3 PROGRAM INSTRUCTIONS

Instructions to PHYSICA can come from two sources:

- interactively via the terminal keyboard
- non-interactively via a PHYSICA script file

The maximum total length of an input line from any of these sources is 255 characters.

It is possible to continue any instruction on several input lines. If an input line *ends* with a minus character, -, the next input line will be appended to the end of that line. The interactive mode prompt that is displayed for continuation lines is `continuation:` . For example,

```
PHYSICA: READ -
continuation: FILE.DAT -
continuation: X Y Z
```

Remember, though, that the maximum total length of an input line is 255 characters, *including all continuation lines*.

3.1 Keyboard input

One way to enter instructions is interactively, via the keyboard. The terminal interface closely mimics the DCL command recall facility.

The prompt `PHYSICA:` indicates that the program is ready for keyboard input.

There are three input line recall buffers: the dynamic buffer, the static buffer, and the keypad buffer. The `BUFFER` command controls these line recall buffers. Table 3.9 shows the control keys recognized by the terminal interface. Table 3.10 shows the function keys recognized by the terminal interface.

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 3.9: 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 3.10: Function keys recognized by the terminal interface

3.2 Script files

Instructions to the program may come from a PHYSICA script file, also called a macro file. One script file can execute another script file, and so on, up a limit of twenty (20) nested files.

To direct the input to be from such a file, use the EXECUTE command, or, equivalently, use the 'at', @, character. For example: `EXECUTE file.pcm` or `@file.pcm`

3.2.1 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 DATAFILE dum.dat
physica
read $DATAFILE x y z
```

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

```
setenv DATAFILE dum
physica
read $DATAFILE.dat x y z
```

3.2.2 Parameter correction

By default, in many cases, if an incorrect parameter of a valid command is read from the script file or is substituted from the parameter list, then the user will be asked to enter the correct information from the terminal keyboard, and the command will then be executed.

VMS: Whenever you are prompted for a command parameter, typing control-z, that is, simultaneously pressing the control key and the z key, aborts execution of that command, and of the script and any higher level calling scripts. If you type control-space, which is a null, only that command, not the script, will be aborted.

UNIX: Whenever you are prompted for a command parameter, typing control-d aborts execution of that command, and of the script and any higher level calling scripts. If you type control-space, which is a null, only that command, not the script, will be aborted.

During execution of a script file, messages are displayed on the terminal screen. If there is a non-severe input error or if required parameters are missing, the user is prompted to make corrections via the terminal keyboard and then execution of the script file resumes. This property can be turned off with the DISABLE PROMPTING command.

3.2.3 Flow control

The user may type control-c, that is, type the control key and the c key simultaneously, any time during execution of a script to abort that script, and return to interactive input.

Within a script file, it is possible to have labels, GOTO statements, IF statements or blocks, and DO loop blocks.

See the description of the EXECUTE command in the PHYSICA Reference Manual for more information. See also the descriptions of other relevant commands: STACK, TERMINAL, RETURN, DISPLAY, INQUIRE, BELL, and WAIT.

3.2.4 Filename extension

The default script filename extension is .pcm, so, by default, the command @file is equivalent to @file.pcm

The default filename extension can be changed with the EXTENSION command.

3.2.5 Comments

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

```
READ file.dat x y z  ! This is a comment
```

3.2.6 Echoing

If the ENABLE ECHO command is entered, the commands that are read from the file will be displayed on the terminal screen. This is useful for following the progress of a script file. If ECHO is disabled,

with the DISABLE ECHO command, but is enabled within a script, it will be enabled only while within that script.

3.2.7 Flow control

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

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

3.2.8 Parameter passing

Parameters that are entered with the EXECUTE command and after the file name are used in one of two ways. Either the n^{th} parameter will replace the n^{th} question mark, ?, that is found in the file, or the n^{th} parameter in the list will replace all ?n's found in the file. For example, suppose you have the following script file:

```
! This script file is called file.pcm
READ ?1 ?2 ?3
WRITE\APPEND ?1 x y
```

and you interactively enter the command

```
EXECUTE file file.dat a b     or
@file file.dat a b
```

then file.dat is substituted for every occurrence of ?1, a is substituted for every occurrence of ?2, and b is substituted for every occurrence of ?3. The commands that would be executed are:

```
READ file.dat a b
WRITE\APPEND file.dat x y
```

The same thing could be accomplished with sequential parameter substitution:

```
READ ? ? ?
WRITE\APPEND ? x y
```

but you would have to enter EXECUTE file file.dat a b file.dat to get the same result.

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.

3.2.9 Initialization file

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. That script will then be executed automatically whenever PHYSICA is run.

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.

3.2.10 Creating script files interactively

Executable script files may be created while running PHYSICA by making use of the STACK command. Subsequent input to the program will be written to the specified stack file.

Any incorrect commands that are entered interactively will not be stacked in the file, and any inappropriate command parameters will be corrected before they are written to the stack file.

3.2.11 Labels and GOTO statements

Labels and GOTO's can only be used in script files.

A label is a string terminated with a colon, :, 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. For example:

```
 ...
GOTO A_LABEL
 ...
A_LABEL:
 ...
```

3.2.12 DO loop blocks

DO loops can only be used in script files.

DO loops in PHYSICA are similar to Fortran do loops, but *must* be terminated with an ENDDO statement. The range of the looping variable can be any expression resulting in a vector. The loop will execute a number of times equal to the length of the loop range vector, with the loop variable taking on

successive values of the loop range vector. Nested loops are allowed. The maximum number of DO loops in a script is fifty (50).

The looping variable will be made into a scalar variable. In the following example, the variable J will be a scalar.

```
 ...
DO J = X    ! looping variable is J, range of the loop is X
 ...        ! J will take on each value of X
ENDDO
 ...
```

The looping variable is initialized to the first value of the loop range. The commands contained within the loop block are executed a number of times equal to the length of the loop range vector.

3.2.13 IF statements and blocks

IF statements and blocks can only be used in script files.

The general form of an IF statement is: `IF (boolean) THEN instruction`
The general form of an IF block is:

```
IF (boolean) THEN
 ...
ENDIF
```

Use an IF statement to execute a single instruction. Use an IF block to execute a block of instructions. An IF block *must* be terminated with an ENDIF.

The boolean can be any expression with a scalar result. The boolean expression can take any form, but must be a simple function or it must be enclosed in parentheses. If the result of the boolean is one (1) it is true, otherwise it is false.

Nested IF blocks are allowed. The maximum number of IF blocks in a script is fifty (50).

The following are examples of IF statements.

```
 ...
IF EXIST(B) THEN DISPLAY 'variable B has already been made'
IF (A>B) THEN DISPLAY 'A > B'
IF (A=B) THEN DISPLAY 'A = B'
IF (A<B) THEN DISPLAY 'A < B'
 ...
```

The following is an example of an IF block.

```
 ...
IF (A>B) THEN     ! an example of an IF block
   ...
ENDIF
 ...
```

The following is an example of IF blocks used with GOTO's.

```
!
J=5
START:
IF (J>8) THEN GOTO END
  ...
  J=J+1
  GOTO START
END:
J=5
START2:
IF (J<=8) THEN
  ...
  J=J+1
  GOTO START2
ENDIF
```

3.3 Instruction types

There are four types of PHYSICA instructions. See Table 3.11.

Operating system commands	input that starts with a dollar sign, $, for VMS input that starts with a percent sign, %, for UNIX
Assignments	input of the form `variable = expression`
Evaluations	input of the form `= expression`
Program commands	all other input

Table 3.11: Instruction types

Comments are allowed on assignment, evaluation and program command lines, but not on operating system commands. A comment begins with an exclamation mark, !, and continues to the end of the line. An input line can be simply a comment.

3.3.1 Operating system commands

VMS: Any input line that begins with a dollar sign, $, is considered to be a DCL command. These commands are executed by spawning a subprocess.

UNIX: Any input line that begins with a percent sign, %, is considered to be a UNIX command and are passed on to the shell.

3.3.2 Assignments

An assignment stores the value, or values, of an expression into an output variable. An assignment has the form

```
variable = expression
```

where expression is some combination of no more than 1500 literal constants, variables, functions, and operators. For example:

```
Y=A*COS(X)+3*SIN(X/6)+C*EXP(-X)
```

The output variable is the variable on the left side of the equal sign. The type of variable generated by an assignment is determined by the expression or by indices on the output variable.

If the output variable does not exist, it is created. If the output variable exists but its type is different than that indicated by the assignment, it is destroyed first and then recreated as the appropriate type. If the output variable already exists, and is of the right type, with dimension greater than or equal to the dimensions specified by an index, then just the necessary elements are changed. If the output variable exists, and is of the right type, with dimension less than that specified by an index, then the variable dimension will be expanded, and the new, unassigned, elements will be zero filled.

For example:

```
X=2               ! defines X to be a scalar
X=[1:10]          ! now X is a vector of length 10
X[6:15]=[3:12]*2  ! X[1:5] are unchanged, X now has length 15
X[2:7]=[-7:-2]    ! X still has length 15
X[10:20]=[1:5]    ! now X has length 20, elements 15 to 20 are zero
```

3.3.2.1 String variable assignments

A string assignment stores the value of a string into a string variable. A string assignment has the form:

```
variable = string
```

where string is some combination of literal strings, string variables, string functions, and string operators. A simple example of a string assignment: `T=‘ This is a string’` after which T has the value ‘ This is a string’.

A literal quote string is any set of characters enclosed in quotes, that is, an opening quote, ‘, and a closing quote, ’.

3.3.3 Evaluations

An evaluation is similar to an assignment, except that the value(s) of the expression are displayed on the terminal screen and not stored in variables. An evaluation has the form

```
=expression
```

For example: `=(6.5/44)*COSD(30)`

3.3.4 Comments

Any input line that begins with an exclamation mark, `!`, is considered to be a comment line. Comments are simply ignored. Comments can also be appended to the end of any input line, just start the comment with an exclamation mark.

If a comment line is read from a script file and echoing has been turned on with the `ENABLE ECHO` command, the comment will be displayed on the terminal screen.

3.3.5 Program commands

It is assumed that if an input line is not an operating system command, an assignment or an evaluation, then it must be a command line. In general, a command line consists of a command field, followed by one or more parameter fields. The maximum number of parameter fields is forty-nine (49).

The general form for a command line is:

```
command\qualifier...\qualifier p1\qualifier p2\qualifier ...
```

The parameter fields in a command line are separated by commas and/or blanks. If a field is enclosed by quotes, ‘ ’, or parentheses, ( ), then enclosed commas and blanks are considered to be a part of that field. For example, the following command lines:

```
command\qualifier p1 ‘a, test line’ p3
command\qualifier p1 (a, test line) p3
```

consists of four fields:

1. `command\qualifier`
2. `p1`
3. `a, test line`
4. `p3`

VMS: Whenever you are prompted for a command parameter, typing `control-z`, that is, simultaneously pressing the `control` key and the `z` key, aborts execution of that command, and of the script and any higher level calling scripts. If you type `control-space`, which is a null, only that command, not the script, will be aborted.

UNIX: Whenever you are prompted for a command parameter, typing `control-d` aborts execution of that command, and of the script and any higher level calling scripts. If you type `control-space`, which is a null, only that command, not the script, will be aborted.

To make use of a default value for a command parameter, indicate the position of that parameter by entering a null field. Two successive commas define a null field. For example, the command line: `command p1,,p3` has three parameter fields, with a second parameter field which is null.

3.3.5.1 Qualifiers

Command qualifiers are attached to a command and separated from the command, or from a preceding qualifier, by a backslash, \. No blanks are allowed in a qualifier.

A qualifier can be negated, if it makes sense to do so, by preceding the qualifier with a minus sign, or with `NO`. For example, `\COLOUR` can be negated with `\-COLOUR` or by `\NOCOLOUR`.

Some commands allow qualifiers on parameters. For example:

```
READ file.dat\[line1:line2] x\a y\b
```

4 VARIABLES

All numeric literals in PHYSICA are stored internally as double precision real numbers. Numeric valued variables are stored internally as double precision real numbers, but PHYSICA also allows string valued variables. Table 4.12 shows the variable types.

scalar	a single number
vector	a one dimensional list of numbers
matrix	a two dimensional array of numbers
string	a character string
string array	an array of strings

Table 4.12: Variable types

Except for some variables which are created automatically by various commands, each variable is named by the user.

4.1 Number and size of variables

Other than restrictions due to memory allocation on your computer, there is

- no maximum number of vectors, scalars or matrices
- no maximum length for vectors and no maximum size for matrices
- no maximum number of string variables
- no maximum dimension for string array variables
- no maximum length for string variables
- no maximum length for string elements of string array variables

4.2 Variable names

The first character of a variable name *must* be an alphabetic character, that is, `A` to `Z`. Except for this restriction, variable names can be any combination of:

alphabetic characters	`ABC...XYZ`	digits	`0123456789`
underscore	`_`	dollar sign	`$`

The maximum length of a variable name is thirty-two (32) characters.

All variable names are stored internally as upper case, but can be refered to as upper or lower case.

Function names and some other keywords are reserved names and cannot be used as variable names. See Table 4.13 for a list of the reserved keywords. See the PHYSICA Reference Manual for a list of the function names.

ALL	ARC	ARROW	AUTOHEIGHT	BOX
CAREA	CCONT	CIRCLE	CLOSE	COMMENSURATE
CVOLM	CXMIN	CXMAX	CYMIN	CYMAX
DAREA	DVOLM	DCONT	EDIT	ELLIPSE
FRAME	GRAPH	HEIGHT	IFF	LANDSCAPE
LIST	NSYMBOLS	OFF	ON	OPEN
PHYSICA	POLYGON	PORTRAIT	STATUS	SUB1
SUB2	SUB3	SUB4	SUB5	SUB6
SUB7	SUB8	TITLE	TRANSPARENCY	WEDGE
XAXIS	YAXIS			

Table 4.13: Reserved keywords

The `VARNAME` function accepts a variable, either string or numeric, as its argument, and converts that variable's name into a string. For example, `a=VARNAME(x)` would create a string variable called `A` with the value `'X'`.

4.3 Indices

Indices can be used on any numeric variable, except scalars, and on any string variable. A string variable is a string, and can have only one index, which is interpreted as a character index. A string array variable is an array of strings, and can have either one or two indices. If one index is used, it is

assumed to be an array index. If two sets of indices are used, the first is assumed to an array index and the second is assumed to be a character index. Indices can also be used on functions, and on expressions. An index can be a simple numeric literal or a complicated expression. Indices can also be nested.

4.3.1 Special characters

There are two special characters that are defined only within a variable index: # denotes the last element of a dimension, and * denotes the entire range of a dimension. The * character is not allowed on string variables. The # character can be used in an index expression, for example, X[#-1] is the next to last element of vector X. Refer to Table 4.14 on page 21 for some examples of these special index characters.

X[#]	is the last element of vector X
TS[#]	is the last character of string variable TS
TA[#]	is the last string of string array variable TA
TA[#][3:7]	is characters 3 to 7 of the last string of TA
M[#,2:10]	is elements 2 to 10 from the last row of matrix M
M[3:12,#-1]	is elements 3 to 12 from the next to last column of M
X[*]	is all of vector X
M[3,*]	is all of row 3 from matrix M
M[*,5]	is all of column 5 from M

Table 4.14: Index special character examples

and * cannot be used in indices on output variables, since they would be undefined. They also cannot be used in indices on functions or in indices on expressions.

4.3.2 Index on a function or expression

Indices can be used on any function, *except* for the array functions LOOP, SUM, PROD, RSUM, and RPROD. For example, if x and y are vectors of length 10, then sin(x+y) is also a vector of length 10, and an index can be used, such as sin(x+y)[3]. The special index characters # and * cannot be used in indices on functions.

Indices can be used on an expression. For example, if x and y are vectors of length 10, then (x+y) is also a vector of length 10, and an index can be used, such as (x+y)[3]. The special index characters # and * cannot be used in indices on expressions.

4.3.3 Index as an expression

An index can be any expression that results in a scalar or a vector. The index expression is evaluated first, and the resultant values are truncated to integers. For example, suppose x = [1;2;3;...;100], then:

