C-PLOT™

Scientific Graphics
and
Data Analysis Software

USER REFERENCE MANUAL

This is version 2.1 of the C-PLOT documentation, printed January 12, 1999, describing features of release 4.0 of the software.

The material in this manual is furnished for informational use only, is subject to change without notice and should not be construed as a commitment by Certified Scientific Software. Certified Scientific Software assumes no responsibility or liability for any errors or inaccuracies that may appear in this manual. The software described in this manual is furnished under license and may only be used or copied in accordance with the terms of such license.

C-PLOT

Scientific Graphics
and
Data Analysis Software

Certified Scientific Software

PO Box 390640 □ Cambridge, Massachusetts 02139 □ (617) 576-1610
FAX: (617) 497-4242 □ cplot@certif.com
http://www.certif.com

Table Of Contents

Chapter 1 Introduction

Certified Scientific Software's C-PLOT offers scientific graphics and data analysis for UNIX operating systems. The package provides publication-quality 2D and 3D graphs, sophisticated fitting and modeling, immediate plotting of real-time data, and integration of these functions with the user's own data collection and analysis routines. Once a plot or procedure is developed interactively, C-PLOT can be run in batch mode using easy-to-program command files. C-PLOT supports most popular output devices, including X Windows and PostScript (both in color) and HP-GL plotters.

Analytical tools

C-PLOT's built-in analytical tools include a function for fast Fourier transforms, a cubic spline interpolator, a histogram-maker, a contour plot generator and a general-purpose data calculator. A special user function will filter data through any existing UNIX utility.

Fitting

The package's most powerful built-in function is its nonlinear least-squares fitting package. An interactive program in its own right, the fitting package is closely integrated with the plot program, letting you readily see graphic displays of the results of your fits. Just as with the ordinary user functions, you need only insert the code describing your model equation into the prototype provided.

User functions

The package's user functions are separate processes that communicate directly with the plot program, allowing you to create your own data collection and processing routines and integrate them into C-PLOT. The package includes overhead modules and prototype C-modules containing most of the code that comprises these programs. C-PLOT will create files containing the prototype subroutines and invoke your favorite editor. After you add your code, it will compile and run the function. Your role may be as simple as entering the C-language expression that gives y as a function of x, or it may involve writing many lines of code, depending on your need.

Data

C-PLOT takes data from binary and ASCII files or from the keyboard, it can digitize data from an HP-GL plotter or it can receive data through the user function facility. It will plot an unlimited number of points, and overlays of additional plots, text or formulas may be placed anywhere on the page and scaled to any size.

Batch mode

Besides its interactive mode, in which commands are typed at the keyboard, C-PLOT can take input from command files — ASCII files containing a command script entered just as it would be typed at the keyboard. Parameters can be passed to command files using a simple argument-substitution scheme, and command files can be nested up to four deep. C-PLOT can be invoked as a background process, taking its input entirely from command files — plotting and analyzing your data as you perform other tasks.

Graphics

For high-resolution graphics, C-PLOT produces a device-independent output stream that is directed through device-dependent filter programs. The C-PLOT package includes filter programs for many popular graphics terminals, laser printers and other output devices; if a graphics filter doesn't exist already for your graphics device, it is relatively simple to build one by adapting the existing C-language modules and libraries included in the plotting package.

Apart from the filter output stream is C-PLOT's bidirectional communication with an HP-GL plotter over a serial or GPIB interface. You can set the size and position of the plot axis and the position of plot annotations through the program or with the plotter's front-panel controls.

C-PLOT's fast PseudoGraphics will work on almost any video terminal that can run a UNIX visual editor. You can use PseudoGraphics, for example, to view the trend or scaling of your plot before sending it out to a high-resolution device. PseudoGraphics can be used with C-PLOT's data editor to delete, insert and modify points as they are displayed on the video screen.

Text

C-PLOT text formatting shares many of the control sequences of troff, the standard UNIX text formatter. The C-PLOT character set includes all the printing ASCII characters, all the Greek letters and more than 90 other math, foreign and special characters. You can choose from 28 special built-in symbols and 7 line types for plotting your points, or you can use any of the other characters as a symbol.

Eight type fonts are included. The default font is the lowest resolution and the fastest to draw. The alternate fonts are of increasing detail and complexity, allowing you to choose the most appropriate font for the size of your plot, the resolution of your display device and the required aesthetics. With each font, you can independently control the height, width and slant of text.

Help and demos

The C-PLOT package includes access to on-line help files that provide concise summaries of all the commands. C-PLOT's help utility will format the files on the fly to fit whatever terminal screens and workstation windows might be in use. The package includes *nroff*/*troff* macros and a *Makefile* to aid in producing hard copies of the help material. Demonstration command files also are included that show off many of the program's capabilities.

Commands

All C-PLOT commands, listed on the next two pages, are one- or two-letter mnemonics. Commands and parameters are shown in courier type. Italic parameters are to be replaced with the appropriate characters for the desired instruction. Optional parameters appear in square brackets following the mnemonics. When several parameters are shown separated by vertical lines, you use only one of them with the command. For commands that simply indicate *options*, consult the detailed command description for an explanation of the syntax.

<table>
<tr><th>Command</th><th>Description</th><th>Page</th></tr>
<tr><td>2d</td><td>Select 2D mode</td><td>25</td></tr>
<tr><td>3d</td><td>Select 3D mode</td><td>26</td></tr>
<tr><td>bo [*x*|. *y*|. *z*|.]</td><td>Set 3D box ratios</td><td>38</td></tr>
<tr><td>cd [*direc*]</td><td>Change directory</td><td>8</td></tr>
<tr><td>ch [p|z|0]</td><td>Change target of drawing cmds</td><td>77</td></tr>
<tr><td>cs [*options*]</td><td>Set character sizes</td><td>41</td></tr>
<tr><td>do [*cmdfile*|.]</td><td>Take commands from a file</td><td>73</td></tr>
<tr><td>eb [x|y|z] [0|1] [?]</td><td>Select error-bar mode</td><td>19</td></tr>
<tr><td>em</td><td>End making a command file</td><td>77</td></tr>
<tr><td>er</td><td>Erase the video screen</td><td>22</td></tr>
<tr><td>ex</td><td>Exit program</td><td>7</td></tr>
<tr><td>fn [*options*]</td><td>Run user function #1</td><td>84</td></tr>
<tr><td>f*#* [*options*]</td><td>Run user function 1 to 8</td><td>84</td></tr>
<tr><td>ft [*#*]</td><td>Select font</td><td>42</td></tr>
<tr><td>gd [*options*]</td><td>Get data</td><td>11</td></tr>
<tr><td>gk</td><td>Enter symbols and text for key</td><td>45</td></tr>
<tr><td>gr [*term*] [*char_set*]</td><td>Select PseudoGraphics terminal</td><td>23</td></tr>
<tr><td>h [*command*]</td><td>Get on-line help</td><td>7</td></tr>
<tr><td>in [m] [*dev*] [*baud*]</td><td>Open and initialize pen plotter</td><td>59</td></tr>
<tr><td>lc [0|1]</td><td>Select line-control mode</td><td>26</td></tr>
<tr><td>lo [*llx lly urx ury*]</td><td>Locate plot</td><td>37</td></tr>
<tr><td>mk cmdfile</td><td>Make a command file</td><td>76</td></tr>
<tr><td>np [x][y][z]</td><td>Reset axes for new points</td><td>29</td></tr>
<tr><td>p</td><td>Draw complete plot on the plotter</td><td>53</td></tr>
<tr><td>p*#*</td><td>Select pen (# is an integer)</td><td>61</td></tr>
<tr><td>pa</td><td>Draw axes on the plotter</td><td>54</td></tr>
<tr><td>pb</td><td>Draw error bars on the plotter</td><td>54</td></tr>
<tr><td>pd</td><td>Draw date in corner of plot</td><td>57</td></tr>
<tr><td>pk [*options*]</td><td>Draw key on the plotter</td><td>56</td></tr>
<tr><td>pl</td><td>Draw labels on the plotter</td><td>55</td></tr>
<tr><td>pn [*options*]</td><td>Draw annotation text on the plotter</td><td>55</td></tr>
<tr><td>pp</td><td>Draw points on the plotter</td><td>54</td></tr>
<tr><td>pt</td><td>Draw title on the plotter</td><td>55</td></tr>
<tr><td>pv [*velocity*]</td><td>Select pen velocity</td><td>61</td></tr>
<tr><td>pw</td><td>Don’t move pen off page</td><td>62</td></tr>
<tr><td>px</td><td>Move pen off page</td><td>62</td></tr>
<tr><td>pz</td><td>Draw complete plot on the plotter</td><td>53</td></tr>
<tr><td>ra [x][y][z] [*ranges*]</td><td>Select axis range</td><td>27</td></tr>
<tr><td>re</td><td>Reset program for new plot</td><td>30</td></tr>
<tr><td>ro [x][y][z]</td><td>select options for axis range</td><td>27</td></tr>
<tr><td>rp</td><td>Release pen plotter</td><td>60</td></tr>
<tr><td>sa[[-b] [*file* [a|w]]</td><td>Save current data</td><td>19</td></tr>
</table>

Command	Description	Page
sc [*h v*]	Select filter scaling factors	65
se [*options*]	Set parameters	47
sf [*filename*]	Save current format	77
st [*code*]	Select plot style	34
sw [*angle*]	Swivel plot	35
sy [*char*]	Select plotting symbol	44
tu [0|1]	Turn plot	31
tw	Tweak plot orientation	39
tx [*options*]	Enter text for plot labels and title	46
ty [*options*]	Select type of plot	31
u [*cmd*]	Create a UNIX subshell	9
v	Draw PseudoGraphics plot	21
va	Draw PseudoGraphics axes	21
vb [0|1]	PseudoGraphics axes inclusion	22
vi [*x*|. *y*|. *z*|.] [*dist*]	Set 3D view point	38
vp	Draw PseudoGraphics points	22
vt [0|1]	PseudoGraphics auto. drawing	22
w	Wait for user to enter <return>	78
wi [*options*]	Select size, place of plot window	35
yg [0|*#*]	Select gap between *y*-axis, label	49
z	Draw complete plot on filter	53
z*#*	Select filter line style	66
za	Draw axes on filter plot	54
zb	Draw error bars on filter plot	54
zd	Draw date in corner of filter plot	57
ze	Erase old plot	66
zE	Erase current window	66
zf	Select filter	65
zf*#*	Select filter 1 to 8	65
zi [*filter*] [*options*]	Initialize graphics filter	63
zk [*h v*]	Draw key on filter plot	56
zl	Draw labels on filter plot	55
zn [*h v*] [*file*]	Draw annotation text on filter plot	55
zp	Draw points on filter plot	54
zq	Don’t write text to screen	66
zs	Close filter, synchronous	68
zt	Draw title on filter plot	55
zw	Don’t close filter yet	67
zx	Close filter	68
zz	Draw complete plot on filter	53

Chapter 2 Utility commands

In this chapter you will find instructions for:

- □ interacting with the operating system
- □ accessing on-line help

Commands covered

ex	exit program
h	get on-line help
cd	change directory
u	create a UNIX subshell

ex exit program

terminates C-PLOT

```
ex
```

When `ex` is entered in interactive mode, a prompt will ask if you really want to leave the plot program. Pressing `<return>` will take you out of C-PLOT. If you enter any character other than `y`, `Y` or `1`, you will stay in C-PLOT. You also may exit the program by typing `^D` at the `PLOT->` prompt, in which case you will not be asked for confirmation.