```
=x[1:10:2]          ! displays   1  3  5  7  9
=x[2.1;2.5;2.9]     ! displays   2  2  2
y=[2:5]             !   define Y to be   2  3  4  5
=x[y/2+3]           ! displays   4  4  5  5
z=[2;3]             !   define Z to be   2  3
=x[y[z+1]-2]        ! displays   2  3
```

4.3.4 Index starting value

By default, a variable index starts at one (1), but you can define variable indices to start at any integer value, positive or negative. For example:

```
x = [1:100]         ! define a vector X to be the numbers 1;2;3;...;100
x[-5:3]=x[1:9]      ! redefine X to have a starting index of -5
SHOW x              ! display information on vector X
-------------------------------------------------------------------------------
vector indices     length type history
-------------------------------------------------------------------------------
     X [ -5:100]       106      X[-5:3]=[1:9]
y=[1:106]           ! make vector Y with same length as X, starting at 1
z=x+y
SHOW\VECTORS        ! display information on all vectors
-------------------------------------------------------------------------------
vector indices     length type history
-------------------------------------------------------------------------------
     X [ -5:100 ]      106      X[-5:3]=[1:9]
     Y [ 1:106 ]       106      Y=[1:106]
     Z [ 1:106 ]       106      Z=X+Y
```

The FIRST function returns a scalar value equal to the starting index of a vector, while the LAST function returns the final index of a vector. The LEN function returns a scalar value equal to the total length of a vector. The VLEN function returns a vector whose n^{th} element is the length of the n^{th} dimension of its argument. VLEN of a vector returns a vector of length 1, while VLEN of a matrix returns a vector of length 2.

Note: The CLEN function returns a scalar value equal to the length of a string. The TLEN command gives the number of string elements in a string array.

4.4 Scalars

A scalar is a single valued double precision real numeric variable. A scalar can be used wherever a single numeric value is expected. Indices have no meaning, and so are not allowed on scalars.

A literal scalar can be a single number, such as, 3.456, or an expression that results in a single number, such as LEN(x)+2.

Table 4.15 shows the possible ways that variables can be considered to be equivalent to scalars, that is, can be used wherever scalars are expected.

Let a and b be scalars Suppose that M is a matrix, V is a vector, and S is a scalar		
S	=	the value of S
V[a]	=	the value stored in the a^{th} element of V
M[a,b]	=	the value stored in row a and column b of M

Table 4.15: Scalar variable equivalents

4.5 Vectors

A vector is a one dimension array of double precision real numbers. A vector can be thought of as a list of numbers. There is no maximum length for vectors.

A literal vector can be a list of numbers, such as, [3;4.2;.456;-8], or a range of numbers, such as, [3:21:2], or an expression that results in a list of numbers, such as 3*[2:5]^2. Elements of a list are separated by semicolons, ;, while the colon, :, is used as the range element separator.

Table 4.16 shows the possible ways that variables can be considered to be equivalent to vectors, that is, can be used wherever vectors are expected.

Let a be a scalar and let x be a vector. Suppose that M is a matrix and V is a vector.			
V	=	V[i]	for i = 1, ..., LEN(V)
V[x]	=	V[i]	for i = x[i], x[2], ..., x[#]
M[x,a]	=	M[i,a]	for i = x[i], x[2], ..., x[#]
M[a,x]	=	M[a,j]	for j = x[i], x[2], ..., x[#]

Table 4.16: Vector variable equivalents

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.

4.6 Matrices

A matrix is a two dimensional array of double precision real numbers, with rows and columns. The row and column indices of a matrix are separated with a comma. The row dimension is specified first. There is no maximum size for matrices.

A literal matrix can be a list of vectors, such as, [[1;2;3];[4;5;6];[7;8;9]], or an expression that

results in a matrix, such as [2:5]><[2:6]. Elements of a list are separated by semicolons, ;.

Table 4.17 shows the possible ways that variables can be considered to be equivalent to matrices, that is, can be used wherever matrices are expected.

Let x and y be vectors Suppose that M is a matrix.		
M	=	M[i,j] for i = 1, ..., VLEN(M)(1), j = 1, ..., VLEN(M)(2)
M[x,y]	=	M[i,j] for i = x[i], x[2], ..., x[#], j = y[i], y[2], ..., y[#]

Table 4.17: Matrix variable equivalents

4.7 String variables

String variables can be used wherever strings are expected, such as file names and keyword parameters.

A string is defined to be a one dimensional array of ASCII characters. A string can be a literal quote string, such as, ‘this is a quote string’, or an expression that results in a string, such as RCHAR(35.6).

A literal quote string must begin with an opening quote, ‘ and end with a closing quote, ’.

A string variable is a single string, that is, a one dimensional array of characters. A string array variable is an array of strings. An element of a string variable is a single character. An element of a string array variable is a string. There is no maximum length for either a string variable or any element of a string array variable, nor a maximum number of elements of a string array variable. The elements of a string array variable need not be the same length.

The CLEN function returns a scalar value equal to the length of a string. The TLEN command gives the number of string elements in a string array.

A string variable, or an element of a string array variable, can be entered directly by means of an assignment. For example:

```
TS=‘This is a string’    ! string variable
TA[3]=‘This is a string’ ! array string variable: third element
```

An entire string array variable can be created with the READ\TEXT command.

Commands that expect strings, such as the TEXT command, which draws a string, or the PLOTTEXT command, which expects a file name as a parameter, will only accept a single string. Remember, though, that a string can be a literal quote string, a string variable, *one* element of a string array variable, and/or some combination of string functions and string operators.

Table 4.18 shows all of the possible ways that a string variable can be considered to be equivalent to a single string, that is, can be used wherever a string is expected.

Let a be a scalar and let x be a vector
Suppose that TA is a string array variable and T is a string variable

T	=	T[i]	for i = 1, ..., CLEN(T)
TA[a]	=	TA[a][i]	for i = 1, ..., CLEN(T[a])
T[x]	=	T[i]	for i = x[i], x[2], ..., x[#]
TA[a][x]	=	TA[a][i]	for i = x[i], x[2], ..., x[#]

Table 4.18: String variable equivalents

4.7.1 Appending strings

Strings may be appended together using the append operator, //. For example, suppose that T is a string variable with the value 'this is a string'. You can make a new string variable using the assignment:

```
T2='start of new string '//T//' end of new string'
```

and T2 will have the value: 'start of new string this is a string end of new string'.

A variable name can be converted to a string by means of the VARNAME function. A scalar *value* can be converted to a string by means of the RCHAR function. For example, if A is a scalar with the value -1.234, and T is a string variable with the value ' units', then the assignment:

```
T2='The value of '//VARNAME(A)//' is '//RCHAR(A)//T
```

makes a string variable T2 with the value: 'The value of A is -1.234 units'.

A format string is allowed as the second argument of the RCHAR function. For example:

```
T2='The value of '//VARNAME(A)//' is '//RCHAR(A,'F4.1')//T
```

makes a string variable T2 with the value: 'The value of A is -1.2 units'.

For information on VARNAME, RCHAR and other string functions, refer to the PHYSICA Reference Manual.

4.7.2 Expression variables

String variables can be used in numeric expressions, as so called expression variables, to shorten or to simplify an expression. Parentheses around an expression variable are assumed during a 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. You can force numeric evaluation by

using the EVAL function. For example:

```
T='3+2'     ! define T to be a string variable
=T          ! the string '3+2' will be displayed
=EVAL(T)    ! the numeric value 5 will be displayed
```

The EXPAND function produces a string by parsing the input string and expanding any expression variables present in this string. If an expression variable, contained in the original string, also contains expression variables, 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.

As an example of expression variable use, consider the following set of instructions:

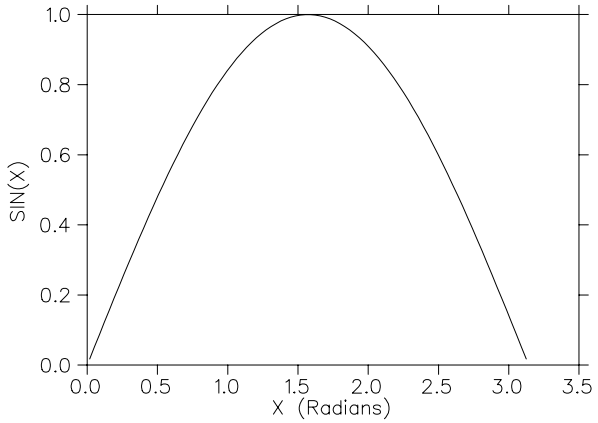
```
A=2               ! define a scalar A
B=3               ! define a scalar B
FC1='(A+B)/A'     ! define a string variable FC1
FC2='SQRT(A/B)'   ! define a string variable FC2
FC3='FC1*FC2'     ! define a string variable FC3
FC4='FC3+4*FC2'   ! define a string variable FC4
=FC4              ! displays 'FC3+4*FC2'
=EXPAND(FC4)      ! displays '(((A+B)/A)*(SQRT(A/B)))+4*(SQRT(A/B))'
=EVAL(FC4)        ! displays 5.307228
```

5 GRAPH EXAMPLES

5.1 Basics

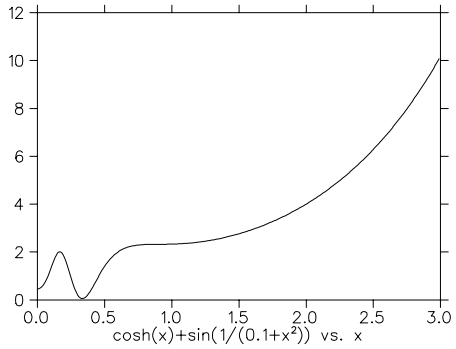
Let's start with a graph of vector Y versus vector X, using all the program defaults. First we generate some 'data', then set up the automatic axis labels and then draw the graph.

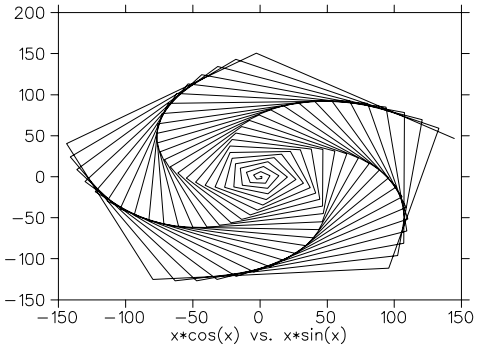
```
X = [1:180:2]*PI/180
Y = SIN(X)
LABEL\X 'X (Radians)'
LABEL\Y 'SIN(X)'
GRAPH X Y
```



Now consider the following commands, which produce two graphs in seperate windows.

```
WINDOW 5
LABEL\XAXIS 'cosh(x)+sin(1/(.1+x<^>2<_>)) vs. x'
X=[0:3:.01]
GRAPH X COSH(X)+SIN(1/(.1+X^2))
WINDOW 7
LABEL\XAXIS 'x*cos(x)  vs.  x*sin(x)'
X=[0:19*PI:.5]*2.55555
GRAPH SIN(X)*X COS(X)*X
```





Graph Examples

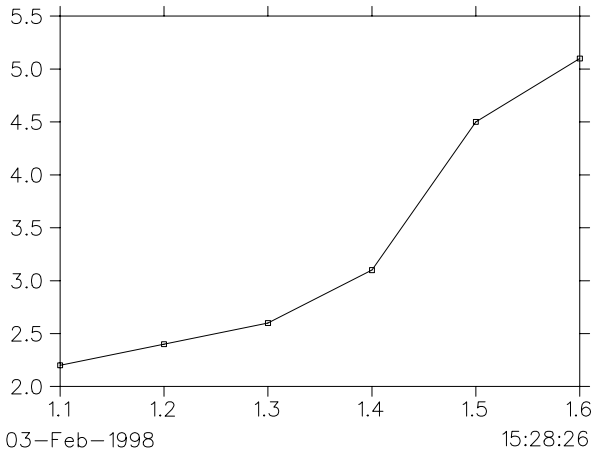
5.2 A script to plot a curve with axes

The script file, graph.pcm, listed below, reads two columns of numbers, in free format, from a file. The name of the file is passed to the script via the generalized parameter, ?1. The first column is stored in a vector called X and the second column is stored in a vector called Y. The script then produces an autoscaled graph of Y versus X using joined plotting symbols. Plotting symbol number one, a 'box', is used. The plot is then labeled with the date and time, positioned interactively. Note that comments begin with a !.

```
! script file   graph.pcm
!
READ ?1 X Y    ! read two columns of numbers into vectors X and Y
SET PCHAR 1    ! plotting symbol #1, joined
GRAPH X Y      ! plot the data, autoscaled, with axes
TEXT DATE      ! draw the current date using the graphics cursor
TEXT TIME      ! draw the current time using the graphics cursor
```

The figure below was produced by entering the command: `@graph file.dat` The data file, with two columns of numbers, is also shown below.

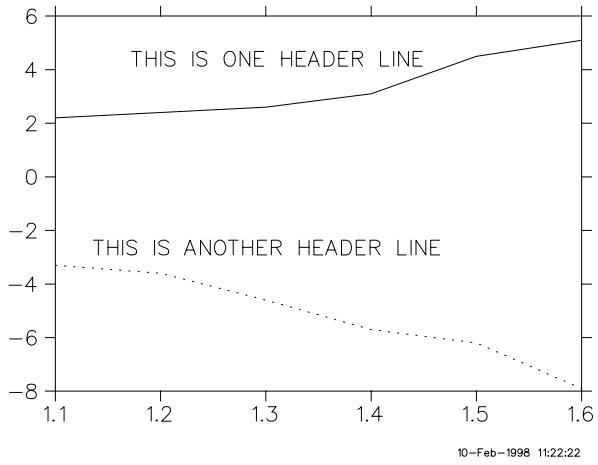
```
1.1  2.2
1.2  2.4
1.3  2.6
1.4  3.1
1.5  4.5
1.6  5.1
```



5.3 Read some data and plot two curves on common axes

The script file, graph.pcm, listed below, reads three columns of numbers from a file. The filename is passed to the script via a generalized parameter, ?1. The first column is stored in a vector called X, the second column is stored in a vector called Y1, and the third column is stored in a vector called Y2. The script then produces a graph of Y1 versus X and overlays a curve of Y2 versus X. It then replots both curves on a common scale. Identifying text is read into a string array variable called TXT and located on the plot with the graphics cursor. The date and time are also drawn on the plot, but the position and justification are pre-set, so the graphics cursor is not used. The figure below was produced by entering the command: `@graph file.dat` The data file, with three columns of numbers and two lines of header is also shown below.

```
THIS IS ONE HEADER LINE
THIS IS ANOTHER HEADER LINE
1.1  2.2  -3.3
1.2  2.4  -3.6
1.3  2.6  -4.6
1.4  3.1  -5.7
1.5  4.5  -6.2
1.6  5.1  -7.9
```

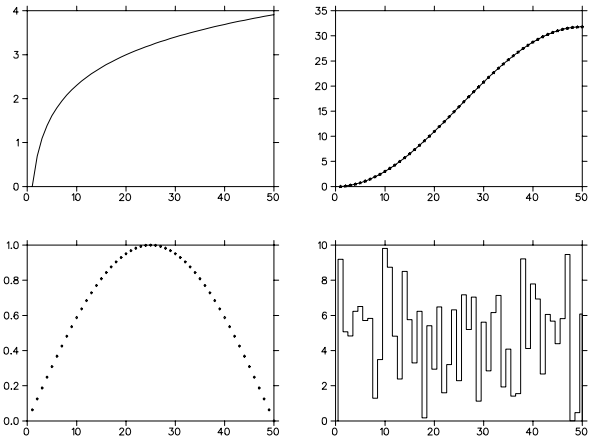



```
! script file  graph.pcm
!
READ\TEXT\CONTINUE ?1\[1:2] TXT ! read string array variable from lines 1 and 2
                                 !   and don't close the file after
READ ?1 X Y1 Y2                  ! read vector data (starting in line 3)
GRAPH X Y1                       ! plot data with axes
SET LINTYP 10                    ! change line type
GRAPH\-AXES X Y2                 ! overlay a curve
REPLOT                           ! replot on common scale
TEXT TXT[1]                      ! draw the first text string
TEXT TXT[2]                      ! now the second string
SET
  %TXTHIT 1.5                    ! set text height to 1.5% of the window height
  %XLOC 95                       ! set text x location to 95% of window width
  %YLOC 2                        ! set text y location to 95% of window height
  CURSOR -3                      ! set text justification to right justification
                                 ! blank line to finish SET command
TEXT DATE//' '//TIME             ! plot the date and time, not positioned interactively
DEFAULTS                         ! reset program defaults
```

Graph Examples

5.4 Using four windows to draw four graphs

The following set of commands will draw $\log(x)$ versus x with no plotting symbol, the data points joined by straight line segments, and on the same page but in other windows, $sin(x)$ versus x with unjoined 'diamond' symbols, $\int \sin(x)$ versus x with joined 'star' symbols, and finally a histogram of some randomly generated 'data'.



```
X = [1:50]                          ! fake some 'data'
WINDOW 5                            ! choose a physica window
SET PCHAR 0                         ! no plotting symbol
GRAPH X LOG(X)                      !
WINDOW 6                            !
SET PCHAR -5                        ! unjoined 'diamonds'
GRAPH X SIN(X/50*PI)                !
WINDOW 7                            !
SET PCHAR 14                        ! joined 'stars'
GRAPH X INTEGRAL(X,SIN(X/50*PI))    !
WINDOW 8                            !
SET HISTYP 1                        ! choose histogram type
GRAPH X RAN(X)*10                   !
```

5.5 Getting the axis numbers right

Suppose you want a y-axis number range from 0 to 0.0008 and an x-axis number range from -40 to 30. You set the axis scales with the commands below and by default you will get the left axis box in Figure 5.4.

```
SCALE -40 30 0 .0008  ! x-axis: -40 to 30,  y-axis: 0 to 0.0008
GRAPH\AXESONLY        ! only plot the axes
```

But suppose you want the y-axis numbers to be of the form $n.0$ with something like $(\times 10^{-m})$ included as part of the y-axis label. Then you could enter the commands below to produce the axis box on the right in Figure 5.4.

```
SET
```

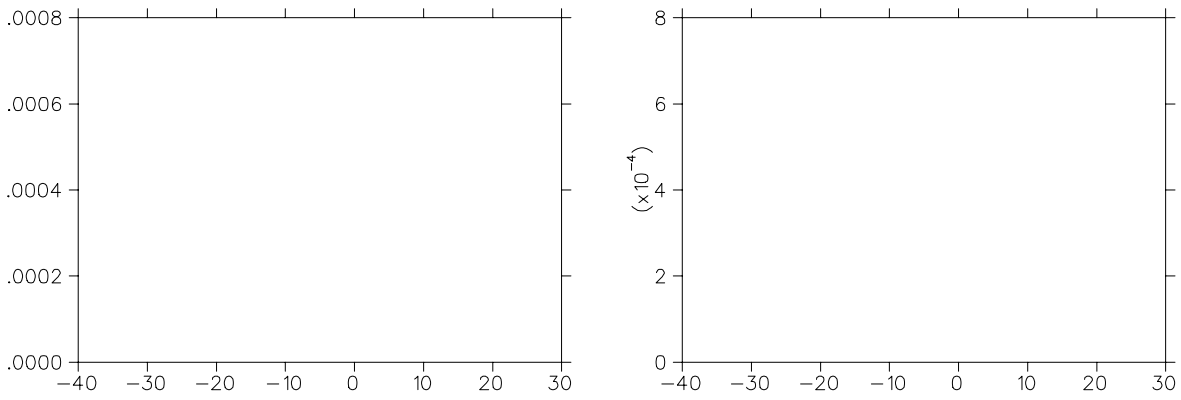


Figure 5.4: Numbering the axes

```
  YPAUTO 0              ! do not automatically calculate the y-axis power
  YPOW -4               ! set the y-axis power
  NYDEC 1               ! number of decimal places for y-axis numbers
  NYDIG 3               ! number of digits for y-axis numbers
                        ! blank line finishes the SET command
SCALE -40 30 0 .0008    ! set the axis scales after the numbering format
GRAPH\AXESONLY          ! just plot the axes
```

5.6 Plotting symbols

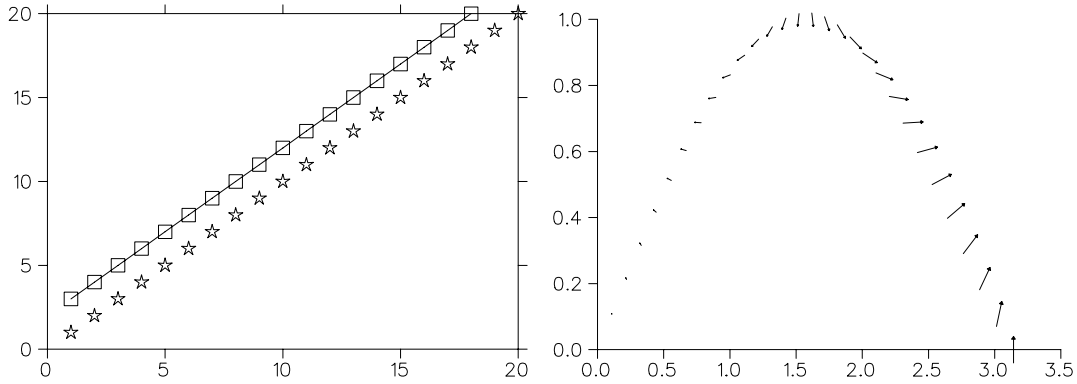
To illustrate one way to set up plotting symbols for a graph, consider the following commands, which produce the left graph in Figure 5.5 on page 32.

```
SET %CHARSZ 3       ! make the plotting symbols bigger
SET PCHAR -14       ! plotting symbols to be unjoined ‘star’s
X=[1:20]            ! create vector X = {1;2;3;...;20}
GRAPH X X           ! draw the graph
SET PCHAR 1         ! plotting symbols to be joined ‘box’s
GRAPH\-AXES X X+2   ! overlay a curve
```

5.6.1 Size and angle

To illustrate how to control the size and angle of the plotting symbols, consider the following commands, which produce the graph on the right in Figure 5.5 on page 32.

```
GENERATE X 0,,PI 30         ! make some ’data’
GENERATE SIZE 0,,2 30       ! symbol size vector
GENERATE ANGLE 0,,360 30    ! symbol angle vector
```



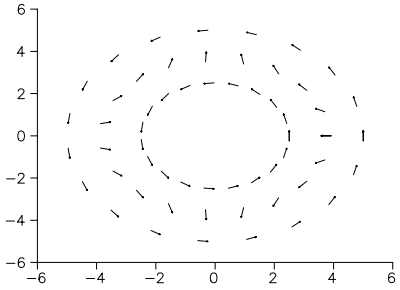
(a) Simple setup (b) Size and angle

Figure 5.5: Plotting symbols

```
SET PCHAR -18 SIZE 1 ANGLE  ! 'centred arrow' symbols, unjoined, all colour 1
SET BOX 0                   ! turn off graph box
GRAPH X SIN(X)              ! plot sin(x) vs. x
```

5.6.2 Plotting a vector field

The following commands illustrate one way to plot a vector field, where the angle of the plotting symbols conveys information.




```
GENERATE A 0,,360 20      ! create vector A: min= 0, max= 360, length= 20
GENERATE R 5,,5 20        ! create vector R: min= 5, max= 5, length= 20
SET PCHAR -18 1 1 A       ! symbol 'centered arrow', size 1, colour 1, angle A
GRAPH\POLAR R A           ! draw graph in polar coordinates
GRAPH\-AXES\POLAR R/2 A   ! overlay curve in polar coordinates
SET PCHAR -18 1 1 90-A    ! symbol 'centered arrow', size 1, colour 1, angle 90-A
GRAPH\-AXES\POLAR 3*R/4 A ! overlay curve
```

5.7 A graph with two y axes

This example illustrates one way to display two different y axes on a single graph. See Figure 5.6. Suppose we have three vectors, a, b, and frequency and we want to plot a and b versus frequency on one set of axis scales, and then plot some function of a and b versus frequency in the same axis box, but with a different y-axis scale on the right. We also want to label the curves and the axes. *Note*: The REPLOT command does not work with this type of constructed graph.

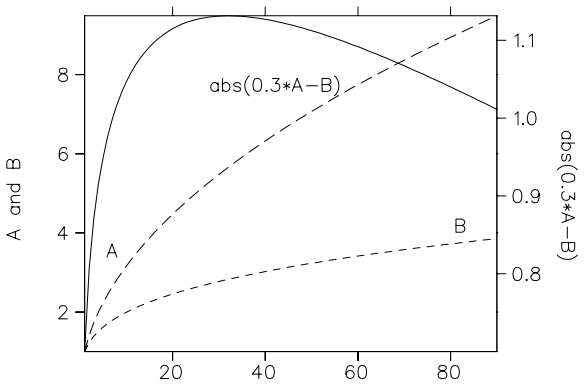


Figure 5.6: A graph with two y axes

Start by generating some "data" to plot, and then executing the 2yaxes.pcm script, passing the newly created vectors as parameters. This script allows you to try out any function of a and b.

```
frequency=[1:90]
a=sqrt(frequenc)
b=frequency^0.3
@2yaxes frequency a b 'abs(0.3*A-B)'
```

```
! this is the  2yaxes.pcm  script
!
xtemp = ?1    ! the x variable
ytemp = ?2    ! the first y variable
ztemp = ?3    ! the second y variable
fnc   = ?4    ! an expression (string) variable
set
 rittic 0     ! turn off right side tics
 lintyp 3     ! choose a line type
%xuaxis 85    ! move the right side in for numbers and label
 colour 1     !

scale min(xtemp) max(xtemp) min([ytemp;ztemp]) max([ytemp;ztemp])
graph xtemp ytemp
set lintyp 5
graph\-axes xtemp ztemp
!
lenf = len(xtemp)
world\percent xtemp[10] ytemp[10] xloc1 yloc1         ! for labeling the curves
world\percent xtemp[lenf-5] ztemp[lenf-5] xloc2 yloc2 !
set
```

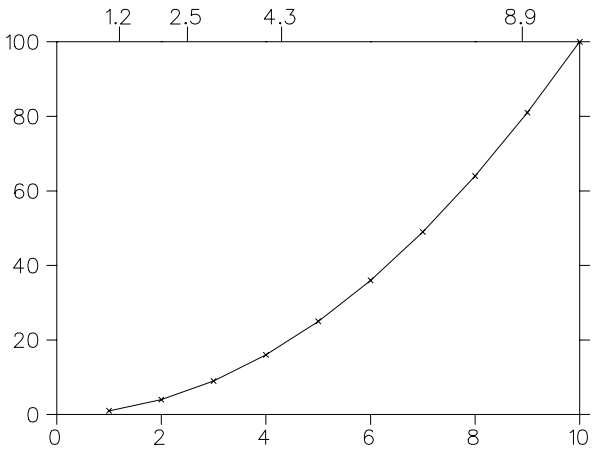
```
 %xloc xloc1-2
 %yloc yloc1+2
 cursor -2

text varname(?2)
set
 %xloc xloc2-2
 %yloc yloc2+2
 cursor -2

text varname(?3)
get
 %xuaxis xux
 %ylaxis ylx
 %yuaxis yux
 %yiticl yiticl

set
 xaxis 0      ! turn off both axes
 yaxis 0      !
 box 0        ! turn off the box
 lintyp 1     !
 colour 2     !

scale min(xtemp) max(xtemp) min(fnc) max(fnc)
graph xtemp fnc
world\percent xtemp[lenf/2] (fnc)[lenf/2] xloc1 yloc1
set
 %xloc xloc1-2
 %yloc yloc1-15
 cursor -2

text '<c2>'//fnc          ! label the curve
set
   yitica -90             ! rotate the yaxis numbers
   ytica -90              ! rotate the yaxis tics
  %yiticl yiticl*1.2      ! move the numbers out a bit
  %xlaxis xux             ! move the y-axis to the right side
   yaxis 1                ! only turn on the y-axis

graph\axesonly
set
 %xloc 4
 %yloc (yux+ylx)/2
  cursor -10
  txtang 90
```

```
text ‘<c1>’//varname(?2)//‘ and ’//varname(?3) ! label the axes
set
 %xloc 96
 %yloc (yux+ylx)/2
  cursor -10
  txtang -90

text ‘<c2>’//fnc
destroy xtemp ytemp ztemp xloc1 yloc1 xloc2 yloc2 fnc
defaults
```

5.8 Two x axes

This example illustrates one way to add tic marks at arbitrary user-defined locations on a second x-axis. See the Figure opposite. First, generate some “data” and create a vector with the locations of the tic marks, in graph units.

Then the data is plotted, followed by the conversion of the tic mark locations into world units so the locations of the numbers to be drawn above the tic marks can be determined. The trick here is to plot the tic marks as if they were data.

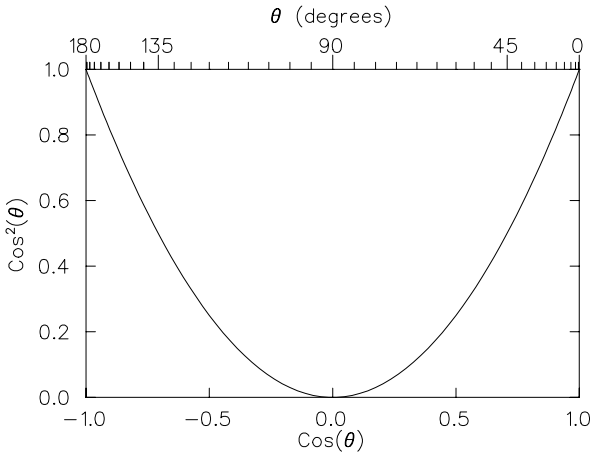



```
x   = [1:10]              ! fake some data
y   = x^2                 !
x2  = [1.2;2.5;4.3;8.9]   ! locations for tic marks (in graph units)
lx2 = len(x2)             ! length of vector x2
y2  = [1:lx2]             ! y2 = [1;2;3;4;...]
set
 toptic 0                 ! turns off automatic tic marks on top axis
 pchar 2                  ! plotting character 2 (connected)
 autoscale on             ! autoscale the plot

graph x y                 ! graph the data with numbers on bottom axis
get
 %xnumsz xnumsz           ! x-axis number size (% of window)
 %ylaxis ylaxis           ! location of bottom of y-axis (% of window)
 %yuaxis yuaxis           ! location of top of y-axis (% of window)
 %xiticl xiticl           ! distance from x-axis to numbers (% of window)
  ylaxis ylaxisw          ! location of bottom of y-axis (world units)
  yuaxis yuaxisw          ! location of top of y-axis (world units)
```

```
  xiticl xiticlw           ! distance from x-axis to numbers (world units)
  ymin   ymin              ! number at bottom of y-axis
  ymax   ymax              ! number at top of y-axis

!
!  figure out where to plot the tic marks
!
y1 = xiticlw/(yuaxisw-ylaxisw)*(ymax-ymin)+ymax
yloc = yuaxis+1.2*xiticl   ! y location for plotting the numbers
set
 %txthit xnumsz            ! text height (% of window)
  cursor -2                ! centre justification
 %yloc yloc                ! y location for text (% of window)
  txtang 0                 ! text angle
  clip 0                   ! allow plotting outside of axis box
  pchar 0                  ! no plotting symbol

world\percent x2 y2 xp yp  ! convert graph units to % world coordinates
do j = [1:lx2]             ! for each large tic mark
  set %xloc xp[j]
  text rchar(x2[j])                    ! draw the upper x-axis numbers
  graph\-axes [x2[j];x2[j]] [ymax;y1]  ! plot the tic marks on top of box
enddo
```

5.8.1 Non-linear user-defined axis

This example illustrates one way to have two x axes on the same plot. The goal here is to plot one function versus another function, instead of a function versus an independent variable, and to show functional values on the bottom x-axis and the original "independent" variable on the upper x-axis. Of course, the upper axis will be non-linear. See the figure opposite.



In this case, suppose $x = \text{cosd}(\theta)$ and $y = x^2$. We want to plot y versus x, with the bottom x-axis scaled to the values of the cosd function, and the upper x-axis scaled to the values of θ. First, we generate some 'data'.