When running from a command file (see Chapter 10), `ex` does not require confirmation.

h get on-line help

gives access to on-line information about C-PLOT's commands and functions.

```
h
or
h topic
```

Enter `h` with no parameters to display a listing of all C-PLOT help topics. When `h` and the desired subject are entered, a help file named *topic* will be formatted to your screen size and displayed. Type `h` cmds for a list of C-PLOT's commands.

Help files reside in `$CPLOTHOME/`*help*. If *`topic`* includes a `/` character, the specified path (absolute or relative) is used to find the file.

The help-file format is described in the *help_fmt* help file. C-PLOT formats help files on the fly to fit whatever terminal screens and workstation windows may be in use. The help files can be printed using the *troff*, *ditroff*, or *gtroff* UNIX text formatters and the *head.man* and *Makefile* files in the `$CPLOTHOME/`*help_tools* directory. See the *README* file in that directory for specific information on running off printed copies of the help files.

Additional help files can be added by each site. The site can create a file named `$CPLOTHOME/`*motd* that C-PLOT will display when it starts.

cd change directory

changes the current working directory of the program, letting you move about the directory tree into the directories that contain the files you want to access.

```
cd
or
cd directory
```

Entered without arguments, `cd` changes the working directory to your home directory and prints its name as confirmation. Otherwise, C-PLOT changes the working directory to the name given as the argument and only prints a message if it cannot do that.

All subsequent subshells and subprocesses of C-PLOT will have the new directory as their initial working directory. Changing directory won't change the working directory of any subprocesses (user functions and filters) that are already running, nor will it change the directory of C-PLOT's parent process. All file names that you use in any of the commands are taken with respect to the current working directory. However, you can use the `se` command to set directories that will be searched for command files used with the `do` command and for data files used with the `gd` command if such files aren't in the current directory.

Changing the directory of a subshell created with the `u` command (described next) doesn't change C-PLOT's current directory. However, entering `u`pwd will print the name of the current working directory.

u create a UNIX subshell

lets you temporarily leave C-PLOT to perform other tasks. When you exit the subshell, you will be returned to C-PLOT.

```
u
or
u command
```

With no arguments, a subshell is created. If the environment variable `SHELL` is set to, for instance, */bin/sh* or */bin/csh*, that shell will be used. If the environment variable isn't set, the default */bin/sh* will be used.

If a command string is entered after the `u`, */bin/sh* is run to execute that single line, and the return to the plot program is immediate.

Don't try to change directory using `u cd` *direc*. This command will only change the directory of the subshell. The subshell disappears as soon as it has changed its directory and has no effect on the plot program. Use the `cd` command described above to change the current directory of the plot program.

To return to C-PLOT, exit the subshell; don't re-execute C-PLOT. When a subshell is executed, C-PLOT adds the variable `CPLOTLOCK` to the subshell environment. If C-PLOT is invoked again with `CPLOTLOCK` set in the environment, a message will appear warning of the nested invocation of C-PLOT.

Chapter 3 Getting and saving data

This chapter gives basic instructions:

- □ for entering data points from the keyboard, from files or with a digitizer
- □ for saving data to files
- □ for modifying data using the interactive editor
- □ for entering and using error-bar information

(Note that data also can be obtained from user functions. See Chapters 11 and 12.)

Commands covered

gd	get data
sa	save data points
eb	select error-bar mode

gd get data

```
gd
gd mode [filename] [+skip] [=total] [&] [@]
or
gd . [+skip] [=total] [&] [@]
```

Using *get data* and its various modes, you can enter data points from the keyboard, read data points from ASCII files, modify points in memory using the interactive editor, digitize points from the pen plotter or perform one of several other data manipulations. Up to 65,536 points can be obtained, although mode 7 will only manipulate the points resident in memory. The number of in-memory points is user configurable in the site-initialization file (see Appendix A). Also, modes 11, 12 and 14 allow an unlimited number of points to be plotted.

If no arguments are entered with the command, you will be prompted with a list of the 15 modes for generating or altering data. When appropriate, you also will be prompted for a file name.

If a file name is specified and it is not in the current directory, C-PLOT will look for the file in the directory specified by the environment variable, `CPLOT_GD_DIR` That directory can also be set with `set gd_dir` (see Chapter 6). `CPLOT_GD_DIR` may contain a colon-separated list of directories, in which case C-PLOT will look for the file in each directory in turn.

Get-data options

Mode	What it does
1	Enter data from the terminal keyboard
2	Take data from ASCII file
3	Same as 2, but with columns specified
4	Break current data around points in the file
5	Same as 4, but with columns specified
6	Reuse current data points
7	Modify current data points
8	Same as 1, but with columns specified
9	Digitize data from the pen plotter
10	Switch *x* and *y*
11	Take data from file with no limit on number of points
12	Same as 11, but with columns specified
13	Take data from a binary file
14	Same as 13, with no limit on number of points
15	Erase current data

File format; specifying columns

Get data modes 2, 3, 4, 5, 11 and 12 take data from ASCII files. Modes 1 and 8, which read from the keyboard, obey similar rules.

C-PLOT interprets columns as sequences of characters (not necessarily numbers) separated by any number of space or tab characters. For *get-data* modes 1, 2, 4 and 11, the default column ordering puts *x* and *y* values in the first two columns. For 2D plotting the next columns are assigned to *x* error bars (represented as *r*), *y* error bars (represented as *s*) and pen control (represented by *p*), but these columns will be scanned for values only if the respective features have been turned on using the *error-bar* or *line-control* commands. (See `eb` below and `lc` in Chapter 5.) In 3D mode, the first three columns are for *x*, *y* and *z* data. Optional columns for *z* error bars (also represented as *s*) and pen-control information follow.

Under *get-data* modes 3, 5, 8 and 12, you specify the columns from which C-PLOT will take data values. If you enter a value of 0 for the *x*, *y* or *z* column, the current value for each point will be retained. If you enter a negative value for the columns for *x*, *y*, *z* or all three, the index number of the point (starting at 0) will be assigned rather than a number from the file. There may be up to 2,048 characters on an input line; any additional characters are discarded. There is no limit on the number of columns as long as the data is within the first 2,048 characters of the input line.

Comments in data files

Rows in data files beginning with a # are considered comments and are not scanned for data. If a row begins with `#%`, the text that follows will be printed on the screen as the file is scanned. This lets you include explanatory notes that will be displayed each time the file is used with C-PLOT. Comment lines are included in the line count used for the *skip* option described below.

Modes 1 & 8: Entering data from keyboard

Under *get-data* mode 1, you will be prompted to type in values for *x* and *y* and, optionally, *r*, *s* and *p* in 2D mode and *x*, *y* and *z* and, optionally, *s* and *p* in 3D mode. Error bars may be entered but will be read only if the error-bar mode has been turned on using the `eb` command or, if in mode 8, a positive number is entered for the error-bar column. You can enter line-control information only if you have turned the mode on with the `lc` command.

It is unlikely you would need to choose columns when you are typing in data at the keyboard, but mode 8 may be useful when constructing command files or if you want the *x*, *y* or *z* value to be simply the index number of the point (by entering a negative column number).

Modes 2 & 3: Reading data

Under mode 2, data will be read into the program from ASCII files —created independently or using the `sa` command described below —containing columns of *x* and *y* (in 2D mode) and *z* (in 3D mode), plus, optionally, *r* (in 2D mode), *s* and *p*. Error bars will be read under mode 2 only if the error-bar mode has been turned on using the `eb` command. Line-control information will be read only if line-control mode has been turned on using `lc`.

Under mode 3, you instruct C-PLOT which columns in the file to use to obtain values for *x*, *y*, *z*, *r*, s and *p*. If the columns for the error bars are set equal to 0, no values will be read in for *r* or *s*. If a nonzero value is entered for an error-bar column, values will be read in but error-bar mode will not be turned on. To enter a column for line-control values, you must first have turned on line-control mode with the `lc` command.

Typing a `^C` while reading from a file will abort the read, and the number of points that were read will be reported.

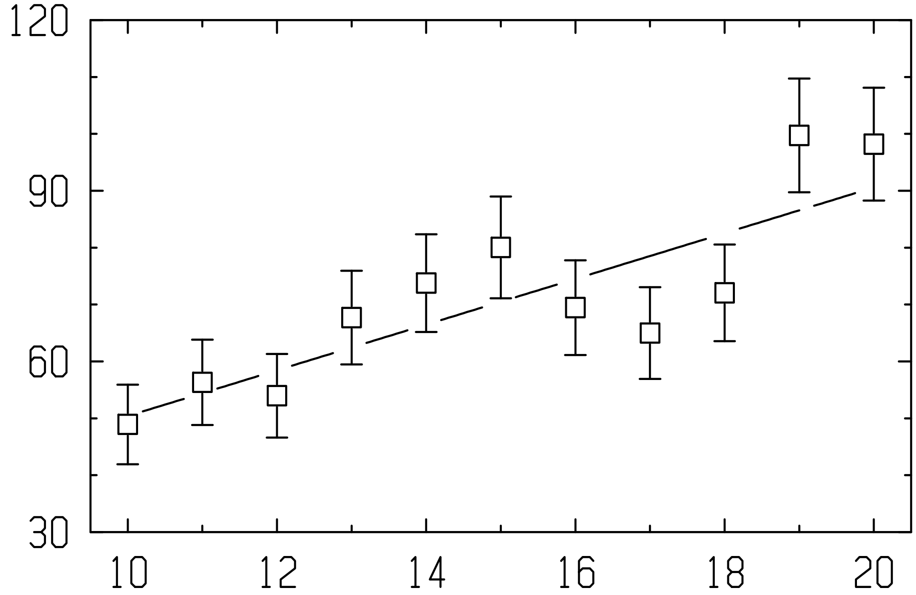
Modes 4 & 5: Break lines around points

This mode is useful if the current data is a dense set of points to be drawn as a line and you wish to break the line around a sparser set of points drawn as discrete symbols. This may be the case, for instance, when you draw a fitted or theoretical curve through data.

Assume, for example, that you have drawn the axes and data points from *datafile*, and the current data is to be the smooth curve, perhaps generated from a user function (see Chapter 11). If you type

```
gd 4 datafile
```

points "near" the sparse data points of *datafile* won't be drawn when a point-drawing command such as `pp` is executed with a line symbol, as in the following example.



The actual distance used to define a nearby point is proportional to the character size of the symbol (see `cs`, Chapter 6). To increase the gap around the symbols, increase the size of the symbol width before entering `gd 4`. If the error-bar mode is on when the points are read, the line also will be broken about the error bars.

This mode does not interpolate between points. You must have a large number of points for the breaks in the line to be symmetric.

Mode 5 is the same as 4, except that you specify columns for *x*, *y* and, optionally, *r* and *s*.

These modes work by assigning line-control information to each point. However, the user-toggled line-control mode is turned off. Although the data will be drawn using the line-control information, that information will not be used by other commands such as `sa` or `gd 7` If you do turn line-control mode on (or back on) with the `lc` command, the line-control information obtained from modes 4 or 5 will be available for other commands.

Note that the symbol filling available on many filters can be used to produce a similar effect by drawing open symbols over solid lines.

Mode 6: Reuse current data points

Mode 6 is useful after reading in data with modes 11 or 12. It keeps the current data in memory, but prevents the program from reading more points from the open-ended file when you draw the plot.

Mode 7: Modify points

Under mode 7, the values for each data point in memory will be listed in turn, and you may alter, insert or delete points. Error-bar and line-control values will be displayed for editing only if the corresponding mode is on. Only the in-memory points may be modified, and if modified, the total number of points will be truncated to the number of allowed in-memory points as set in the initialization file (see Appendix A).

The commands for modifying points are shown in the following table. The notation `[v]` means that entering a numerical value is optional for that command.

Point editing commands

Command	What it does
num `g` or `<backspace>`	Go to point number *num*
`[v] <return>`	Change data to *v*, go to next value
`[v] f`	Change data to *v*, go to next point
`[v] b` or `\`	Change data to *v*, go to prior point
`[v] c` or `<linefeed>`	Change data to *v*, update display
`G`	Go to last point
`a`	Append a point after current point
`i`	Insert a point before current point
`d`	Delete current point
`^D`	Exit
`^C` or `<break>`	Exit without using changes

When you enter `<return>` to go to the next value, you will move from *x* to *y* to *z* (in 3D mode) and, if error-bar and/or line-control modes are on, to *r*, *s*, z and/or *p*, then to the next point. All the other commands move you directly to another point.

If there is no data present, the number of points is set to one. If you delete the last point with `d`, you are returned to the `PLOT->` prompt. The values of points inserted with `i` or `a` are initialized to zero.

PseudoGraphics interactive point editing

You can use mode 7 to interactively modify 2D data points that are displayed in PseudoGraphics, using the commands described in the table above. The data is displayed on the video terminal with the current point highlighted.

The following chart shows the commands for interactive editing with PseudoGraphics.

Key	Arrow key	Action
A	Up	Turn on interactive mode.
B	Down	Turn off interactive mode.
C	Right	Scan forward through points. Press any key to stop. (Unavailable on some systems.)
D	Left	Scan backward through points.

You can edit at one time only as many points as are held in memory, a value set in the initialization file (see Appendix A).

Mode 9: Digitize from pen plotter

Mode 9, available only in 2D mode, lets you use the pen plotter as a digitizer. If possible, the digitizing site (available from Hewlett-Packard, part number 09872-60066) should be installed in the plotter to make it easy to align the pen carriage over the desired points. To digitize, position the site over the appropriate points using the plotter's front-panel controls. Press the *enter* control on the plotter to send a point to the computer.

The computer will first ask you to set three scaling points that will be used to translate the plotter's native coordinates to the coordinates of the plot you are digitizing. Since there are three scaling points, it is not necessary for the axes of the points being digitized to be perfectly aligned with the plotter motions. Choosing three points that are widely separated gives the least error when C-PLOT calculates the transformation factors. When setting the scaling points, move to each one using the plotter control panel and press the *enter* control. The computer prints out the plotter coordinates and asks you for the equivalent coordinates on your plot.

After setting the three scaling points, site each point to be digitized in turn and press the *enter* control on the plotter. As each point is digitized, your coordinates appear on the terminal. You don't have to type anything else at the keyboard until you are done. To finish, type `^C`.

If line-control mode is on while digitizing, the pen up or down status at the time you press the *enter* control is stored with each point.

Mode 10: Switch *x* and *y*

In 2D mode, values for *x* and *y* and for *x*- and *y*-error bars are switched. In 3D mode, only *x* and *y* values are switched.

Modes 11 & 12: Read unlimited data from files

These modes let you plot an unlimited of number of data points with only one call to `gd`. When you enter `gd 11`, a first set of points is read into memory. The number of points read is the number of in-memory points as set in the site-initialization file (see Appendix A). These points will be drawn when you plot the points or the error bars using the `p`, `pz`, `z`, `zz`, `pp`, `zp`, `pb` or `zb` commands. The next set of points will then automatically be read from the file and plotted, and so on, until the file is exhausted or the limit set with the `=`*`total`* option (described below) is reached. At the conclusion of the point plotting, the points in memory will be the last points read. However, if you plot the points again, the first points in the file will be read and all the points will be plotted.

All other commands that use data points will only use those currently in memory after reading in data with these modes.

If you enter `gd 6`, the current points in memory will continue to be available, and only those points will be drawn when you enter the commands to draw points.

Typing a `^C` while reading from the file, either during the `gd` command or while drawing the points, will abort the read, and the number of points successfully read will be reported. A subsequent command to plot the data points will still read the entire data set from the file and plot all the points.

Modes 13 & 14: Read data from binary files

These modes allow points to be plotted from binary files, the fastest way to read data into C-PLOT. The format of the file is given by the struct `pt` given in the include file *p_plot.h* That structure is:

```
struct  pt {
        int     p_flags;    /* flags for this point */
        float   p_d[4];     /* data */

};
```

Presently, `p_flags` contains line-control status in the low-order two bits. The elements in `p_d[4]` hold *x*, *y*, *x* error bars and *y* error bars in 2D mode. In 3D mode they hold *x*, *y*, *z* and *z* error bars.

The general gd command options, described below, can be used with these modes.

Mode 14 reads from an indefinitely long file, like modes 11 and 12.

Mode 15: Erase current data

You may wish to erase the current data before calling user functions that will ignore the current data. Erasing the current data does not change the current ranges.

Options +*skip*, =*total*, &, . and @

The +*skip* and =*total* options let you select particular windows of data from your file. They also can be used to read in sections of a file for editing under mode 7. *Skip* is an integer telling the program how many lines in a file to skip before starting to read in data points. When skipping lines, each line in the file is counted, whether it contains valid data or not. *Total* specifies the maximum number of points to be read from the file.

The & argument causes the data points being obtained to be appended to the current data points.

The @ argument specifies real-time plotting. When you plot the points, C-PLOT will first draw any data already in a file. It will then continue to check the file to see if more points have been added, plotting them as they appear. If the end-of-file character (ASCII 04) or a ^D are read from the file after a newline, the program stops reading the file and proceeds to the next command (if running from a command file) or to the PLOT-> prompt (if running interactively.) You also can use a ^C to interrupt the reading. In the present implementation, the program sleeps for one second between checks for new data.

The optional arguments & and @ are typed after the file name and separated from the file name by space.

Entering gd . tells C-PLOT to get data using the same mode and file name as before. For the modes with specified columns (3, 5, 8 and 12), you will still be prompted for column numbers. You can use . to keep the same file name when switching to a different mode. After entering gd 2 *filename*, for example, you can enter gd 3 . to indicate you want to use the same file, but this time you want to specify the columns.

You can use the & and @ options with the . option. For instance, gd . +1024 will begin reading at the 1025th line in the current data file. If you followed that with just gd ., the first points in the file will be read.

The `.` option does not repeat modes 6, 7, 10 or 15.

sa save current data

lists the current data points on the screen or, if specified, to a file or device.

```
sa
or
sa [-b] filename [a or w]
```

You may, for example, wish to save data to a file if you have entered data using the digitizer or the keyboard, if you have created or modified data with a user function (see `fn`, Chapter 11), if you have modified data using the data editor, `gd 7`, or if you wish to save data in binary format.

If no parameters are given with the command, *save data* lists the values of the current data on the screen. If 3D, error-bar and/or line-control modes are on, *r*, *s*, *z* and/or *p* values are listed, as appropriate, in addition to *x* and *y*.

`Filename` is the path name of a file or device to which to write the data points. If the file already exists, you will be asked whether you want to write over the current contents of the file or add the data to the end of the file. Type `<return>` if you don't want to do either. You also can choose on the command line to write over or append the file by putting an `a`, for append, or a `w`, for write over, after the file name.

A `-b` before the file name will cause the data to be written to the file in a binary format suitable for reading back with `gd` modes 13 or 14. Binary data files can be read in and written out faster than ASCII files and are generally smaller.

Save data will abort if there is a write error to the file or device.

eb select error-bar mode

turns error-bar mode on and off.

```
eb [x|y|z] [0|1]
```

Entered without parameters, `eb` toggles error-bar mode on or off. In 2D mode it toggles *y*-axis error bars and in 3D mode *z*-axis error bars. In 2D mode, `x` or `y` can be given as arguments to toggle the respective error bars off or on. An argument of `0` or `1` will turn error bars off or on, respectively.

When a `?` is entered with `eb`, C-PLOT will indicate which error-bar modes are on and which are off.

When switching between 2D and 3D plotting, the *y* and *z* error-bar modes will reflect each other's states. For example, if *y* error bars are on in 2D mode, *z* error bars will be on in 3D mode, and vice versa.

Although a file may include columns for error bars, C-PLOT will not read them unless the error-bar mode has been turned on or a nonzero column for the error bars has been specified under gd modes 3, 5, 8 or 12.

When the error-bar mode is on, error bars are scanned in gd modes 1, 2 and 11; error-bar values are displayed in gd mode 7; error-bar values are included in the *save data* command, sa ; and error bars are drawn when you enter the *draw plot* commands, p , pz , z or zz .

When the error-bar mode is on, the automatic data-ranging will include the error bars in the range settings when the first data set is read or when the *new points* command (np , see Chapter 5) is used. Data masking with gd modes 4 and 5 will take error-bar lengths into account when doing the masking.

Chapter 4 Using PseudoGraphics

In this chapter you will find instructions for using PseudoGraphics —fast, low-resolution displays that work on most video terminals. Note that the PseudoGraphics feature is available only for 2D plotting.

Commands covered

v	draw PseudoGraphics plot
va	draw PseudoGraphics axes
vp	draw PseudoGraphics points
vb	select inclusion of PseudoGraphics axes
vt	select automatic drawing of PseudoGraphics
er	erase the video screen
gr	select PseudoGraphics terminal type

v draw PseudoGraphics plot

uses the alternate characters available on many terminals (or else standard ASCII characters) to quickly draw a low-resolution version of the plot, with axes, range settings and points.

```
v
```

The `v` command erases the video screen and draws a bare-bones display of your plot. The values of the tick numbers and the positions of the major tick marks are shown as they will appear on the plot. However, symbol, text and most other formatting features described in Chapters 5 and 6 are not available with PseudoGraphics.

va draw PseudoGraphics axes

draws the plot axes without displaying points.

```
va
```

The plot axes will be drawn with ranges set according to the `ra`, `ro` or `np` commands. The screen is not erased before the axes are drawn, and the cursor is put in the upper-left corner of the screen after the axes are drawn. This command is the quickest way to see how the axes will appear on the final plot.

vp draw PseudoGraphics points

overlays the screen with the current data points.

```
vp
```

Just the points will be drawn on the screen. The screen is not erased before the points are drawn, and the cursor is put in the upper-left corner of the screen after they are drawn.

vb select inclusion of PseudoGraphics axes

lets you instruct C-PLOT to draw only the points and not the box formed by the axes when the v command is entered.

```
vb
or
vb state
```

With no parameters, the inclusion option is toggled. If *state* is a 0, only the points are drawn with v. If *state* is a 1, both the points and axes are drawn.

vt select automatic drawing of PseudoGraphics

causes a plot to be drawn automatically after certain commands are executed.

```
vt
or
vt state
```

With no parameters, the automatic drawing option is toggled. If *state* is 0, automatic drawing is turned off. If *state* is 1, automatic drawing is turned on. When the option is enabled, the PseudoGraphics plot will be drawn whenever the np, gd, fn, f1, f2 or f3 commands return without an error.

er erase the video screen

clears the video screen.

```
er
```

The er command will work only if the terminal variable exported by the shell (or set in the site-initialization file) or the terminal type set with the gr command (described next) and the corresponding entry in the terminal-capabilities data base correctly describe your terminal (see Appendix A).

gr select PseudoGraphics terminal type

lets you identify the terminal in use so that C-PLOT can send the proper control codes to erase the screen and draw PseudoGraphics. You also can select the proper alternate character set for PseudoGraphics.

```
gr
gr terminal_name
or
gr terminal_name character_set
```

C-PLOT normally obtains the terminal name from the environment exported by the shell, and you won't need to use this command. However, if the name from the environment is wrong or absent, you can use this command to set the terminal type from within C-PLOT. You also can use the `gr` command to select a PseudoGraphics character set independently of the terminal type.