```
generate theta 0,,180 100      ! make a vector with 100 elements from 0 to 180
x = cosd(theta)
y = x^2
x2 = [0;45;90;135;180]         ! the upper axis large tic mark locations
```

```
x3 = [5;10;15;20;25;30;35;40]  ! the upper axis small tic mark increments
```

The trick here is that the upper x-axis tic marks are plotted as if they were data, in relation to the lower x-axis scales.

```
lx2 = len(x2)          ! length of vector x2 = number of large tic marks
y2 = [1:lx2]           ! y2 = [1;2;3;4;...]
xupper = cosd(x2)      ! convert to functional values
lx3 = len(x3)          !
y3 = [1:lx3]           !
set
 %yuaxis 85            ! make room for a label
  toptic 0             ! turn off automatic tic marks on top axis
  pchar 0              ! no plotting symbol
  autoscale on         ! autoscale the plot
  xtica 90             ! tic marks to point inside the axis box
  ytica -90            !

label\xaxis 'Cos(<theta>)'
label\yaxis 'Cos<^>2<_>(<theta>)'
graph x y              ! draw the graph with the bottom x-axis
get
 %xnumsz xnumsz        ! number size on x-axis
 %ylaxis ylaxis        ! lower end of y-axis (as % of window)
 %yuaxis yuaxis        ! upper end of y-axis (as % of window)
 %xiticl xiticl        ! distance from x-axis to top of numbers (as %)
  ylaxis ylaxisw       ! lower end of y-axis (world units)
  yuaxis yuaxisw       ! upper end of y-axis (world units)
  xiticl xiticlw       ! distance from x-axis to top of numbers (world units)
  ymin   ymin          ! number at bottom of y-axis
  ymax   ymax          ! number at top of y-axis

ticlen = xiticlw/(yuaxisw-ylaxisw)*(ymax-ymin)
y1 = ymax+ticlen
y11 = ymax+0.5*ticlen
yloc = yuaxis+1.2*xiticl
set
 %txthit xnumsz        ! height of text (% of window height)
  cursor -2            ! centre justify text
 %yloc yloc            ! vertical location of text
  txtang 0             ! text angle (degrees)
  clip 0               ! allow plotting outside of axis box

world\percent xupper y2 xp yp ! convert graph units to % of window
do j = [1:lx2]                ! for each large tic mark ...
```

```
  set %xloc xp[j]             ! set horizontal location of text
  text rchar(x2[j])           ! draw the upper x-axis numbers
  graph\-axes [xupper[j];xupper[j]] [ymax;y1]  ! draw large tic marks
  xtmp = cosd(x2[j]+x3)                       !   on top of box
  world\percent xtmp y3 xpp ypp
  do k = [1:lx3]
    graph\-axes [xtmp[k];xtmp[k]] [ymax;y11]   ! draw small tic marks
  enddo                                       !   on top of box
enddo
get                    ! find the mid point of the x-axis
 %xuaxis xux
 %xlaxis xlx

set
 %xloc (xux+xlx)/2     ! position a label at the mid point
 %yloc 95
  cursor -2            ! centre justify

text '<theta> (degrees)'
defaults
```

5.9 Using two adjoined axis frames

The following example script file produces a drawing for inclusion into a publication. Figure 5.7 on page 39 illustrates the drawing that results from execution of this script. The data for this figure is not included here.

```
CLEAR                      ! clear graphics
ORIENTATION PORTRAIT       ! choose orientation
SCALE -10 40 5 0.0 8 8     ! set axis scales
WINDOW 12 5 0 100 100      ! define and use a window
FILE='afig.dat'            ! define a file name
READ FILE//'1' E Z0 Z6\4   ! read data from afig.dat1
READ FILE//'2'\10 EK K0 K5 ! read afig.dat2 starting on line 10
READ FILE//'3'\7 EW W6\4   ! read afig.dat3 starting on line 7
READ FILE//'4'\9 EB A B    ! read afig.dat4 starting on line 9
!
! following are needed for smoothing
GENERATE EKS MIN(EK),,MAX(EK) 100
GENERATE EWS MIN(EW),,MAX(EW) 100
GENERATE EBS MIN(EB),,MAX(EB) 100
!
! automatic axis labels
LABEL\X 'Excitation Energy (MeV)'
LABEL\Y  '<V4%>d<^>2<_,sigma>/d<OMEGA>/dE (mb/sr/MeV)'
```

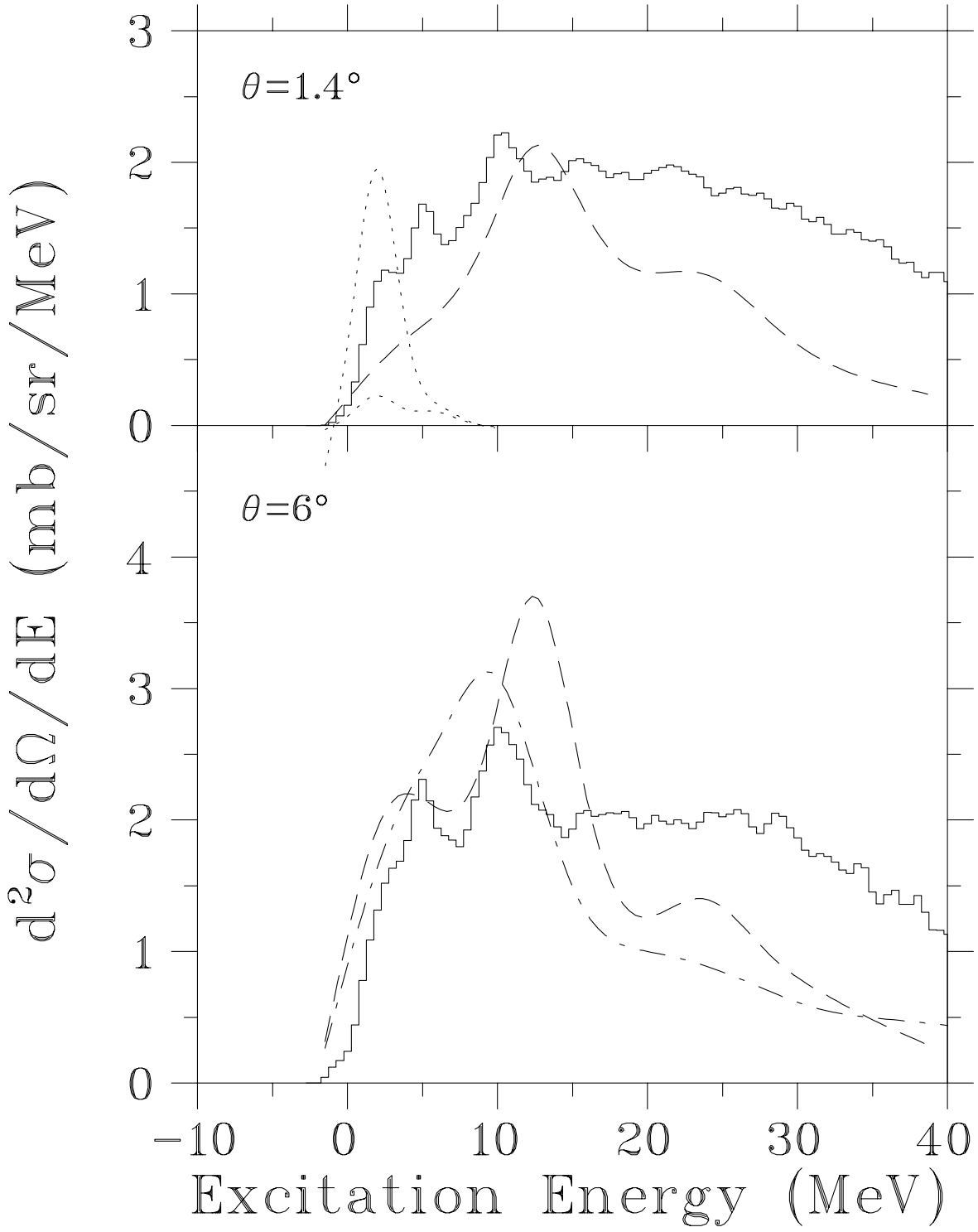


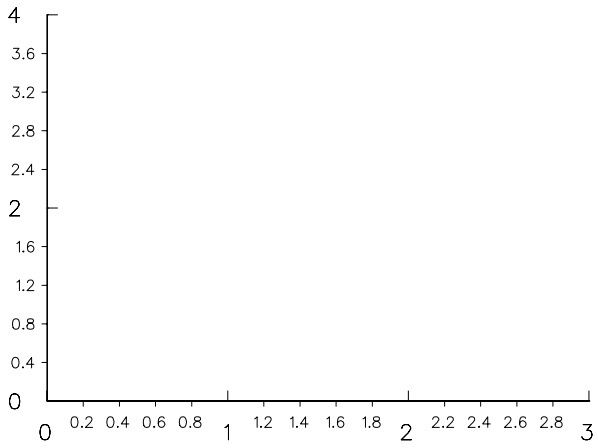
Figure 5.7: Two adjoined axis frames

```
!
! lower frame of the figure
SET
  FONT ROMAN.3            ! select text font
 %YUAXIS 96               ! y-axis upper edge
 %YNUMSZ 2.5              ! y-axis number size
 %XNUMSZ 2.5              ! x-axis number size
  YMOD 5                  ! y-axis numbers modulo 5
  NSYINC 2                ! number of short y-axis tics
  LINTYP 3                ! line type
  NSXINC 2                ! number of short x-axis tics
  COLOUR 1
  TENSION 0.1             ! spline tension for smoothing
  LINTYP 3                ! set line type for data curve

GRAPH\AXESONLY            ! just draw axes
! plot smoothed Klein 5 deg csx, long dashes
COLOUR BLUE               ! choose colour
GRAPH\-AXES EKS SMOOTH(EK,K5,EKS) ! overlay smooth curve
! plot smoothed Wambach 6 deg csx, dash-dot
SET LINTYP 4              ! set line type
COLOUR RED                ! choose colour
GRAPH\-AXES EWS SMOOTH(EW,W6,EWS) ! overlay smooth curve
! plot the 6 deg data, histogram form
SET
 HISTYP 1                 ! histogram without tails
 LINTYP 1                 ! line type

COLOUR MAGENTA            ! choose colour
GRAPH\-AXES E Z6          ! overlay data curve
!
! now set up the upper frame of the figure
LABEL\X                   ! blank out x-axis label
SET
  BOX 0                   ! turn off axis box
 %XNUMSZ 0                ! x-axis number size
  YAXIS 0                 ! turn off y axis
  YCROSS 1                ! force axis crossing
  YMIN -5                 ! y-axis minimum
  YMAX 3                  ! y-axis maximum
  COLOUR 1

GRAPH\AXESONLY            ! just draw axes
! draw the 0 deg data
COLOUR RED                ! choose colour
GRAPH\-AXES E Z0          ! overlay data curve
```

```
SET
  BOX 1                   ! turn axis box on
  HISTYP 0                ! turn off histogramming
  LINTYP 3                ! line type
  YAXIS 1                 ! turn on y-axis
  YCROSS 0                ! do not force axis crossing
 %XLOC   19.9             ! text x location
 %YLOC   90.8             ! text y location
 %TXTHIT 2                ! text height
  TXTANG 0                ! text angle
  CURSOR -1               ! text justification

COLOUR BLUE               !
TEXT ‘<theta>=1.4<X>A1<X>’
! plot Klein 0 deg csx with dashed line
SET LINTYP 3              ! set line type
COLOUR GREEN              !
GRAPH\-AXES EKS SMOOTH(EK,K0,EKS)
! plot Bloom 0 deg csx with dotted line
SET LINTYP 9              ! set line type
SET COLOUR 1              !
GRAPH\-AXES EBS SMOOTH(EB,A,EBS)
COLOUR MAGENTA            ! choose colour
GRAPH\-AXES EBS SMOOTH(EB,B,EBS) ! overlay smooth curve
SET
 %XLOC   19.9             ! text x location
 %YLOC   58.3             ! text y location
 %TXTHIT 2                ! text height
  TXTANG 0                ! text angle
  CURSOR -1               ! text justification

COLOUR BLUE               ! choose colour
TEXT ‘<theta>=6<X>A1<X>’
DEFAULTS                  ! reset program defaults
```

5.10 Numbering the small tic marks

The script below shows one way to number the small tic marks on either the x or y-axis, producing the figure on the right.

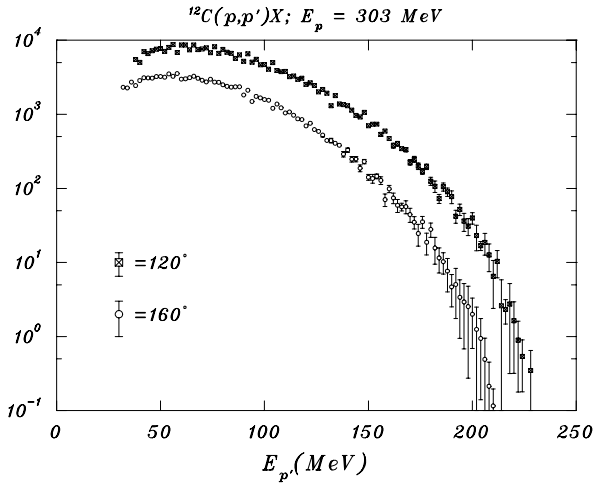