The *`terminal_name`* option determines which control codes are sent to the terminal to erase the screen and position the cursor for drawing PseudoGraphics. The *`character_set`* option determines which characters are used to draw the PseudoGraphics axes and points. For some of the more common terminal types, the terminal name and character set name are the same.

Without an argument, `gr` lists the names associated with each of the built-in PseudoGraphics character sets, prints the current terminal name and character-set name and prompts you for new names. Entering `<return>` retains the current names. Entering one name (or one argument after `gr`) selects a new terminal type. If the name matches one of the character sets, that set is selected also. If you enter a second name (or argument) and it is a valid character-set name, that set is selected. An invalid name selects the dumb set.

Refer to Appendix A for details on PseudoGraphics implementation, including a list of valid character-set names. Valid terminal names are system dependent. On BSD UNIX systems, they are in the file */etc/termcap*. On System V installations, look in the subdirectories of */usr/lib/terminfo*.

Chapter 5 Designing the Plot

In this chapter you will find instructions for:

- □ changing the format of the plot, including selecting the plotting symbol and plot window
- □ selecting the axis type
- □ choosing between linear and logarithmic axes
- □ 3D plotting

Commands covered

2d	select 2D mode
3d	select 3D mode
lc	select line-control mode
ra, **ro**	select axis range/range options
np	reset axes for new points
re	reset program for new plot
tu	turn plot
ty	select type of plot
st	select plot style
sw	swivel plot
wi	select size and location of plot window
lo	locate plot
bo	set 3D box ratios
vi	set 3D view point
tw	tweak plot orientation

2d select 2D mode

changes plotting to 2D mode.

```
2d
```

The `2d` command switches C-PLOT to 2D mode, which is the startup condition. In 2D mode, the axes are drawn and the data is plotted in a 2D window.

Also in 2D mode, error bars are available for both the *x* and *y* data.

3d select 3D mode

changes plotting to 3D mode.

```
3d
```

The 3d command switches C-PLOT to 3D mode. In 3D mode, the axes are drawn and the data is plotted using a 3D perspective box.

The commands vi, to set the view point of the 3D box, and bo, to set the relative lengths of the 3D box, are available in 3D mode. Both are described below.

The absolute size of the 3D box is the largest that just fits in the 2D window set by the window command, wi, without distorting the aspect ratio of the box.

Three arguments are required when position arguments are used with the commands to draw annotation or a key, pn, zn, pk or zk. The units of the position arguments are data units, as in 2D mode, or box units, as used with the bo command.

In 3D mode, the gd 9 command (digitize) and the gd 4 and gd 5 commands (break data) are not available. The other gd modes and the sa command work with three columns of data, plus optional error-bar and pen-control values. Note, however, that only *z* error bars are available in 3D mode.

In 3D mode, the commands ra, ro, np, ty and tx take extra arguments and/or prompt for extra values associated with the *z* axis.

In its current state, 3D functionality has some limitations. Circle symbols are drawn as flat 2D circles in 3D mode. Axis numbering and annotation are in fixed planes and orientation.

lc line-control mode

allows multiply segmented lines to be drawn from a data set and/or the filling of shapes formed by consecutive data points.

```
lc
or
lc state
```

Line-control mode lets you draw multiply segmented lines from a single data set, or draw filled shapes. Line-control information instructs the program whether to move to a point with the pen up or down while drawing with one of the line symbols. (With graphics filters, line-control information instructs the filter whether to draw continuously or not.)

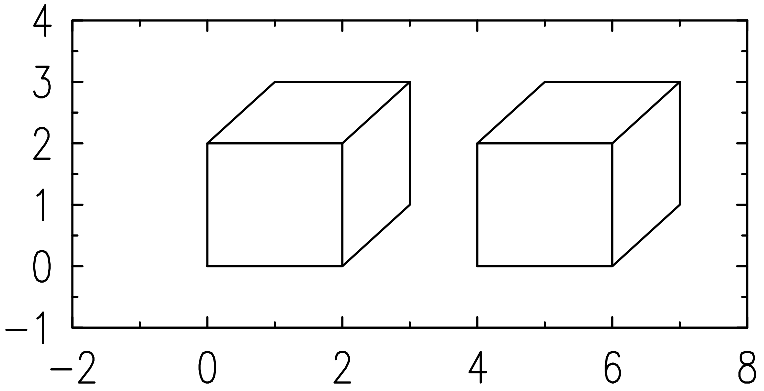
Line-control information is associated with each point. A 0 (or blank) in the line-control column of a file provides continuous drawing. A 1 instructs the program to move to that point with the pen up. (The first point is always reached with pen up.) A `2` in the line-control column also means to move to the point with pen up, but the figure formed by subsequent points with `0` line control will be filled with the current “white” fill color.

Entered without parameters, `lc` toggles line-control mode on and off. If the command is entered with a `0` as an argument, line-control mode is turned off. Entered with a `1`, it is turned on.

For example, with line-control mode on, the following 24 data points produce the displayed shapes:

Line control

```
0 0 1       4 0 1
2 0         6 0
2 2         6 2
0 2         4 2
0 0         4 0
0 2 1       4 2 1
1 3         5 3
3 3         7 3
3 1         7 1
2 0         6 0
2 2 1       6 2 1
3 3         7 3
```



Line-control information can be read using the *get data* command and is saved with the *save data* command. Although a file may include a column for line control, it will not be read in unless the line-control mode has been turned on.

`ra`, `ro` select axis range, select range options

C-PLOT sets the axis ranges and the positioning and style of the tick marks when the first set of data is read. It will maintain these parameters until `np` is executed, the ranges are changed using `ra` or `ro`, or the *reset* command, `re`, clears the current data.

Select axis range lets you set minimum and maximum ranges for each axis. *Select range options* lets you choose the style of the axis tick marks in addition to setting the range.

```
ra
ra x|y|z
ra xmin [xmax [ymin [ymax [zmin [zmax]]]]]
ra y ymin [ymax]
ra z zmin [zmax]
ro
or
ro x|y|z
```

When `ra` is entered, C-PLOT prompts for minimum and maximum values for each axis. The current minimum and maximum are shown in parentheses. Enter `<return>` to maintain the current values. If you specify a single axis by entering `x`, `y` or `z` as an argument, you will only be asked to range that axis.

If you only range one axis and there is data present, you will be asked if you want C-PLOT to reset the ranges on the other axes. If so, the program will reset the other axes so they include only the range of points corresponding to the plot section you chose for the first axis range.

If you specify one to six numerical arguments, they will be taken as, in order, the minimum and maximum values for the *x*-axis, then the *y*-axis, then the *z*-axis. You also can give the argument `y` and specify one or two values for the *y*-axis range or the argument `z` and specify one or two values for the *z*-axis range. You won't be prompted for other information when you give numerical arguments.

C-PLOT ordinarily automatically determines the positioning and spacing of the tick marks and the numbering of each axis. You can control these parameters, however, either by using the `ty` command (see below) or by setting them with the `ro` command.

Entered without an argument, `ro` will prompt for the range minimums and maximums for all axes. If you specify `x`, `y` or `z`, you will be prompted only for that axis. In addition, `ro` will prompt you to choose among three other options:

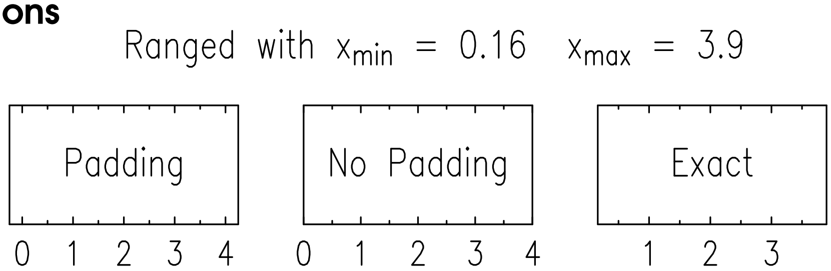
□ **Exact Ranges**. The axis will begin and end at the specified minimum and maximum. The tick spacing and numbering will be decided according to C-PLOT's standard algorithm. The default is off.

□ **Axis Padding**. The first and last tick marks will be moved in slightly from the end of the axis if the minimum or maximum range value is too near to or coincides with the first and last tick positions. Axis padding cannot be used with the exact-ranges option. The default is on.

□ **User-Defined Tick Spacing**. The first and last tick marks and the corresponding axis numbers will coincide with the minimum and maximum range values. With linear axes, you specify how many intervals are to be used and how many intermediate tick marks are to be placed in each interval. With logarithmic axes, you specify how many major intervals between numbers and the number of intermediate tick marks. The latter is rounded down to eight, two or zero. The default is off.

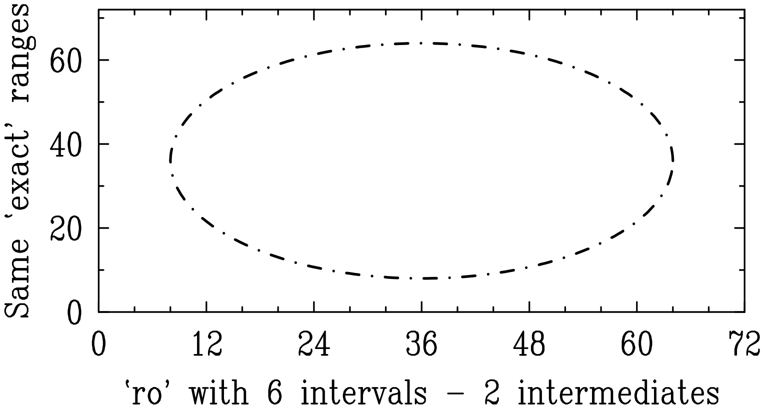
The next example shows the effect of axis padding and exact ranges:

Range options



The next example illustrates user-defined tick spacing:

Tick spacing



np reset axes for new points

sets new axis ranges based on the current set of data.

```
np
or
np x|y|z
```

Once the data ranges are set, automatically with the first data, explicitly using the `ra` or `ro` commands, or as part of a reset with the `re` command, those ranges remain in effect when new data is entered. The `np` command will reset the axis ranges to encompass the maximum and minimum values of the current data. If error-bar mode is on, the ranges are set to include the error bars.