```
SCALE 0 3 3 0 4 2        ! axis scales
GET                      !
  XTICS XSHORT           ! length of the short x-axis tic marks
  XTICL XLONG            ! length of the long x-axis tic marks
  YTICS YSHORT           ! length of the short y-axis tic marks
  YTICL YLONG            ! length of the long y-axis tic marks
 %YITICL YITICL          ! distance from axis to y-axis numbers
 %XITICL XITICL          ! distance from axis to x-axis numbers
                         ! blank line finishes GET command
SET                      !
  BOX 0                  ! turn off axis box
  NSXINC 5               ! number of small x-axis tic marks
  NSYINC 5               ! number of small y-axis tic marks
  YTICL 0                ! length of long tic marks on y-axis
  XTICL 0                ! length of long tic marks on x-axis
 %YITICL 1.5*YITICL      ! distance from y-axis to numbers
 %XITICL 1.5*XITICL      ! distance from x-axis to numbers
                         ! blank line finishes SET command
GRAPH\AXESONLY           ! draw axes
SET                      !
  NSXINC 0               ! turn off small tic marks on x-axis
  NSYINC 0               ! turn off small tic marks on y-axis
  YTICA -90              ! flip tic marks on y-axis
  XTICA 90               ! flip tic marks on x-axis
  YTICL YLONG            ! reset the y-axis long tic mark length
  XTICL XLONG            ! reset the x-axis long tic mark length
                         ! blank line finishes SET command
GRAPH\AXESONLY           ! draw axes
GET                      !
 %XLAXIS XLAXIS          ! location of left end of x-axis
 %YNUMSZ YNUMSZ          ! size of y-axis numbers
```

```
 %YLAXIS YLAXIS          ! location of bottom end of y-axis
 %XNUMSZ XNUMSZ          ! size of x-axis numbers
                         ! blank line finishes GET command
YNUMS = [0:4:.4]         ! the numbers to place on the y-axis
XNUMS[1:LEN(YNUMS)] = 0  ! the x coordinates of these numbers
WORLD\PERCENT XNUMS YNUMS XP YP  ! convert to world units %
SET                      !
 %XLOC XLAXIS-0.5*YITICL ! x location for the numbers
 %TXTHIT 0.7*YNUMSZ      ! text height for the numbers
 CURSOR -11              ! justify right centre
                         ! blank line finishes SET command
DO J = [2:5]             !
  SET %YLOC YP[J]        ! y locations for numbers
  TEXT RCHAR(YNUMS[J])   ! draw the first set of numbers
ENDDO                    !
DO J = [7:10]            !
  SET %YLOC YP[J]        !
  TEXT RCHAR(YNUMS[J])   ! draw the second set of numbers
ENDDO                    !
! DESTROY XNUMS YNUMS
XNUMS = [0:3:.2]         ! the numbers to place on the x-axis
YNUMS[1:LEN(XNUMS)] = 0  ! the y coordinates of these numbers
WORLD\PERCENT XNUMS YNUMS XP YP  ! convert to world units %
SET                      !
 %YLOC YLAXIS-XITICL     ! y location for the numbers
 %TXTHIT 0.7*XNUMSZ      ! text height for the numbers
 CURSOR -10              ! justify upper centre
                         ! blank line finishes SET command
DO J = [2:5]             !
  SET %XLOC XP[J]        ! x locations for numbers
  TEXT RCHAR(XNUMS[J])   ! draw the first set of numbers
ENDDO                    !
DO J = [7:10]            !
  SET %XLOC XP[J]        !
  TEXT RCHAR(XNUMS[J])   ! draw the second set of numbers
ENDDO                    !
DO J = [12:15]           !
  SET %XLOC XP[J]        !
  TEXT RCHAR(XNUMS[J])   ! draw the third set of numbers
ENDDO                    !
DESTROY XNUMS YNUMS XSHORT XLONG YSHORT YLONG YITICL XITICL -
        XLAXIS YNUMSZ YLAXIS XNUMSZ XP YP
DEFAULTS                 ! reset defaults
```

5.11 Error bars

The following script produces the figure opposite, which illustrates the use of error bars.




```
READ xsec.dat E2 E3 X2 X3 -
      DX2 DX3 Y2 Y3 R2 R3
S12[1]  = 30
SY12[1] = 10
SY13[1] = 2
DY12[1] = 3.5
DY13[1] = 1.0
SCALE 0 250 5 -1 4 5
SET
  YLOG  10
 %CHARSZ .8
 %XNUMSZ 2.5
 %YNUMSZ 2.5
  XTICA  90
  YTICA -90
  FONT ITALIC.3
  PCHAR  -12
  COLOUR 1

GRAPH\AXESONLY E2 X2
COLOUR RED
GRAPH\-AXES E2 X2 DX2
SET %CHARSZ 1.5
GRAPH\-AXES S12 SY13 DY13
SET PCHAR  -3
SET %CHARSZ 0.8
COLOUR BLUE
GRAPH\-AXES E3 X3 DX3
SET %CHARSZ 1.5
GRAPH\-AXES S12 SY12 DY12
```

```
GET
 %YLAXIS YLAX
 %YUAXIS YUAX
 %XLAXIS XLAX
 %XUAXIS XUAX

SET
 %XLOC (XUAX+XLAX)/2
 %YLOC 3
  CURSOR -2
  COLOUR 1

 TEXT 'E<_>p'//CHAR(39)//'<^>(MeV)'
SET
 %XLOC -4
 %YLOC (YLAX+YUAX)/2
  CURSOR -6

TEXT 'd<sigma>/d<Omega>(nb/sr/MeV)'
SET
 %XLOC (XUAX+XLAX)/2
 %YLOC (100+YUAX)/2
 %TXTHIT 2.5
  CURSOR -9

COLOUR MAGENTA
TEXT '<^>12<_>C(p,p'//CHAR(39)//-
     ')X; E<_>p<^> = 303 MeV'
SET
 %XLOC 27
```

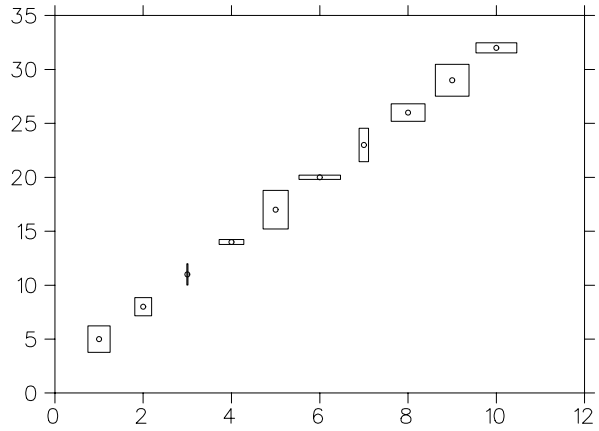
```
 %YLOC 45
  CURSOR -7

COLOUR BLUE
TEXT '=120<degree>'
```

```
SET %YLOC 34.5
COLOUR RED
TEXT '=160<degree>'
DEFAULTS
```

5.11.1 User defined error bars

The script below draws rectangular error bars, for y-errors, where the width of the error bar rectangle is controllable by the user. The following commands produced the figure on the right.

```
X = [1:10]
Y = 2+3*X
YERROR = 2*RAN(X)
XERROR = RAN(X)/2
SET PCHAR -12
GRAPH X Y
@EBARS X Y XERROR YERROR
REPLOT
```



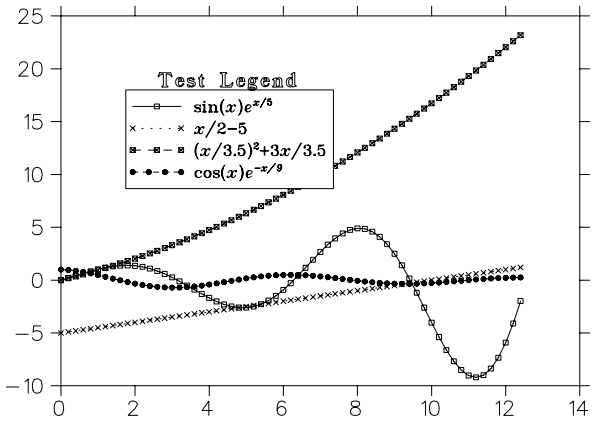

```
! This script produces rectangular error bars centered about the data
! points.  Execute it with:    @EBARS X Y DY DW
! where you would normally enter GRAPH X Y DY DW to produce a graph with
! errors given in DY and DW.
XTEMP  = ?1
YTEMP  = ?2
DYTEMP = ?3
DXTEMP = ?4
DO J = [1:LEN(XTEMP)]
  XTEMPX[1+(J-1)*5] = XTEMP[J]-DXTEMP[J]
  YTEMPY[1+(J-1)*5] = YTEMP[J]+DYTEMP[J]
  PC[1+(J-1)*5] = -16
  XTEMPX[2+(J-1)*5] = XTEMP[J]-DXTEMP[J]
  YTEMPY[2+(J-1)*5] = YTEMP[J]-DYTEMP[J]
  PC[2+(J-1)*5] = 16
  XTEMPX[3+(J-1)*5] = XTEMP[J]+DXTEMP[J]
  YTEMPY[3+(J-1)*5] = YTEMP[J]-DYTEMP[J]
  PC[3+(J-1)*5] = 16
  XTEMPX[4+(J-1)*5] = XTEMP[J]+DXTEMP[J]
  YTEMPY[4+(J-1)*5] = YTEMP[J]+DYTEMP[J]
  PC[4+(J-1)*5] = 16
  XTEMPX[5+(J-1)*5] = XTEMP[J]-DXTEMP[J]
```

```
  YTEMPY[5+(J-1)*5] = YTEMP[J]+DYTEMP[J]
  PC[5+(J-1)*5] = 16
ENDDO
! now we can produce our graph
SET PCHAR PC
GRAPH XTEMPX YTEMPY
DESTROY XTEMP YTEMP DYTEMP DXTEMP J PC XTEMPX YTEMPY
DEFAULTS                              ! reset program defaults
```

5.12 Graph legend

5.12.1 A non-transparent legend frame

This example shows how a multiple curve graph with legend could be done.

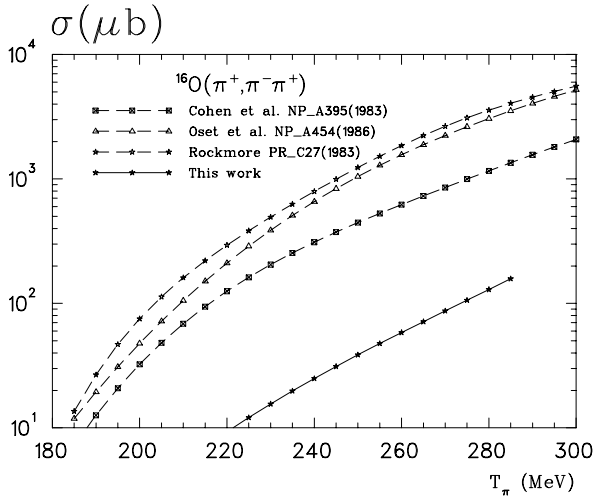



```
!
! first create the string portions of the legend entries
!
t1 = ‘<froman.3>sin(<fitalic.3>x<froman.3>)<fitalic.3>e-
<froman.3,^><fitalic.3>x<froman.3>/5’
t2 = ‘<fitalic.3>x<froman.3>/2-5’
t3 = ‘<froman.3>(<fitalic.3>x<froman.3>/3.5)<^>2<_>+3-
<fitalic.3>x<froman.3>/3.5’
t4 = ‘<froman.3>cos(<fitalic.3>x<froman.3>)<fitalic.3>e-
<froman.3,^>-<fitalic.3>x/9’
!
! create the "data" to plot
!
x  = [0:4*pi:.2]
y  = sin(x)*exp(x/5)
y2 = x/2-5
y3 = (x/3.5)^2+3*x/3.5
y4 = cos(x)*exp(-x/9)
!
```

```
! set up the graph legend
!
legend on
legend frame on
legend\percent frame 25 55 57 75
legend title '<h3%,froman.serif>Test Legend'
legend transparency off
legend autoheight off
!
! plot the data curves
!
set
 %txthit 2.2
  lintyp 1
  pchar 1

legend nsymbols 1
graph t1 x y
set
 lintyp 10
 pchar 2

legend nsymbols 2
graph\-axes t2 x y2
set
 lintyp 5
 pchar 3

legend nsymbols 3
graph\-axes t3 x y3
set
 lintyp 7
 pchar 13

legend nsymbols 4
graph\-axes t4 x y4
!
! replot to clean it all up
!
replot
defaults
```

5.12.2 A legend without a frame

This is another example of how a multiple curve graph with legend could be done.




```
file='sigma.dat'
read file\[3] t1 sigmaexp experr
black = 1
red   = 2
green = 3
blue  = 4
set
 txthit 1
 font triumf.2
 ylog 10
 nsyinc 9
 nlxinc 3
 nsxinc 4
 pchar -1
 xtica 90
 ytica -90
 colour black

scales 180 300 6 1 4 3
legend off
legend nsymbol 3
legend frame off
legend frame 20 63 68 82
legend title '<c1><u>16<d>O(<pi><u>+<d>,<pi><u>-<d><pi><u>+<d>)'
graph\axesonly
read file\[8:31] t2 oset1
read file\[34:57] oset2\2
read file\[60:83] oset3\2
set
```

```
 pchar 3
 lintyp 3
 colour blue

legend on
graph\-axes 'Cohen et al. NP_A395(1983)' t2 oset1
set
 pchar 10
 colour red

graph\-axes 'Oset et al. NP_A454(1986)' t2 oset2
set
 pchar 14
 colour green

graph\-axes 'Rockmore PR_C27(1983)' t2 oset3
read file\[86:105] t3
read file\[108:127] rock2\2
set
 lintyp 1
 colour black

graph\-axes 'This work' t3 rock2*7
set
   cursor -1
  %xloc 15
  %yloc 94

text '<sigma>(<mu>b)'
set
  %txthit 2.5
   cursor -3
  %xloc 95
  %yloc 3.0

text 'T<_><pi><^> (MeV)'
replot
```

5.13 Avoiding plotting symbol overlaps

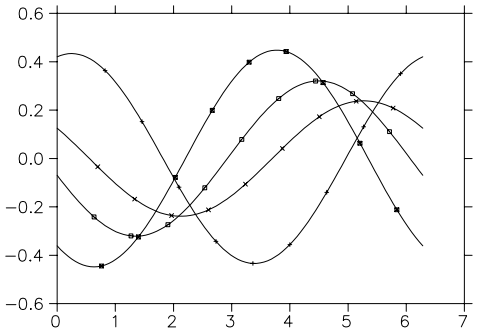
Suppose you have several curves on a graph and you want to have a plotting symbol at every PIth point. Suppose each curve uses plotting symbol number PS. Then it might be nice to arrange that these plotting symbols do not overlap. The following algorithm will prove helpful.

Make vector I=[1;2;3;...] with length equal to be that of the curve you want to plot. Let PS be

the plotting symbol number, increment this for each new curve. Let PI be the plotting interval. A plotting symbol is placed on the curve every PI^{th} point. Let P0 be the point number offset of the first tagged point, increment this also for each new curve. For the plotting symbol array, use:
PS*INT(I/(PI*(INT((I-1-P0)/PI+1)+P0))

The following example plots several random sine functions, tagging every curve with a different symbol.

```
GENERATE X 0,,2*PI 100
TMP = 1
PI = 10
PS = 0
P0 = 0
I = [1:LEN(X)]
DO J = [1:4]
  PS = PS+1
  P0 = P0+1
  SET PCHAR PS*INT(I/(PI*(INT((I-1-P0)/PI)+1)+P0))
  OFFSET = 40*RAN(TMP)
  AMP = RAN(TMP)
  GRAPH\-AXES X AMP*SIN(X-OFFSET)
ENDDO
REPLOT
DESTROY TMP PI PS P0 I J OFFSET AMP
```



5.14 Filling

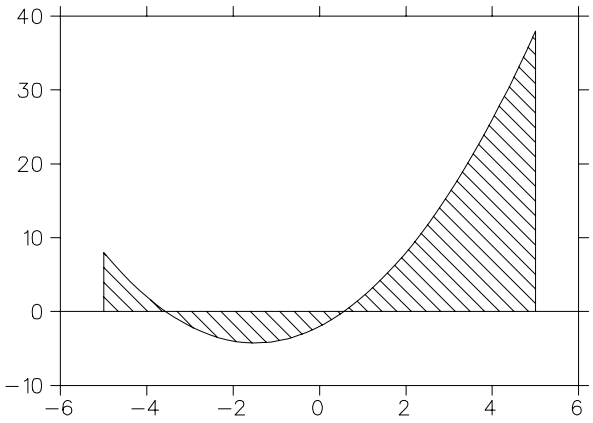
5.14.1 Fill the area under a curve

The script below, curvefill.pcm, will fill the area under a curve with either a hatch pattern or a dot pattern, that is, it will fill the area bounded by the curve and a horizontal line through the origin. It expects the vector names containing the data, with the independent vector first and the dependent vector second.

The third parameter the script expects is the fill pattern number. If this number, say f, is between 101 and 110, then hatch pattern number $100 - f$ is used for filling. If f is between 201 and 299 then a grey scale dot pattern is used for filling, where the fill pattern, mn is $200 - f$. The digit m is the horizontal dot increment and the digit n is the vertical dot increment. For example, a dot pattern of $mn = 45$ means to light up every fourth dot horizontally and every fifth dot vertically.

The following commands generate some data and call curvefill.pcm, producing the figure on the right.

```
GENERATE X -5,,5 50
Y = X^2+3*X-2
WINDOW 5
@curvefill X Y 108
```



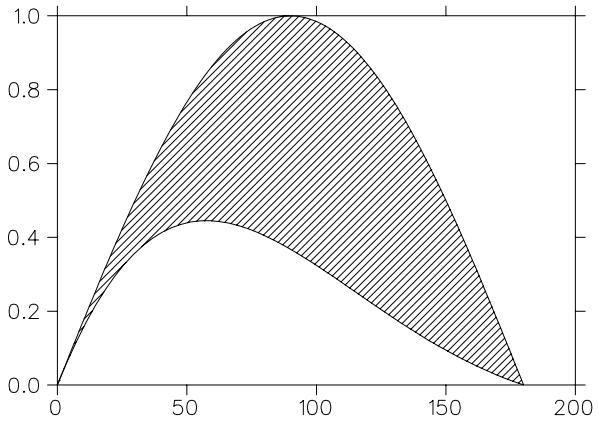

```
! script curvefill.pcm
!
XO = ?1
YO = ?2
SET LINTYP ?3
!            100 < LINTYP <= 110 --> use hatch pattern number LINTYP-100
!                  LINTYP >  200 --> use dot pattern LINTYP-200
L = LEN(XO)
XO[2:L+1] = XO[1:L]
XO[L+2]   = XO[L+1]
YO[2:L+1] = YO[1:L]
YO[1]     = 0
YO[L+2]   = 0
GRAPH XO YO
ZEROLINES\HORIZONTAL     ! draw horizontal line thru (0,0)
DEFAULTS
DESTROY XO YO L          ! eliminate dummy variables
```

5.14.2 Fill the area between two curves

The script below, `fill.pcm`, fills the area between two curves with either a hatch pattern or a dot pattern. The figure on the right was produced with the following commands.

Graph Examples

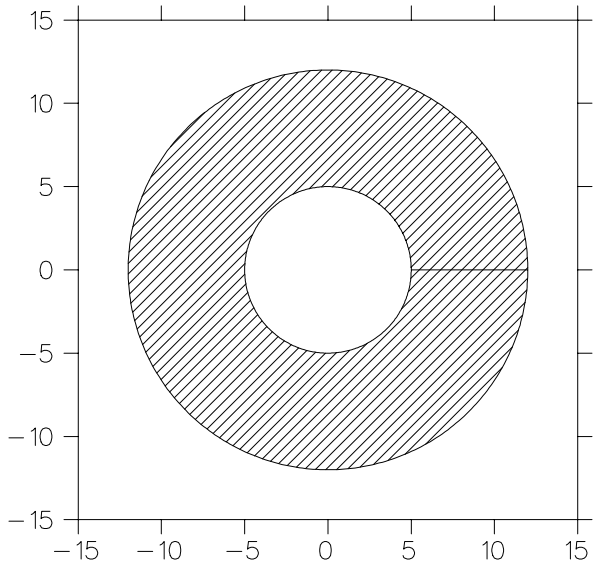
```
generate x 0,,180 100
y = sind(x)
generate x1 0,,180 50
y1 = exp(-x1/90)*sind(x1)
@fill x y x1 y1 107
```




```
! script fill.pcm
!
XO  = ?1                    ! make dummy vector XO = first vector
YO  = ?2                    ! make dummy vector YO = second vector
XO2 = ?3                    ! make dummy vector XO2 = third vector
YO2 = ?4                    ! make dummy vector YO2 = fourth vector
L = LEN(XO)                 ! make scalar L = length of first vector
L1 = LEN(XO2)               ! make scalar L1 = length of third vector
XO[L+1:L+L1] = XO2[L1:1:-1] ! append XO2 in reverse order onto end of XO
YO[L+1:L+L1] = YO2[L1:1:-1] ! append YO2 in reverse order onto end of YO
SET LINTYP ?5
!            100 < LINTYP <= 110 --> use hatch pattern number LINTYP-100
!                  LINTYP >  200 --> use dot pattern LINTYP-200
GRAPH XO YO
DEFAULTS                    ! reset program defaults
DESTROY XO YO XO2 YO2 L L1  ! eliminate dummy variables
```

5.14.3 A filled ring

The script `filled-ring.pcm` generates a commensurate graph of an area filled ring, centred at $(0, 0)$, filled with a hatch pattern. The user inputs the outer radius, the inner radius, and the hatch pattern. The radii are assumed to be in graph units. The figure oposite was produced with the command: `@filled-ring 12 5 107`

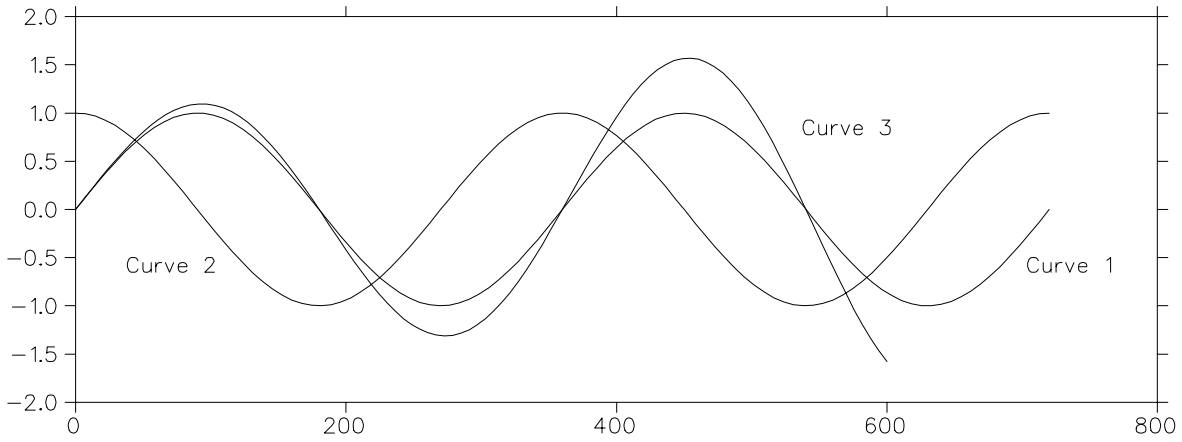



```
! This script will draw two circles and fill the region between them
! OUTER_RADIUS  =  outer circle radius
! INNER_RADIUS  =  inner circle radius
! NPTS          =  number points used to draw circles
! NFILL         =  fill pattern number
!
! set up circle parameters and fill pattern
OUTER_RADIUS = ?1     ! make scalar
INNER_RADIUS = ?2     ! make scalar
NPTS = 360            ! make scalar
NFILL = ?3            ! make scalar
! Generate the data for the circles
GENERATE THETA 0,,360 NPTS         ! make temporary vectors
XTEMP = OUTER_RADIUS*COSD(THETA)   !
YTEMP = OUTER_RADIUS*SIND(THETA)   !
! append in reverse order
XTEMP[NPTS*2:NPTS+1:-1] = INNER_RADIUS*COSD(THETA)
YTEMP[NPTS*2:NPTS+1:-1] = INNER_RADIUS*SIND(THETA)
! Draw the circles and fill
SET
  LINTYP NFILL
  PCHAR 0                          ! no plotting symbol
  AUTOSCALE COMMENSURATE           ! choose axis autoscaling type
 %XNUMSZ 2.5                       ! x-axis number size
 %YNUMSZ 2.5                       ! y-axis number size

GRAPH XTEMP YTEMP     ! plot data with axes
DEFAULTS              ! reset program defaults
```

```
DESTROY OUTER_RADIUS INNER_RADIUS NPTS NFILL THETA XTEMP YTEMP
```

5.15 Text tied to a curve

The following script illustrates the use of string arrays and the way that text can be forced to follow a data curve with the REPLOT command. The figure below was produced with this script.



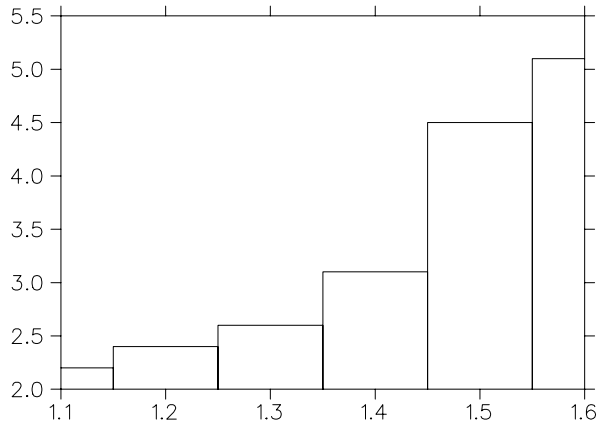

```
GENERATE X1 0,,720 100          ! make vector X1
X2 = X1                         ! make vector X2
GENERATE X3 0,,600 100          ! make vector X3
Y1 = SIND(X1)
Y2 = COSD(X2)
Y3 = EXP(X3/1000)*SIND(X3)
WINDOW 3                        ! use pre-defined window
TXT[1] = ‘Curve 1’              ! store text string in text array variable
TXT[2] = ‘Curve 2’
TXT[3] = ‘Curve 3’
GRAPH X1 Y1                     ! plot data with axes
TEXT\GRAPH TXT[1]               ! draw text
GRAPH\-AXES X2 Y2               ! overlay data curve
TEXT\GRAPH TXT[2]               ! draw text
GRAPH\-AXES X3 Y3               ! overlay data curve
TEXT\GRAPH TXT[3]               ! draw text
CLEAR\-REPLOT                   ! clear graphics but not replot buffers
REPLOT 3                        ! replot on common scale
DESTROY X1 X2 X3 Y1 Y2 Y3 TXT
DEFAULTS
```

6 HISTOGRAM EXAMPLES

6.1 Basic histogram

The commands listed below produce the histogram in the figure on the right. All of the program's defaults are used.

```
X = [1.1:1.6:0.1]
Y = [2.2;2.4;2.6;3.1;4.5;5.1]
GRAPH\HISTOGRAM X Y
```



6.2 Histogram types

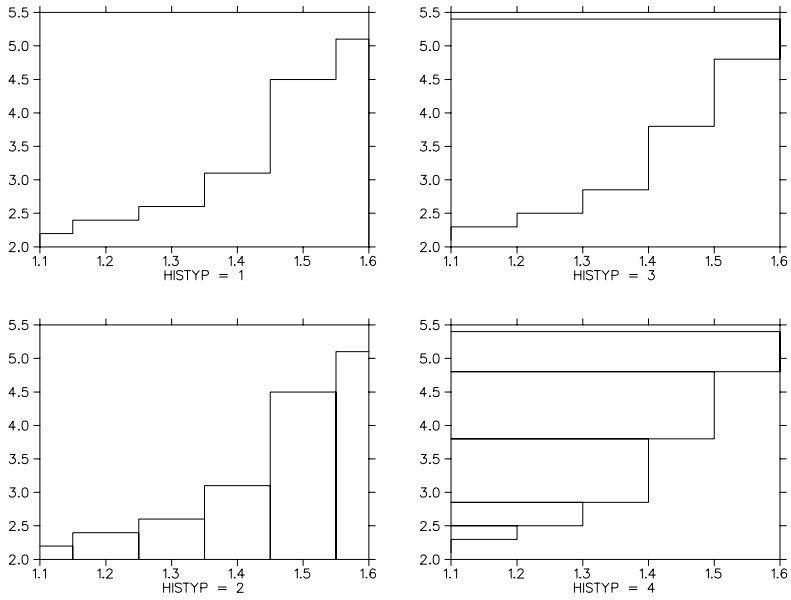
HISTYP	*result*
0	(default value) line graph, not a histogram
1	histogram with no tails and profile along the x-axis. You may control the width and colour of each individual bar.
2	histogram with tails to $y = 0$ and profile along the x-axis. You may control the filling pattern, width and colour of each 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 6.19: The basic histogram types

It is possible to draw four basic types of histograms using the SET HISTYP approach, or you can use the \HISTOGRAM qualifier on the GRAPH command to plot a histogram with tails to $y = 0$ and profile along the x-axis. Table 6.19 shows the histogram type that will be produced depending on the value of HISTYP.

The following commands produce examples of the four basic histogram types.

```
X = [1.1:1.6:0.1]
Y = [2.2;2.4;2.6;3.1;4.5;5.1]
WINDOW 5
SET HISTYP 1
LABEL\X 'HISTYP = 1'
GRAPH X Y
WINDOW 6
SET HISTYP 2
LABEL\X 'HISTYP = 2'
GRAPH X Y
WINDOW 7
SET HISTYP 3
LABEL\X 'HISTYP = 3'
GRAPH X Y
WINDOW 8
SET HISTYP 4
LABEL\X 'HISTYP = 4'
GRAPH X Y
```



6.3 Filled histograms

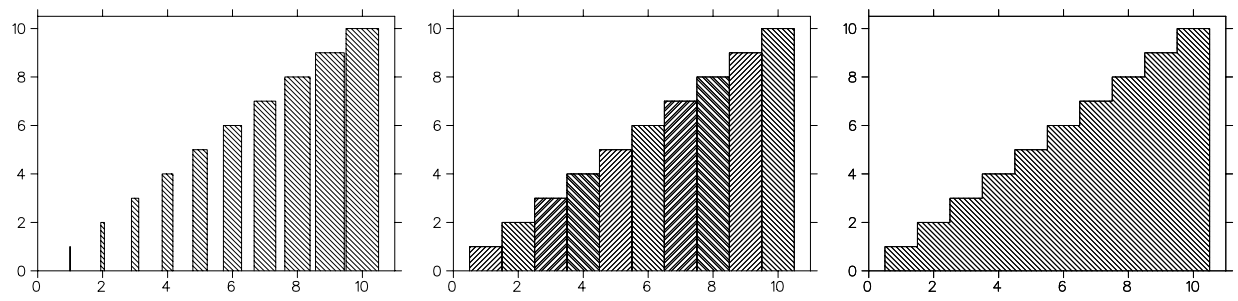
6.3.1 Fill bars with different widths

The following commands illustrate one way to produce a histogram with filled bars, where the bars also have different widths. The first graph in Figure 6.8 on page 57 is the result.