With an argument of `x`, `y` or `z`, `np` will reset the range only of the axis specified.

If the plot type is set for a logarithmic axis, C-PLOT ignores points less than or equal to zero when doing the automatic ranging.

The *new points* command will clear any range specifications set with the `ra` or `ro` command. If there is no data present or if all the *x*, *y* or *z* values are the same, an error message is printed.

re reset program for new plot

automatically restores several program parameters to their start-up condition.

```
re
```

Reset will restore the parameters listed below to their original condition.

Command	Description	State
cs	Character size	Set to defaults
gd	Points	Number of points set to 0
gk	Key	Clear key
ra	Range	Clear axis ranges
ro	User ticks	Clear user settings
sy	Symbol	Set symbol to a dot
tu	Plot orientation	Landscape
tx	Text	Clear label and title
ty (and ro)	Plot types	Set all type flags to 0
wi	Window	Set to selection code 0
vi	3D view point	Set to 1.3 −2.4 2
bo	3D box	Set to 1 1 1
st	Plot style	Set to 0
lo	Location	Set to 0 0 1 1
se	Set options	Dash length 1mm Line spacing 1.5 × text height Tick length 1.5%

It is possible to retain axis-range settings following a reset. The axes-range values last used are actually remembered after a reset. If you enter the ra command before reading in new data, those values become the default values and can be selected by simply entering <return>. If you don’t enter the ra command, the program behaves as if the ranges haven’t been set.

The current font, filters, active user functions, command-file directory, get-data directory and user-function directory are not changed by the re command.

tu turn plot

determines whether the plot is drawn in landscape or portrait mode. (Landscape is the default.)

```
tu
or
tu state
```

If no parameter is used, the plot orientation is toggled. An argument of `0` puts the plot in landscape mode. If `1` is given as the argument, the plot is drawn in portrait orientation.

C-PLOT will try to maintain the window dimensions chosen with `wi`. If a window won't fit when the plot is turned, the default window dimensions are used and an error message is printed.

Plots displayed using PseudoGraphics are not affected.

ty select type of plot

lets you control many details of how the plot looks, such as whether to use linear or logarithmic axes and how to number and put tick marks on the axes.

```
ty
ty x_type y_type overall_type
or
ty x_type y_type z_type overall_type
```

The `ty` command allows you to format plots interactively or by entering a simple set of codes. C-PLOT encodes the format for a plot in three or four numerical flags. The first flag describes the *x* axis. The second controls the *y* axis. The third, in 3D mode, controls the *z* axis. The last selects overall plot features. There are three ways to select values for the flags: If you know these numbers you can enter them as arguments to the `ty` command. If you know the numbers but don't remember the order in which to enter them, `ty` will prompt for them one at a time. Finally, if you don't know the appropriate numbers, you can select one feature at a time as prompted. C-PLOT will display the encoded numbers after you have made your format choices so you can select the same features more quickly next time. (Instructions for working with the coding scheme are given below.)

The two tables that follow list features controlled by `ty`. The value column gives the numerical code for the alternate mode shown for each feature. (The default mode in each case has a value of zero.) The octal column is an alternative notation for each value. Hexadecimal notation also is recognized (see below). The first table presents the options you can select that control the overall plot type.

Axis type

Usual mode	Alternate mode	Value	Octal
Automatic tick spacing	User tick spacing	1	01
Use normal auto-ranging	Entered ranges exact	2	02
Can move in end ticks	Don't move ticks	4	04
Use linear axis	Use logarithmic axis	8	010
Number axis	Don't number axis	16	020
Scientific notation	Engineering notation	32	040
Log axis decimal optional	Power-of-ten notation	32	040
Use trailing zeros	No trailing zeros	64	0100
Use leading zeros	No leading zeros	128	0200
Print all axis numbers	Don't print first number	256	0400
Draw tick marks	Don't draw tick marks	512	01000
Ticks inside axis	No ticks inside axis	1024	02000
No ticks outside axis	Ticks extend past axis	2048	04000
Dual-height ticks	Uniform ticks	4096	010000
Normal tick marks	Tick marks form a grid	8192	020000
Draw axis and numbers	Don't draw them	16384	040000

Note that the first three axis modes above —user-defined tick spacing:exact ranges and axis padding —also can be selected using the `ro` command. Parameters for user-defined tick spacing normally are entered using the `ro` command. If selecting plot features interactively, you will be prompted for user-defined tick-space settings.

Overall plot type

Usual mode	Alternate mode	Value	Octal
Draw a complete box	Just draw x and y axes	2	02
Put ticks all around	No ticks left and right	4	04
Cut off plot symbols	Let symbols overlap axes	8	010
Drop out-of-range points	Draw them on axes	16	020
Don't draw border	Draw border around edge	32	040
[ ] enclose units	( ) enclose units	64	0100
Y-axis label ticks on left	Draw them on right side	128	0200
Draw left and right y-axis	No right-side y-axis	256	0400
Draw left and right y-axis	Draw just right-side y-axis	512	01000
Traditional axis labels	APS-style labels	1024	02000

Plot type 256 is allowed only when type 128 is not selected. Plot type 512 is allowed only when type 128 is selected.

Selecting overall plot type 128 places the *y*-axis label and tick numbers on the right side of the plot. By itself, this plot type is useful if the plot only has one type of *y*-axis numbering.

To have different numbering and labels on each side of the plot, first enter labels and ranges for the left side, select overall plot type 256 and draw the plot. Plot type 256 prevents the right side axis from being drawn. Next select overall plot type 640 (128+512), enter the ranges (`ra`) and labels (`tx`) for the right-side axis, and draw the axis and labels. Plot type 128 places the label and numbers to the right, and plot type 512 draws only the right-side axis.

APS-style labels (1024) have parentheses around the units and no multiplication sign in front of the scale factor. For dimensionless quantities, the inverse of the scale factor is placed in front of the label with no parentheses.

Setting values

The values in the tables above select the respective alternate modes for each plot feature via the three (or, in 3D mode, four) plot-type codings, which are normally entered as arguments on the command line. Each of the codings —one for the overall plot and one for each axis —is the sum of the values associated with the alternate modes in the tables. A zero for any feature chooses the default mode, so you only include values for those alternate modes you want to select.

Entering 1040 for an axis, for instance, selects no numbering (16) and no tick marks inside the axis (1024). Other features are set to the usual mode. Entering a zero means all the usual modes are used. If you enter `.` for a plot-type number, C-PLOT will use the previous value for that plot type and not prompt for additional information.

You can turn off a feature by entering its value preceded by `+` or `-`. A `+` turns on the alternate mode. A `-` restores the usual mode. For instance, entering `ty . +8 .` turns on logarithmic axis mode for the *y*-axis without affecting any previously selected alternate mode.

You can code values for the three plot-type numbers in decimal, octal or hexadecimal. For octal, precede the number by `0`. For hexadecimal, precede the number by `0x`. C-PLOT will show the coding for each set of features using the most recent coding type.

`st` select plot style

selects plot style.

```
st [code]
```

The `st` command selects from several defined plot styles. The default style in both 2D and 3D mode is style 0.

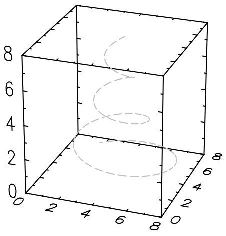
In 2D mode the following styles are available:

Code	What is drawn
`0`	All four sides
`1`	Only center lines through zero

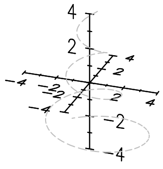
In 3D mode the following styles are available:

Code	What is drawn
`0`	All six sides of the cube
`1`	Only center lines through zero
`2`	Only bottom and two back sides of the cube
`3`	Only the bottom of the cube
`4`	Only the three numbered edges

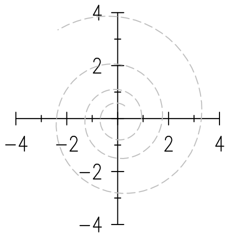
Here are examples of the selected styles:



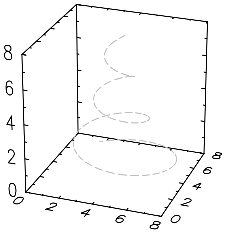
style 0



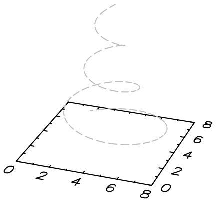
style 1



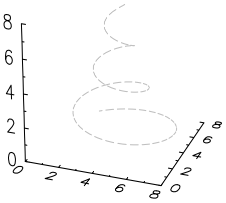
2D style 1



style 2



style 3



style 4

sw swivel plot

sets the plot swivel in the plane of the page.

```
    sw [angle]
```

The sw command rotates the 2D or 3D plot in the plane of the page. The units of *angle* are degrees. The 2D plots are distorted as they are rotated; 3D plots are not.

wi select plot window

lets you place plots anywhere on the page and lets you draw them in any size rectangle that will fit on the page. You can put multiple plots on a single page by choosing suitable windows for each.

```
    wi
    wi #
    wi width height
    wi vert_offset width height
    or
    wi horz_offset vert_offset width height
```

With no arguments and with no pen plotter initialized, wi will display the current window size and position. A single argument will set the plot size according to the following coding:

Selection code	Window size
0	20cm x 15cm (the default)
1	15cm x 15cm
2	15cm x 12.5cm
3	12.5cm x 12.5cm
9	The entire available area

Two arguments will specify the width and height of the plot. The first argument is the width, the second the height.

If there are three arguments, the first value represents the distance from the bottom of the drawing area to the bottom of the plot window. The second and third arguments are the width and height. The plots are drawn centered horizontally on the page.

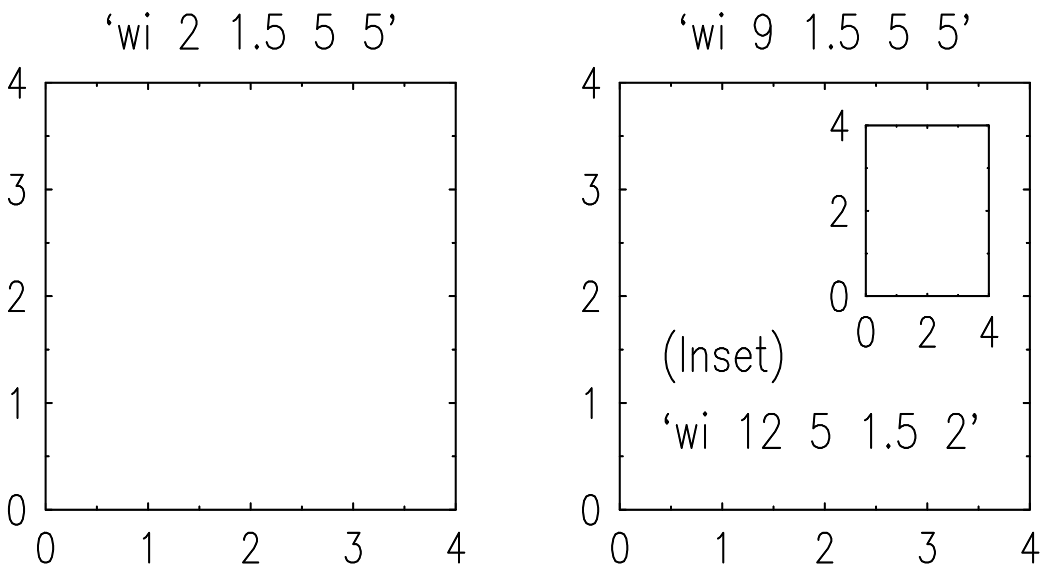
If there are four arguments, the first value is the distance from the left edge of the drawing area to the left edge of the plot window. The second value is the distance from the bottom of the drawing area to the bottom of the plot. The third argument is the width of the plot and the fourth argument is its height.