```
X = [1:10]          ! fake some data
GENERATE W 0,,1 10 ! make width vector W
SET HISTYP 2        ! tails to y=0
SET PCHAR 8 W       ! fill pattern #8, width W
SCALE 0 11 0 10.5   ! fix axis scales
GRAPH X X           ! draw the histogram
```

The following commands illustrate one way to produce a histogram with filled bars, where the bars have different filling patterns. The middle graph in Figure 6.8 is the result.

```
F = MOD([0:9],4)+7  ! produces f=[7;8;9;10;7;8;9;10;7;8]
X = [1:10]          ! fake some data
SET PCHAR F         ! fill patterns are in F
SCALE 0 11 0 10.5   ! fix axis scales
GRAPH\HISTOGRAM X X ! draw the histogram
```



(a) Different widths (b) Different patterns (c) Filled profile

Figure 6.8: Filled histograms

6.3.2 Fill under a histogram profile

The following commands illustrate one way to fill under a histogram profile, that is, to fill between the profile and a horizontal line through $y = 0$. The third graph in Figure 6.8 is the result.

```
X = [1:10]          ! fake some data
SET HISTYP 1        ! no tails
SET LINTYP 108      ! fill pattern #8
SCALE 0 11 0 10.5   ! fix axis scales
GRAPH X X           ! draw the histogram
```

7 DENSITY AND CONTOUR PLOTS

7.1 Box type density plots

Figure 7.9 was produced with the following commands:

```
X = {1;0;1;0;.2;.5;.8}
Y = {1;0;0;1;.2;.5;.8}
Z = {0;0;0;0;10;0;-5}
GRID\XYOUT X Y Z M XOUT YOUT  ! interpolate regular matrix from sparse data
WINDOW 5
LABEL\XAXIS 'DENSITY\BOXES XOUT YOUT M'
DENSITY\BOXES XOUT YOUT M          ! density plot with boxes
WINDOW 7
LABEL\XAXIS 'DENSITY\BOXES\DERIV XOUT YOUT M'
DENSITY\BOXES\DERIV XOUT YOUT M    ! box plot of derivative
```

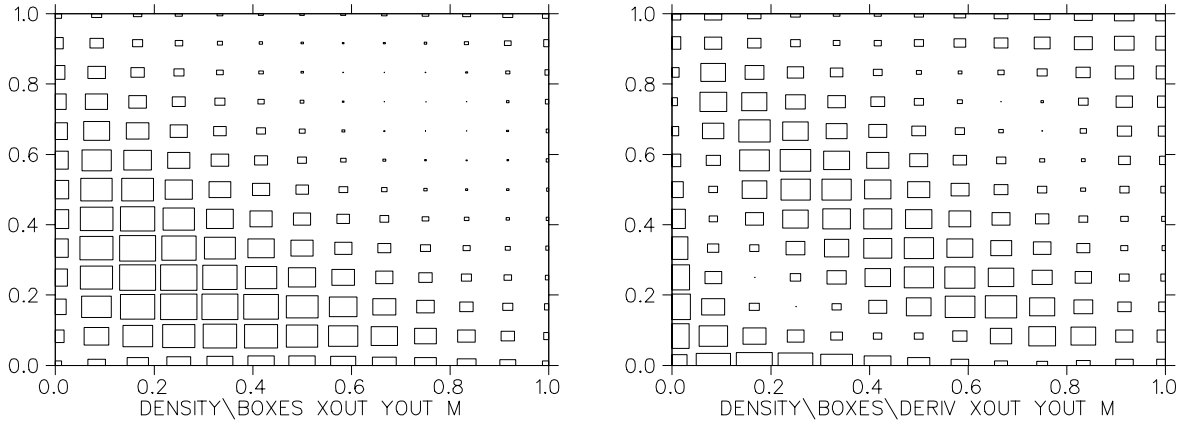


Figure 7.9: Box type density plots

7.2 Profiles on density plots

Figure 7.10 was produced with the following commands:

```
X = {1;0;1;0;.2;.5;.8}
Y = {1;0;0;1;.2;.5;.8}
Z = {0;0;0;0;10;0;-5}
SET %XLABSZ 2.5     ! x-axis label size
WINDOW 5
LABEL\XAXIS 'DENSITY\BOXES\PROFILE XOUT YOUT M'
DENSITY\BOXES\PROFILE XOUT YOUT M
WINDOW 7
SCALE 0 .5 5 0 .5 5
LABEL\XAXIS 'DENSITY\BOXES\PARTIAL\PROFILE XOUT YOUT M'
DENSITY\BOXES\PARTIAL\PROFILE XOUT YOUT M
DEFAULTS
```

7.3 Contour plots

Figure 7.11 on page 60 was produced by the following commands:

```
X = {1;0;1;0;.2;.5;.8}
Y = {1;0;0;1;.2;.5;.8}
Z = {0;0;0;0;10;0;-5}
GRID\XYOUT X Y Z M XOUT YOUT   ! interpolate a matrix
SET %LEGSIZ 2.3               ! legend label size
WINDOW 5
LABEL\X 'CONTOUR\LEGEND XOUT YOUT M'
```

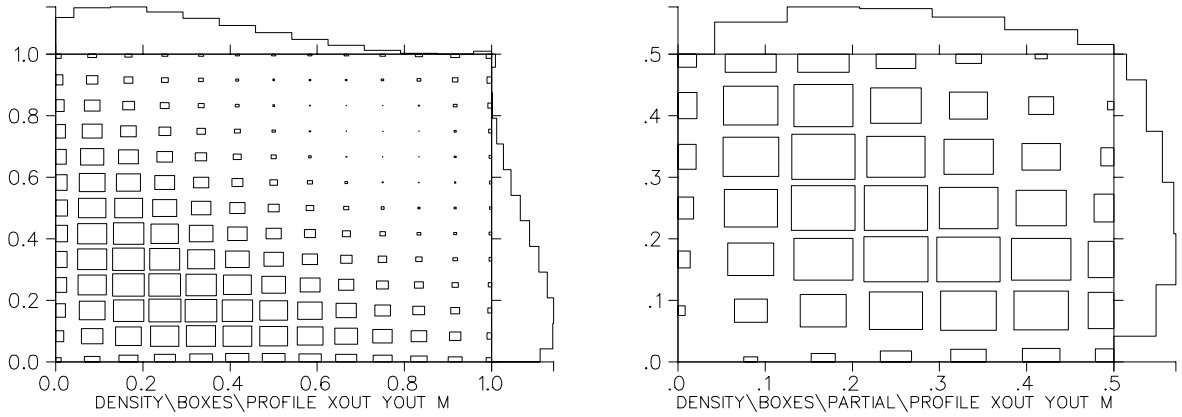
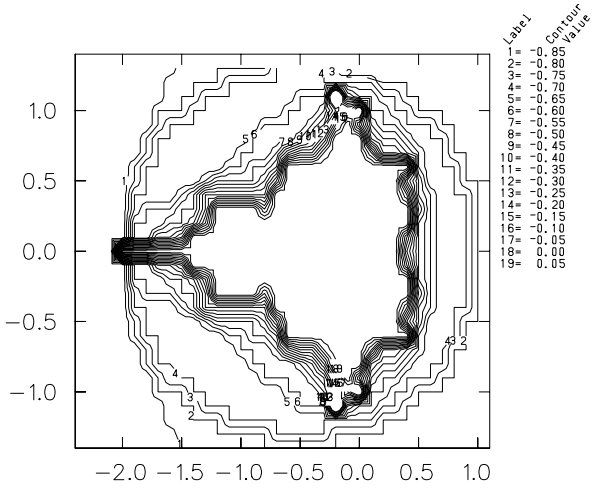


Figure 7.10: Profiles on density plots

```
CONTOUR\LEGEND XOUT YOUT M 10  ! 10 contours of matrix M
WINDOW 7
LABEL\X 'CONTOUR\LEGEND X Y Z'
CONTOUR\LEGEND X Y Z 10        ! 10 contours of scattered points
WINDOW 6
SCALE 0 .5 5 0 .5 5            ! set axes scales for zooming in
LABEL\X 'CONTOUR\PARTIAL XOUT YOUT M'
CONTOUR\PARTIAL XOUT YOUT M 10 ! contour plot with no legend
WINDOW 8
LABEL\X 'CONTOUR XOUT YOUT M'
CONTOUR XOUT YOUT M 10         ! 10 contours, no legend
DEFAULTS
```

7.4 Mandelbrot set

The following script will plot a Mandelbrot set of points. Only the contour plot figure is shown, since the PostScript files for the density plots would be quite large.



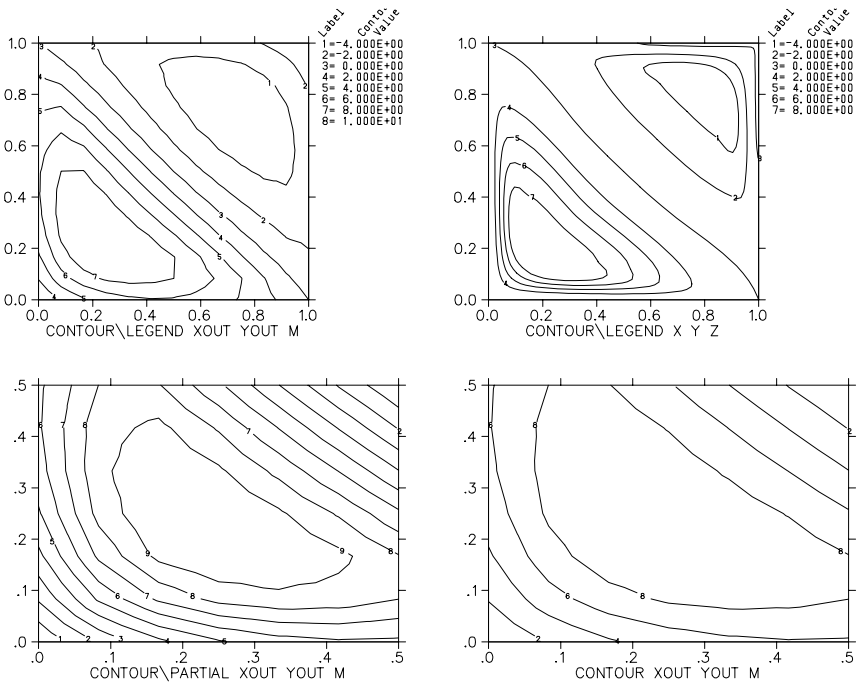


Figure 7.11: Contour plots

```
!
! PHYSICA example: mandel.pcm
!
!Defaults
Disable Journal
Disable History
Set Maxhistory 1
Destroy *
! Specify the size of matrices
! If you have an 'allocating dynamic array space' problem, use Rough
Qu = 'R'
Inquire 'Picture quality (R =Rough/ F =Fine):' Qu
S=0.1
If Eqs(Ucase(Qu[1]),'F') then S=0.02
!
X = [-2.4:1.1:S]
Y = [-1.4:1.4:S]
Nmax = 10
Lx = Len(X)
Ly = Len(Y)
Matrix M Xx Yy Tt Ly Lx
M[1:Ly,1:Lx] = 0
Xx = M
```

```
Yy = M
Xxm[1:Ly] = 20
Xxi[1:Ly] = -20
Yym[1:Ly] = 10
Yyi[1:Ly] = -10
Scalar\Dummy J
Do K = [1:Nmax]
  Display 'Progress='//Rchar(K/Nmax*100,'F5.0')//'%'
  Tt = Xx^2 - Yy^2 + ( <- Loop(X,J,1:Ly))
  Yy = 2*Xx*Yy + Loop(Y,J,1:Lx)
  Xx = Tt
  M = M + (4<(Xx^2+Yy^2))
  Xx = Max(Xx,Loop(Xxi,J,1:Lx))
  Xx = Min(Xx,Loop(Xxm,J,1:Lx))
  Yy = Max(Yy,Loop(Yyi,J,1:Lx))
  Yy = Min(Yy,Loop(Yym,J,1:Lx))
Enddo
M = (1-M)/Nmax
!
! Plot the picture
Device Off
Set
 xtica 90
 ytica -90

Clear
Scales -2.4 1.1 -1.4 1.4
Density\-Border X Y M
Terminal
Clear
Density\-Border\Diff X Y M
Terminal
Clear
Device PostScript A
Cntr=[-0.85:0.05:0.05]
Set Legfrmt (1x,F5.2)
Contour\Specific\Legend\-Border X Y M Cntr
Defaults
```

8 DATA MANIPULATION

8.1 Scaling data

Suppose that a dipole magnet survey has just been completed and tied to an absolute NMR calibration at some point. This calibration requires the entire field measured to be scaled up by a factor of 1.0157. The field was measured over a 10×20 point grid and written into a file containing twenty lines, each line with ten numbers in 10F8.4 format. Only three commands are needed:

```
READ\MATRIX\FORMAT FIELD.DAT (10F8.4) B 10 20    ! read the data into matrix B
B2 = 1.0157*B                                    ! scale the data
WRITE\MATRIX\FORMAT FIELD2.DAT (10F8.4) B2       ! write to a file
```

8.2 Generating data vectors

To generate a vector X = {1.0;1.1;1.2;1.3;...;2.0} enter:

```
GENERATE X 1,,2  11
```

To generate a vector Y = {1;3;5;7;9;...;21} you could enter:

```
GENERATE Y 1 2,,11   or
Y = [1:21:2]
```

To generate vector X filled with 50 random numbers between -2 and $+3$, enter:

```
GENERATE\RANDOM X -2 3 50
```

8.3 Selecting data

Suppose you have three vectors, X, Y and Z, and you want to extract the X's and Y's when Z[i] satisfies some condition, such as $3 <$Z[i]< 5. You could enter:

```
COPY [1:LEN(Z)] KEY IFF (Z>3)&(Z<5)   or
KEY = WHERE((Z>3)&(Z<5))
```

Then you have a vector KEY which can be used for vector indexing. For example,

```
GRAPH X[KEY] Y[KEY]
```

8.4 Copying vectors

To copy all of vector X into vector Y, enter:

```
COPY X Y
```

If Y does not exist, it is created. If Y does exist already, the new data will overlay the old. Of course, you could also enter the following expression to accomplish the same thing, except that if the Y vector exists already, it is destroyed and created anew.

```
Y = X
```

Suppose that you have a vector X. The following will double the length of X by duplicating X into itself.

```
X[LEN(X)+1:2*LEN(X)] = X   or
X = X//X
```

Entering the following command:

```
Y[1:21:5] = X[2:10:2]
```

will put X[2] into Y[1], X[4] into Y[6], X[6] into Y[11], X[8] into Y[16], and X[10] into Y[21]. If Y did not already exist, then Y would be created with 21 elements, and the unfilled elements will be set to zero, that is, Y[2] = Y[3] = Y[4] = Y[5] = Y[7] = 0, etc.

8.4.1 Conditional copying

Suppose you want to copy the elements of vector X into Y and the elements of vector A into B only when an expression is true, say but only when Z*cos(X) < 0. You could use:

```
COPY X A Y B IFF Z*COS(X)<0     or
Y = X[WHERE(Z*COS(X)<0)]   and
B = A[WHERE(Z*COS(X)<0)]
```

8.5 A simple average calculation

The script average2.pcm calculates the average of every two elements of an input vector and outputs the averages into another vector. If X is the input vector with length n, and XX is the output vector, then XX[i] = X[2i-1] + X[2i] for i $= 1, 2, \ldots, n/2$.