If `vert_offset` or `horz_offset` are negative, they represent offsets from the top or right side of the plot window to the top or right side of the drawing area. The offset argument –0 places the plot window adjacent to the corresponding edge of the drawing area.

Select plot window, like *annotate* and *draw key* (see `pn` and `zn`, `pk` and `zk`, Chapter 6), uses pen-plotter centimeter units. The units correspond to actual centimeters on the pen plotter. Centimeter units on the graphics filter device will only be exact if you have selected the appropriate scaling factors using the `sc` command (see Chapter 9).

Using four arguments to `wi`, you have complete control over the position and size of the plot and can easily place multiple plots on a page, as shown in the following example:

Window sizing



In 3D mode, the 3D plot is scaled to fit the 2D window. The aspect ratio of the 3D plot is not changed, however. Generally, two points on the 3D axis will touch the 2D window in one direction while the 3D plot is centered in the 2D window in the other direction.

Using a pen plotter

You can set the plot window with the pen plotter’s controls. With no arguments to `wi`, and if the pen plotter is initialized, you are asked to set the size of the window using the plotter controls. Enter `<return>` to use the current window.

Otherwise, set the plotter's scaling points (P1 and P2) to two corners of the desired plot window according to the pen plotter manual, then enter `<return>`. The current window size will be displayed.

Different models of pen plotters have different sizes for their available plotting areas. Before the plotter is initialized, the size that C-PLOT assumes for a plotting area is set to match that of the HP7440A plotter with A-size paper, 19.125×25.75 centimeters (10.14×7.53 inches). The position of the window on the page is set with respect to this page size. If a pen plotter with a different plotting area is initialized, C-PLOT will use that area. The relative position of plots drawn using different plotter areas but the same arguments to `wi` will vary. Since the window sizes and position (along with character sizes) are specified in pen-plotter centimeter units, the actual size of plots will not vary.

Unless the `sc` command has been used to select filter scaling factors, the current plotting area will be mapped to the entire graphics-filter plotting area, and the aspect ratio between the axes will not be maintained.

Once you initialize the plotter, C-PLOT remembers that plotter's available area, even after you release it (see `rp`, Chapter 7). Don't spend too much time fine-tuning the look of a plot using a filter device if you plan to plot it on a pen plotter (other than the HP7440A) without first initializing the plotter to read in its available area.

lo locate plot

lets you resize the plot and relocate it anywhere on the page.

```
lo
or
lo llx lly urx ury
```

The *locate plot* command rescales the entire plot to fit into the box described by the coordinates you enter. Consider the lower-left and upper-right corners of the available plotting area on the landscape page to have coordinates (0,0) and (1,1), respectively. With the *locate plot* command, you can enter new values for the lower-left and upper-right coordinates. All elements of the plot will then be drawn within the box you have described.

With no arguments, you will be prompted for the coordinate values. You also can enter the four values directly on the command line.

bo select 3D box ratios

sets the relative sizes of the 3D axis box edges.

```
bo
or
bo x|. y|. z|.
```

The axes of 3D plots can be imagined as drawn in a box, whose relative dimensions are set with the `bo` command. One corner of the box is at coordinates (0,0,0) in box units. The opposite corner is at the position specified by *x*, *y* and *z*.

The numbers you enter will be rescaled so that the longest side of the box has a length of one box unit. You can use a dot for any of the arguments to indicate that the current value should be used.

For example, the command `bo 10 10 4` will create an axis box that is shorter in the *z* direction. The length of the *x* and *y* sides of the box will be one box unit. The length of the *z* side will be 0.4 box units.

The commands that draw annotation and the key (`pn`, `zn`, `pk` and `zk`) use box units to position the text when data units are not chosen using the `u` suffix.

vi select 3D plot view

sets the 3D view point.

```
vi
vi x|. y|. z|.
or
vi d
```

The `vi` command sets the 3D-mode view point, which determines the perspective of the 3D plot. The view point is the position in space from which you view the center of the 3D box. The units of the view point are in the box units used with the `bo` command. Those units place one corner of the 3D box at (0, 0, 0) and the opposite corner at the position set by the `bo` command. The default view point is at (1.3, −2.4, 2).

To set the view point, you can enter the coordinates as arguments to the command, or you can just enter `vi` and be prompted for values. When entering arguments on the command line, you can enter a dot to indicate you want to use the current value for the particular coordinate.

By giving one numerical argument to the `vi` command you can change the view distance without changing the view direction.

tw tweak plot orientation

interactively rotates the 3D axis orientation

```
tw
```

The tweak command is useful for interactively setting the view point of a 3D plot. The graphics filter should be a display terminal. If the filter is associated with a printer, you will generate a plot each time you hit <return>.

Although you normally specify the orientation of the 3D axis in C-PLOT using view-point coordinates, the tw command works in terms of rotation angles. When you enter tw, the current rotation angles, the view-point radial distance and the view-point coordinates are printed, along with the four current tweak deltas. The first three deltas are the increments to the rotation angles that will be made each time you hit <return>. The last delta is the increment to the view-point distance.

If you just hit <return>, the current deltas will be added to the angles and view-point distance, the view point will be recalculated and the 3D plot will be redrawn. However, you can enter new deltas before you press <return>. Only the values you enter —up to four —will be used. You can use a dot in any of the positions to use the current value for that delta. A single minus sign in one of the positions changes the sign of the corresponding delta. Entering =*numb* sets the value of the corresponding delta to *numb*.

Type ^D to quit tweak mode and restore text mode on graphics terminals. The view-point parameters set with the vi command will be updated to reflect the current orientation.

On video terminals that use a serial line for sending both text and graphics, such as 4014 emulators, use the zq command to turn off echoing of input and the sending of output text to the terminal before entering tw. When you exit tweak mode, echoing of input and sending of output will be restored.

The tw command also functions in 2D mode, although only the first delta, which corresponds to the swivel parameter (see sw) has an effect.

Chapter 6 Adding Text

In this chapter you will find instructions for:

- □ entering and formatting text for axis labels, plot title, key and annotation
- □ customizing the size and shape of the characters and obtaining scientific, mathematical and foreign characters

Commands covered

cs	set character sizes
ft	select font
sy	select plotting symbol
gk	enter symbols and text for plot key
se	set parameters
tx	enter text for plot labels and title
yg	set gap between *y*-axis and label

cs set character sizes

lets you vary the height, width and slant of text characters and the plotting symbol. Changing the width of the symbol controls the width of error bars (see `eb`, below) and the size of data masking in `gd 4` (see Chapter 3).

```
cs
cs t|l|n|s|d|k height [ratio [slant]]
or
cs t|l|n|s|d|k
```

Character height is given in millimeters and refers to upper-case letters. The width is controlled by setting the ratio of height to width. Character slant is given in degrees. Positive values slant characters to the right, negative values to the left.

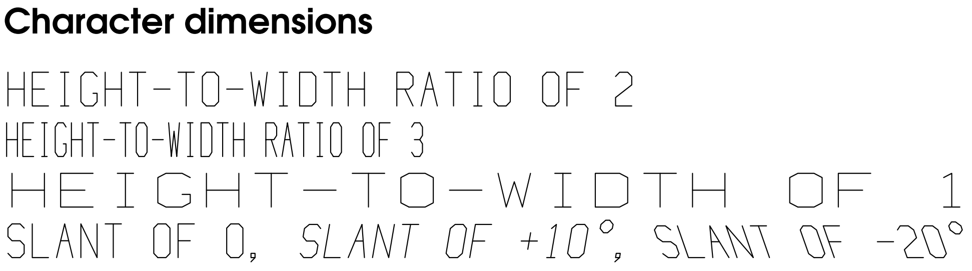
Without arguments, C-PLOT will prompt you separately to set the height, height-to-width ratio and slant for the title, axis labels, axis numbers, symbol, key, annotation and date. The current values are given in parentheses. Only the parameters for which you enter values will be changed. Entering one value changes only the height. Entering two changes the height and the height-to-width ratio. Entering three changes the height, height-to-width ratio and slant.

You also can apply the command to only one text group using the arguments `t`, `l`, `n`, `s`, `k` or `d` to select title, labels, numbers, symbol, key and annotation or date, respectively.

If you enter the parameter for one type of text on the command line, you will be prompted to set the height, height-to-width ratio and slant for that text. (Only one text type at a time can be set using this method.) Alternatively, you can place the parameters for one or more types of text on the command line along with values for height, height-to-width ratio and slant. For example, entering `cs s 6 2` sets the symbol height to 6 and its height-to-width ratio to 2.

The absolute character dimensions, given in millimeters, are always accurate on the pen plotter. The actual dimensions of characters drawn with a graphics filter vary depending on the mapping of pen-plotter units to the particular graphics device. (*Select filter-scaling factors*, or `sc`, lets you control the mapping. See Chapter 9.)

The example below shows text with various dimension settings:


Character dimensions
HEIGHT-TO-WIDTH RATIO OF 2
HEIGHT-TO-WIDTH RATIO OF 3
HEIGHT-TO-WIDTH OF 1
SLANT OF 0, SLANT OF +10°, SLANT OF -20°

At program start-up or after a reset (see `re`), characters will be drawn with their height-to-width ratio set to 2 and slant set to 0. The default title height is 5 millimeters. Axis labels and numbers, key and annotation text and the date default to 4.5 millimeters high. The default symbol height is 4 millimeters.

`ft` select font

controls the font used to draw text on the plot.

```
ft
or
ft #
```

With no parameters, a summary of the available fonts is printed along with the code for the current font, and you are prompted to enter the code for a new font. Note that fonts also can be changed within a text string with the `\f#` special-character sequence, where # is the font code.

You also can select the font by giving its code as an argument. Code numbers are shown in the following table. Font 0 is the default font.

Code	Resolution	Line width	Description	Proportional	Specimen
0	Low	Single	Sans-serif	No	123ABCabc
1	Medium	Double	Serif	Yes	123ABCabc
2	High	Single	Sans-serif	Yes	123ABCabc
3	High	Double	Serif	Yes	123ABCabc
4	High	Triple	Serif	Yes	123ABCabc
5	High	Double	Block	Yes	123ABCabc
6	High	Triple	Gothic	Yes	123ABCabc
7	High	Single	Script	Yes	123ABCabc
8	High	Double	Script	Yes	123ABCabc

C-PLOT forms characters by plotting a series of lines. The font resolution refers to the size of the grid used to design the characters. The low-resolution font, font 0, uses a grid that is six units wide by eight units high. The medium-resolution font, font 1, uses a 13-by-13 grid. All the other fonts use the finest grid, which is 21-by-21.

The grid width includes the space between characters. The grid height is for the height of an upper-case letter.

Font 0 is a non-proportional font. That is, most characters are of the same width, though certain special characters are wider. All the other fonts use proportional spacing —the widths of the characters vary, just as in the text of this manual.