```
! calculate average of every 2 elements of input vector and
! output averages to a vector
!
J = 0            ! initialize counter J
K = 1            ! initialize counter K
START:           ! a label
J = J+1
IF (J>LEN(?1)) THEN GOTO END
A = ?1[J]        ! assign a value to scalar A
J = J+1          ! increment J
IF (J>LEN(?1)) THEN GOTO END
B = ?1[J]        ! assign value to scalar B
?2[K] = (A+B)/2  ! assign average value to vector element
```

```
K = K+1          ! increment K
GOTO START       ! go to label
END:             ! another label
```

For example, if `X = [1:10]` then `@average2 X Y` produces vector `Y = {1.5;3.5;5.5;7.5;9.5}`

8.6 Find the root mean square distribution

Suppose we have a set of events in (X,Y) space and we wish to determine the root mean square distribution of Y versus X.

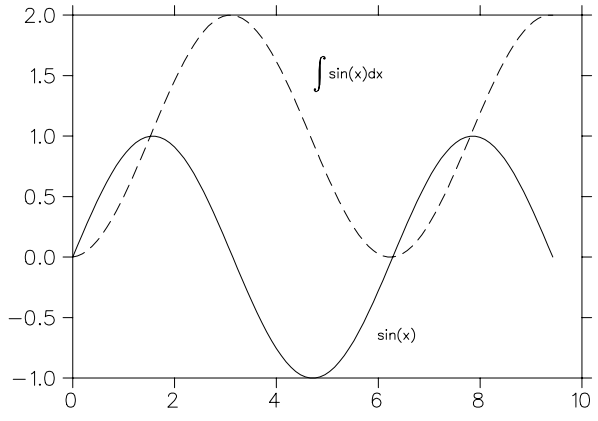
```
W2 = Y*Y                        ! to accumulate mean of squares
W1 = Y                          ! to accumulate mean
W0[1:LEN(X)] = 1                ! to accumulate counts
BIN\WEIGHT\NBINS W2 X XB C2 50  ! bin for mean of squares
BIN\WEIGHT\NBINS W1 X XB C1 50  ! bin for means
BIN\WEIGHT\NBINS W0 X XB C0 50  ! bin for counts
RMS = SQRT(C2/C0-(C1/C0)^2)
```

Here, RMS $= \sqrt{m_s - m^2}$, where m_s is the mean of squares and m is the mean.

8.7 A simple integration procedure

The following script, integrate.pcm, expects two parameters: the independent and dependent vector names. The following commands generate some data and then call the integrate.pcm procedure, producing the figure on the right. Of course, you could use the INTEGRAL function instead of this procedure. This is intended only as an example.

```
generate x 0,,3*pi 100
@integrate x sin(x)
```




```
! script integrate.pcm
!
```

```
! Assume that you have two vectors X and Y with X(i+1)-X(i) constant
! and assume that X and Y have same number of elements.
! This file will integrate Y with respect to X and graph the data
! and the 'integral'
GRAPH ?1 ?2                    ! plot input data, with axes
SET LINTYP 3                   ! set line type to dashed line
SCALAR\DUMMY I
GRAPH\-AXES ?1 RSUM(?2[I],I,1:LEN(?2))*(?1-STEP(?1,1))
REPLOT                         ! redraw plot on common scale
SET
 %XLOC 63        ! text x location
 %YLOC 23        ! text y location
  CURSOR -1      ! left justify text
 %TXTHIT 2       ! text height

TEXT 'sin(x)'                  ! draw text string
SET
 %XLOC 53        ! text x location
 %YLOC 77        ! text y location

TEXT '<Int>sin(x)dx'
DEFAULTS                       ! reset program defaults
```

8.7.1 Dealing with data spikes

Suppose one had noise 'spikes' in an array of data, Y, which were one 'channel' wide. Suppose these noise spikes always exceeded the scalar value YCUT. Now we want to plot Y, but for values $>$ YCUT we want to just plot a straight line between the previous point and the next. The following commands illustrate the technique.

```
X = [1:100]
Y = SIN(X/10)+5*(RAN(X)>0.94)
YCUT = 1
WINDOW 5
GRAPH X Y
WINDOW 7
YGOOD = Y*(Y<YCUT)+(STEP(Y,1)+STEP(Y,-1))/2.*(Y>YCUT)
GRAPH [1:LEN(YGOOD)] YGOOD
```

So all we did was place in YGOOD the 'good' values of Y or an average of the previous and next point when it was 'bad'.

Note that we have not taken care of the case where there are two or more bad points in a row.

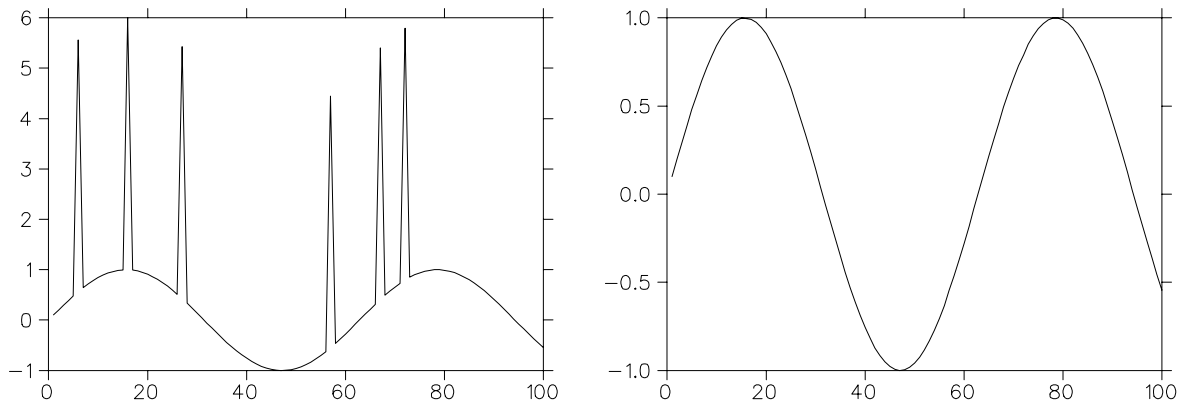


Figure 8.12: Dealing with data spikes

Also, the check for a good point can be much more complex than simply comparing Y with YCUT. For example (Y/2<SQRT(YCUT+10)) could have been the criteria.

8.8 Fitting

Suppose you have generated some data and it resides in columns in a file, file.dat. For example:

```
 .          .
 .          .
THIS IS LINE 100 OF FILE.DAT
1.0       -9.80
1.1       -8.75
1.3       -7.98
 .          .
 .          .
10.35     63.29
THIS IS LINE 201 OF FILE.DAT
 .          .
 .          .
```

Read in the data:

```
READ FILE.DAT\[101:200] X Y
```

Plot the data, using unjoined boxes for plot symbols:

```
SET PCHAR -1
GRAPH X Y
```

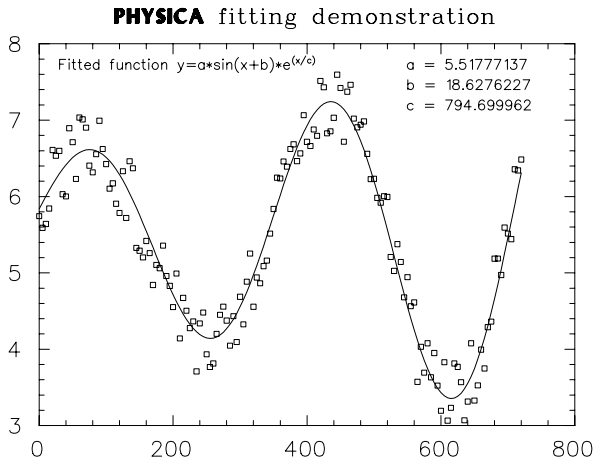
Declare and initialize fitting parameters, do the fit and update the fitted Y values into vector YFIT,

then overlay a plot of the fitted line using no plot symbols, and get a hardcopy of the resultant plot:

```
SCALAR\VARY A B       ! declare fitting parameters
A = 8                 ! initialize fit parameter
B = -10               ! initialize fit parameter
FIT Y=A*X+B           ! do the fit
FIT\UPDATE YFIT       ! update the fitted values
SET PCHAR 0           ! no plotting symbol
GRAPH\-AXES X YFIT    ! overlay fitted curve
HARDCOPY              ! get a hardcopy of the plot
```

8.8.1 Fit to a non-linear function

The following script produces this figure.




```
X = [0:720:5]                      ! make some 'data'
Y = 5+SIND(X+20)*EXP(X/800)+RAN(X) !
SET                                !
 PCHAR -1                          !
 FONT TSAN                         !
 NSXINC 5                          !
 NSYINC 5                          !
 XTICA 90                          ! tic marks to point to inside of axis box
 YTICA -90                         !

GRAPH X Y                          !
SCALAR\VARY A,B,C                  ! declare fit parameters
C = 100                            ! initialize to get a reasonable start
FIT Y=A+SIND(X+B)*EXP(X/C)         !
FIT\UPDATE YF                      !
SET PCHAR 0                        !
GRAPH\-AXES X YF                   ! overlay the fitted function
SET
```

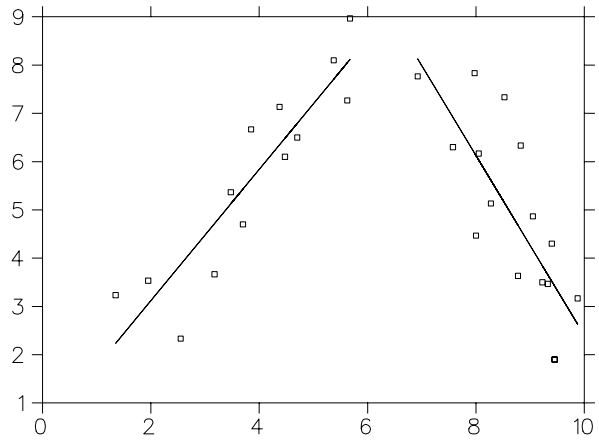
```
  %XLOC   18
  %YLOC   85
  %TXTHIT 2
   CURSOR -1

TEXT 'Fitted function y=a*sin(x+b)*e<^>(x/c)'
SET
  %XLOC 70
  %YLOC 85

TEXT 'a = '//rchar(a)
SET %YLOC 81
TEXT 'b = '//rchar(b)
SET %YLOC 77
TEXT 'c = '//rchar(c)
SET
 %XLOC 55
 %YLOC 94
 %TXTHIT 3
  CURSOR -2

TEXT '<FROMAN.FASHON,B11>PHYSICA <b,FTRIUMF.2>fitting demonstration'
DEFAULTS
```

8.8.2 Fit data with two line segments

Suppose you have already read in, or generated in some way, two vectors, X and Y. You plot Y versus X, first turning on the autoscaling:

```
SET AUTOSCALE ON
SET PCHAR 0
GRAPH X Y
```

Now, you want to fit two line segments to the plotted curve, such that they join at one end point, X0. An example plot can be seen on the right.




```
SCALAR\VARY A B C D
FIT Y=(A+B*X)*(X<X0)+(C+D*X)*(X>=X0)+(A+B*X0-C-D*X0)*1.E5*(X=X0)
I1 = WHERE(X<X0)  ! indices where x < x0
```

```
I2 = WHERE(X>=X0) ! indices where x >= x0
SET PCHAR 0
GRAPH\-AXES X[I1] A+B*X[I1]
GRAPH\-AXES X[I2] C+D*X[I2]
REPLOT
```

8.8.3 More than one independent variable

Suppose we had to dig a straight trench down a street where the houses are set back from the street by varying distances. Find the trench which would result in the minimum usage of the water pipe segments from the trench to the houses. Also determine the sum of the segments needed.

Suppose the data of the X,Y co-ordinates of the connections to the houses are stored in the file houses.dat then proceed as follows:

```
READ HOUSES.DAT X Y                  ! read in house positions
SCALAR\VARY A B                      ! declare the free parameters
A=1                                  ! initialize
B=1                                  ! initialize
D=X-X                                ! connect to centre of trench
FIT D=SQRT(ABS(A*X+B-Y)/SQRT(A*A+1)) ! note SQRT
SEG=ABS(A*X+B-Y)/SQRT(A*A+1)         ! pieces
STATISTICS SEG SUM\SUMSEG            ! sum of the segments
```

Since least squares is used for fitting we had to take the square root of the expression (which is the distance of a point to a straight line with slope A and intercept B). Note that we have more than one independent variable in the expression. Extension to multidimensional cases involving say Z is straightforward.

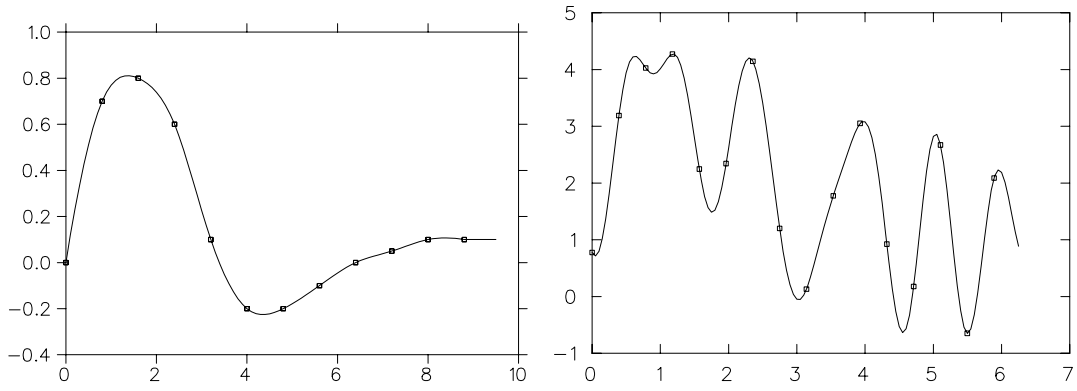
The problem of fitting to a scattered set of X,Y points with a straight line which is on the average S units away will, in general, have two solutions (which one you get will depend on the starting values of A,B). The only line to replace above is:

```
D=X-X                ! connect to centre of trench
D=X-X+5              ! aim for 5 unit offset
```

Note also that the optional weight array could be used to fit 'closer' to selected data points.

8.9 Interpolation and smoothing

The following commands illustrate interpolation, using a spline under tension, producing the left graph in Figure 8.13. Note that this uses the default tension of 1.0, but if the tension is set higher, say to 10, the interpolation would be basically linear.



(a) Spline interpolation (b) FFT interpolation

Figure 8.13: Interpolation examples

```
X=[0:9.6:.8]
Y={0;.7;.8;.6;.1;-.2;-.2;-.1;0;.05;.1;.1}
XI = [0:9.6:.1]                ! where you want interpolated values
SET PCHAR -1                   !
GRAPH X Y                      ! plot the original data
SET PCHAR 0                    !
GRAPH\-AXES XI INTERP(X,Y,XI)  ! overlay the interpolated curve
REPLOT                         ! redraw on common scale
```

8.9.1 Interpolation with a fast Fourier transform

The following example illustrates how data can be interpolated using fast Fourier transforms. See the graph on the right in Figure 8.13.

```
N = 8
X = [0:2*N-1]*PI/N
Y = SIN(X)+5*RAN(X)
SET
  PCHAR -1
  XTICA 90           ! tic marks to point inside the box
  YTICA -90
```

```
GRAPH X Y            ! plot the data
M = FFT(Y,'COS&SIN') ! put the cos and sin coefficients into matrix M
XI = [0:2*PI:.05]    ! the interpolant locations
SCALAR\DUMMY K       ! dummy variable needed for SUM function
SET PCHAR 0
GRAPH\-AXES XI M[1,1]/2+SUM(M[K,1]*COS((K-1)*XI)+M[K,2]*SIN((K-1)*XI),K,2:9)
REPLOT
DEFAULTS
```

Index