All fonts contain all 94 printing ASCII characters. Fonts 0 and 2 contain C-PLOT's full special-character set, which is illustrated at the end of this chapter. Each other font except Gothic contains versions of the Greek letters and some of the special characters.

Note that high-resolution, multiline fonts may not be appropriate with low-resolution graphics devices or small character sizes.

See Appendix B for the character set available with each font.

sy select plotting symbol

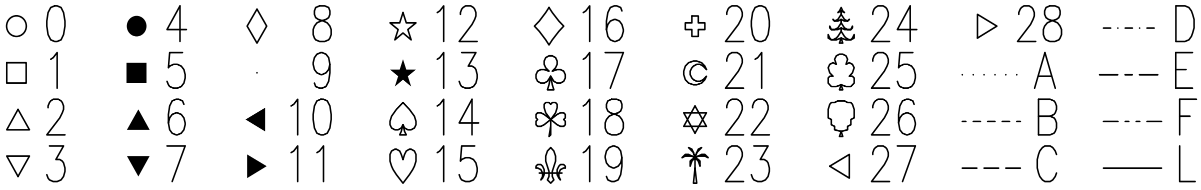
controls the type of symbol or line used to draw a plot. Any character from the current font can be used as a symbol in addition to the special symbols and line types built into C-PLOT.

```
sy
or
sy symbol
```

If *select symbol* is entered with no argument, the current symbol code is printed with a list of the codes for selecting symbols and line types. You can enter a new symbol code or a character. If a symbol code or character is given as an argument, no message is printed.

Any character can be used as a plotting symbol. The table below shows the codes for selecting special symbols and line types. Code `9`, a dot, is the default.

Plotting symbols



To use a numeral or one of the upper-case letters used as a code in the table above as the plotting symbol, precede it with a backslash, `\`. If the characters `c` or `z` are used as symbols, they must also be preceded with a `\` because of their special meaning to the key (see `gk`, below). To use any other character as a plotting symbol, enter it as an argument to `sy` or when prompted.

Symbols, except for lines, may be inserted within text by using the four-character sequence `\[`*##*, *##* is `00` for symbol 0 (a circle), `10` for symbol 10, etc.

Lines may be inserted into text using the three-character sequence `\*`*X*, where substituting `A` for *X* selects line-pattern A (dotted), `B` selects line pattern B (short dashed), etc.

The pattern length of dashed and dotted lines can be adjusted using the *set parameter* command described below.

gk enter symbols and text for plot key

lets you enter text for an explanatory key (or legend) to your plotting symbols. The key is drawn within the plot window using the *draw key* (pk and zk) commands described in Chapter 6.

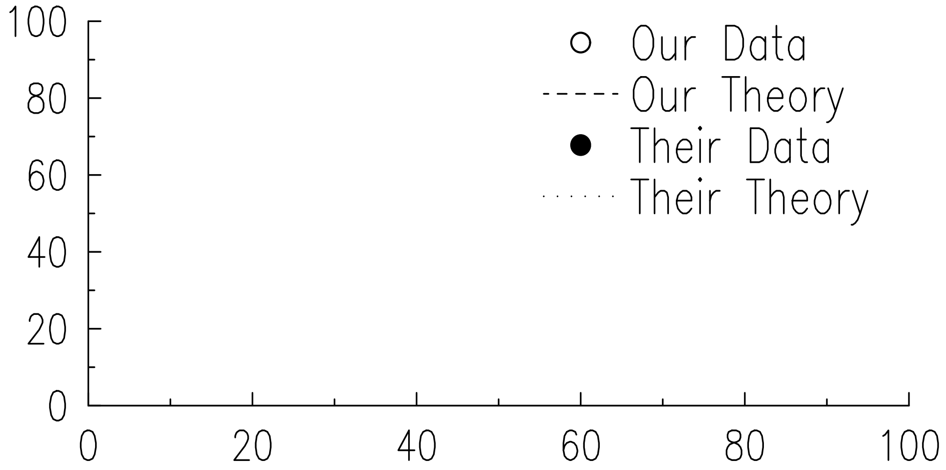
```
gk
```

Enter key will prompt you to enter lines of text for the key. Normally you will enter a symbol or character according to the coding used with the sy command. Insert one space after the symbol code and then enter the desired explanatory text.

For example,

```
PLOT-> gk
Enter symbols and text for up to 16 lines of the key.
Hit <return> for no change.  Hit <^D> to end.
KEY? 0 Our Data
KEY? B Our Theory
KEY? 4 Their Data
KEY? A Their Theory
KEY? ^D EOF

PLOT-> zeak 60u 90
```

will produce:



If you type the letter c before the symbol code, you will be prompted to change pens when the key is drawn on the plotter. If the first character on a line is z, no symbol will be drawn and the explanatory text will be drawn starting at the center of the symbol column.

The maximum text length for each line of the key is 64 characters, and you can have at most 16 symbols in the key.

tx enter text for plot labels and title

lets you add text for the title and axis labels.

```
tx
or
tx t|x|xu|y|yu|z|zu [text ... ]
```

The title will be drawn above the plot window and the labels along the appropriate axis. Units will be indicated following the labels in square brackets (or parentheses or APS style —see ty, described in Chapter 5).

With no arguments, you will be prompted to enter a title, then labels and units for each axis. The axis labels are entered separately from the units, allowing C-PLOT to insert a scaling factor, if necessary, between the label text and the text describing the units.

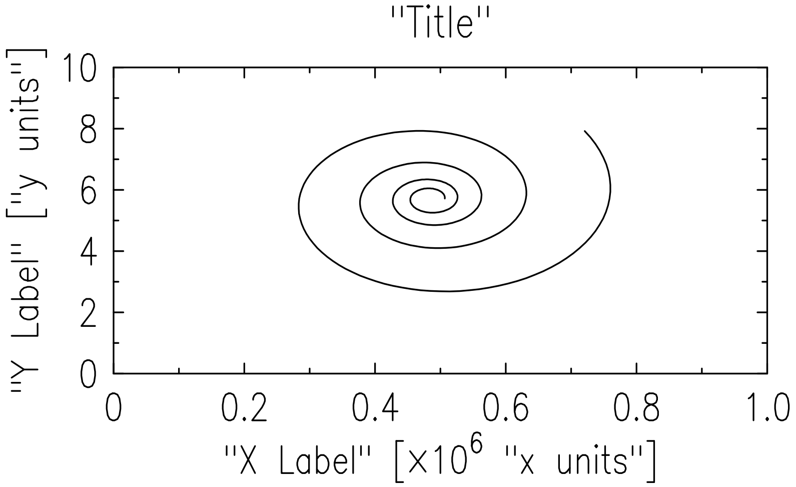
If you enter a single argument such as t, you will be prompted to enter just the title. If you enter x, y or z as an argument, you will be prompted for the axis label and units. Adding a u to the axis letter means you will be prompted just for the units. You also can enter the text on the command line following the argument. For example,

```
PLOT-> tx t this is the title
```

Use the cs command to control the dimensions of the characters. Use the special sequences shown at the end of this chapter to insert special characters in the text.

In the following example, text entered by the user is enclosed by quotes. C-PLOT inserts the square brackets and the scaling factor on the *x*-axis.

Plot title and labels



If you don't enter a unit string and if there is no scaling factor for the axis, there won't be any brackets. Certain plot-formatting options selected with the `ty` command affect how the scale factor is drawn: `ty 64` selects parentheses around the scale factor. `ty 1024` selects APS style. (See Chapter 5 for details.)

When you enter `tx`, the current text will be shown in parentheses. Entering nothing and pressing `<return>` will leave the current text unchanged. Typing `\<return>` will clear the current text.

The maximum number of characters in a title or label line is 256. Each character of a special sequence is counted, but spaces and tabs at the beginning of an entry will be disregarded unless the line begins with a `\`.

The text strings cannot be edited. They must be re-entered entirely. If you are using complicated subscripts, superscripts or other special sequences, you might find it more convenient to read the text in from a command file (see Chapter 10) created and edited with your favorite text editor.

se set parameters

allows you to control certain plot parameters.

```
se
se dash [dash_length]
se spacing [vert_spaces]
se tl [tick_length_percent]
se xtl [x_tick_length_percent]
se ytl [y_tick_length_percent]
se ztl [z_tick_length_percent]
se do_dir [cmd_file_directory]
se fn_dir [function_directory]
se gd_dir [data_directory]
se sigfig [number_of_digits]
```

The `se` command lets you change the values of several C-PLOT parameters. If you type `se` by itself, you will be prompted to enter values for each of the settable parameters. You can then type a new value or enter `<return>` to use the current value.

If you type `se` and the name of a settable parameter, you will be prompted for a new value for just that parameter. You also can directly type the parameter name and value on the command line.

The settable parameters currently are:

`dash` —controls the pattern length of the dashed and dotted lines drawn when the plot symbol is `A`, `B`, `C`, `D`, `E` or `F`. The default pattern length is 1 C-PLOT millimeter. The dash-length value is restored to the default value when you enter the *reset* command, `re`.

`spacing` —controls the spacing used between lines of text drawn as annotation or as the plot key. The units of this parameter are the height of the text. The default value —1.5 times the text height —is restored when you enter the *reset* command, `re`.

`tl` —lets you control the length of the tick marks. The value specifies the length of the long tick marks as a percentage of the length of the longest axis. By entering a value for `tl`, you turn on the mode where the tick marks for all the axes are the same length. This mode is the default and is set by the *reset* command, `re`, as is the default value of 1.5 percent.

`xtl`, `ytl`, `ztl` —let you control the lengths of the tick marks for each axis independently of the other axes. The value is the percentage of the length of the axis along the direction the tick marks point. By entering a value for any of these parameters, you turn on the mode where the tick marks for each axis are scaled independently. The *reset* command, `re`, turns off this mode.

`do_dir` —contains the name of the directory used to look for command files with the `do` command when a file isn't in the current directory or you haven't specified a path name that contains a slash character. The default value is taken from the value of the environment variable `CPLOT_DO_DIR`, if that exists. Otherwise, the default value is `$CPLOTHOME`/*cmdfiles*. The parameter may contain a colon-separated list of directories, in which case each directory is checked in turn for the file. The value of `do_dir` is not changed by the *reset* command, `re`.

`fn_dir` —contains the name of the directory to use for private user functions with the `fn` command. The default value is taken from the value of the environment variable `CPLOT_FN_DIR`, if that exists. Otherwise, the default value is `$HOME`//*functions*. The value of `fn_dir` is not changed by the *reset* command, `re`.

`gd_dir` —contains the name of the directory used to look for data files with the `gd` command when a file isn't in the current directory and you haven't specified a path name that contains a slash character. The default value is taken from the value of the environment variable `CPLOT_GD_DIR`, if that exists. Otherwise, the default value is `.`, the current directory. The parameter may contain a colon-separated list of directories, in which case each directory is checked in turn for the file. The value of `gd_dir` is not changed by the *reset* command, `re`.

`sigfig` —controls the maximum number of significant figures contained in the axis numbers printed on the plot. The default value is six significant figures. The reset command, `re`, restores the default value.

yg set gap between y-axis and label

lets you vary the distance between the y-axis and its label. This command is useful for lining up y-axis labels when several plots are grouped vertically on the same page.

```
yg
yg 0
or
yg value
```

The y-gap is the space between the y-axis and the axis label. Normally it is set so the label is just to the left of the tick numbers. By turning on the y-gap mode using this command, you can specify the number of character spaces to leave between the y-axis and the label.

The `yg` command toggles the y-gap mode. If it is toggled on, the user is prompted to set the number of character spaces between the label and the axis. Entering `yg 0` turns the y-gap mode off. Entering `yg` *`value`* turns the y-gap mode on and sets the gap to the specified number of character widths. The number need not be an integer.

Using special characters

The `cs` command sets text size and orientation, but C-PLOT offers more advanced options for tailoring the text design to your needs.

Formatting sequences

You can change many characteristics of the text by using special sequences embedded in the text. The backslash \ introduces all special sequences. The simple formatting sequences are:

Sequence	Meaning
`\u`	Move up half a line
`\d`	Move down half a line
`\l`	Make text 25% larger
`\s`	Make text 25% smaller
`\r`	Move up a whole line
`\b`	Move back one space
`\\|`	Move forward 1/6 a space
`\^`	Move forward 1/12 a space
`\B`	Move back 1/2 the width of the previous character

The following examples illustrate the use of these special sequences (also using some sequences described next):

$\mathrm{cm}^{2}\mathrm{sec}^{-1}$ → `cm\s\u2\d\l\¦ sec\s\u-1\d\l`

$\bar{M}_{W}$ → `M\b\u-\d\d\sW\l\u`

$\Big[$ → `\(lc\b\d\d\(lf\u` $\Bigg\{$ → `\(lb\b\r\(lk\b\r\(lt`

$\int_{\alpha}^{\infty}$ → `\T'5'\(rb\(*a\b\b\r\(bv\b\r\(lt\¦ \(if\T'0'`

Notice that the size changes to make the "2" superscript smaller and larger occur before and after the up and down movement. This is because the extent of vertical motion depends on the character size —the command `cm\s\u2\l\d` would not return to the same baseline, since the up motion is done at a smaller character size than the down motion.

Delimited special character sequences

Some of the following special sequences take decimal parameters, represented by the variable *N*. The first character before *N*. becomes the delimiter. Scanning for *N*. continues until either a matching delimiter or non-digit, non-sign or non-decimal-point character is found. The delimiter can be any character.

Sequence	Meaning
`\h'`*N*`'`	Move horizontally (12 units per character width; negative is left)
`\v'`*N*`'`	Move vertically (12 units per line; negative is up)
`\S'`*N*`'`	Change character size (in percent; neg. is smaller)
`\T'`*N*`'`	Set character angle (in degrees; negative tilts left)
`\R'`*N*`'`	Rotate text baseline (in degrees; negative is counterclockwise)
`\P'`*N*`'`	Select pen number *N*
`\H'`*N*`'`	Move *N* spaces horizontally from beginning of line
`\V'`*N*`'`	Move *N* lines vertically from beginning of line
`\W'`*text*`'`	Move horizontally the width of *text*

Although the text rotation sequence, `\R'`*N*', may be included in titles and labels, C-PLOT makes no adjustments to center the rotated text or keep it from colliding with other parts of the plot.

The last three commands are especially useful with the proportionally spaced fonts, when you want to line up text in columns or do overstrikes such as underlining. With proportionally spaced fonts, you don't necessarily know how wide a particular string of text will be. As an example, consider this annotation text produced by the fit user functions described in Chapter 11:

```
\H@13@\W@-Linear @Linear = 4.02\(+-0.83
```

The `\H@13@` sequence positions the equals sign 13 character units from the annotation's start. Then the `\W@-Linear@` sequence backs up the width of the text before printing it. The minus sign is part of the `\W` syntax and indicates the motion is to be to the left. The `@` symbol is used as the sequence delimiter.

Special characters

Special characters —scientific, mathematical or foreign —are denoted with a four-character sequence. Each sequence begins with `\(` and ends with a two-character code for the special character. Here is a list of the special characters and the last two characters of the four-character sequences. (Please note that not all characters are available in all fonts. Appendix B shows which characters are available in each font.)

ﬀ ff	ﬁ fi	ﬂ fl	ﬃ Fi	ﬄ Fl	_ ru	— em	¼ o4	½ o2	¾ 34
¢ ct	- hy	© co	° de	† dg	′ fm	® rg	• bu	□ sq	* **
+ pl	− mi	× mu	÷ di	= eq	≡ ==	≧ >=	≦ <=	≠ !=	± +−
¬ no	/ sl	~ ap	≅ ~=	∝ pt	∇ gr	→ −>	← <−	↑ ua	↓ da
∫ is	∂ pd	∞ if	√‾ sr	⊂ sb	⊃ sp	∪ cu	∩ ca	⊆ ib	⊇ ip
∈ mo	ø es	´ aa	` ga	○ ci	🔔 bs	§ sc	‡ dd	☞ rh	☜ lh
⎛ lt	⎞ rt	⎡ lc	⎤ rc	⎝ lb	⎠ rb	⎣ lf	⎦ rf	⎨ lk	⎬ rk
| bv	ς ts	| br	| or	_ ul	‾ rn	Å an	≲ ~<	≳ ~>	⊥ pe
‖ pa	Æ AE	æ ae	Ø O/	ø o/	œ oe	Ä Ad	ä ad	Ö Od	ö od
Å AN	å An	^ as	ç fc	Œ OE	¯ mc	˘ be	˙ dt	¨ um	¨ UM
˚ ri	¸ cd	˛ og	ˇ hc						

Greek characters

Four-character sequences beginning with `\(*` denote Greek letters. The fourth letter of the sequence is the Roman letter that corresponds to the Greek letter. The following table shows the correspondence.

a	b	g	d	e	z	y	h	i	k	l	m	n	c	o	p	r	s	t	u	f	x	q	w
α	β	γ	δ	ϵ	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο	π	ρ	σ	τ	υ	ϕ	χ	ψ	ω
A	B	G	D	E	Z	Y	H	I	K	L	M	N	C	O	P	R	S	T	U	F	X	Q	W
		Γ	Δ				Θ			Λ			Ξ		Π		Σ		ϒ	Φ		Ψ	Ω

Accent marks

The sequences shown below automatically back-space and add the accent marks shown to lower-case letters:

a\\.	ȧ	c\\,	ç	a\\:	ä	a\\´	á	a\\`	à	a\\–	ā
a\\o	å	a\\~	ã	a*u	ă	a*v	ǎ	a*^	â		

Miscellaneous sequences

Finally, there are these remaining special character sequences:

Sequence	Meaning
`\f#`	Change to font #
`\fP`	Change to previous font
`\C`	Center annotation text within plot window
`\*g`	Insert name of current data file
`\*X`	Insert line segment corresponding to symbol, where *X* is *A*, *B*, *C*, *D*, *E*, *F* or *L*
`\[##`	Insert symbol, where ## is `00` for symbol 0, etc.
`\\`	A single backlash, \
`\X`	*X*, any character *not* in this or previous tables

The `\C` sequence must appear at the start of a line of annotation.

Chapter 7 Drawing the Plot

This chapter includes explanations of the commands for drawing plots:

- □ on the pen plotter
- □ on graphics-filter devices

Chapter 8 describes how to initialize the pen plotter, while Chapter 9 shows how to initialize filters for the other graphics devices. The pen-plotter versions of drawing commands described below begin with a `p`; the graphics filter versions begin with a `z`. In both cases, many of the commands can be grouped together by following the initial `p` or `z` with the appropriate letters. For example, `palt` will draw axes, labels and the title on a pen plotter.

Commands covered

p, **pz**, **z** or **zz**	draw complete plot
pa or **za**	draw axes
pp or **zp**	draw points
pb or **zb**	draw error bars
pl or **zl**	draw axis labels
pt or **zt**	draw title
pn or **zn**	annotate
pk or **zk**	draw key
pd or **zd**	place date in corner

p, pz, z, zz draw complete plot

instructs C-PLOT to draw a complete plot using the ranges, text and data you have entered.

```
p or pz
or
z or zz
```

The `p` and `pz` commands will draw the full plot on the pen plotter. The `z` and `zz` command will draw the entire plot on the screen, printer or other graphics device selected with *initialize graphics filter*, `zi`, described in Chapter 9. The `z` and `zz` commands are actually shorthand that groups the individual commands `zealtpb`. Likewise, `p` and `pz` are shorthand for `paltpb`.

pa, za draw axes

draws the axes and axis numbers on the pen plotter or graphics filter.

```
pa or za
```

No parameters are used. The axes and axis numbers are drawn using the current range and type settings. (See `ra`, `ro` and `ty`, Chapter 5.) Note that if the axes have a scale factor, you must draw labels (`pl zl`) for the scale factor to be drawn.

pp, zp draw points

draws just the points for a given set of data.

```
    pp or zp
```

No parameters are used. The points will be drawn without labels, title or axes.

If data is being obtained using `gd 11`, `gd 12` or `gd 14` to read files of indefinite length, the entire data file will be read automatically in sections as large as the program can hold at one time and the data plotted. If real-time plotting is selected with `gd @`, C-PLOT will keep reading points from the file until terminated as described in Chapter 3.

pb, zb draw error bars

puts error bars on the plot.

```
    pb or zb
```

No parameters are used. If you intend to use error bars with a particular set of data, be sure the error-bar mode is on when you read in the data from a file.

The vertical width of the top and bottom horizontal segments of the *y* and *z* error bars is taken from the symbol width set using the `cs` command. The height of the vertical segments of the *x* error bars in 2D mode is taken from the symbol height set using the `cs` command. If the symbol height or width is larger than the error bar, no error bar will be drawn, unless the current symbol type is symbol 9, a dot, in which case the symbol size is ignored.

If data is being obtained using `d 11`, `gd 12` or `gd 14` to read files of indefinite length, the entire data file will be read automatically in sections as large as the program can hold at one time and the error bars plotted.

pl, zl draw axis labels

draws the current labels, entered using the tx command, on the pen plotter or graphics filter.

```
pl or zl
```

Both the labels and units are drawn. The units will be enclosed by square brackets or parentheses, depending on the plot-type option selected with the ty command (see Chapter 5). If a scaling factor is needed for the axis numbers, it is drawn within the brackets or parentheses, preceding the units. The units will be enclosed by parentheses and the scaling factor will be positioned according to APS conventions, if selected with the ty command.

The labels, along with the corresponding units and scale factor, will be drawn centered along the appropriate axis.

The labels also are drawn when p, pz, z or zz are entered.

pt, zt draw title draws the title entered with the tx command on the pen plotter or graphics filter.

```
pt or zt
```

No parameters are used. The title is drawn centered above the current plot window. The title also is drawn with the p, pz, z and zz commands.

pn, zn annotate

lets you copy text line by line, either directly from the keyboard or from a file, to any location on the plot page. With the *annotate* command you can put clarifying information on your plots or create text-only documents using C-PLOT's various fonts and font-control features.

```
pn
pn horz_offset vert_offset [filename]
pn xbox ybox zbox [filename]
or
zn [same options]
```

After entering pn or zn, you can enter annotation text one line at a time, with each line drawn or displayed when you enter <return>. At the end of each line of text the drawing position advances down the page a fixed amount of space, adjustable using the *set parameter* (se) command (see Chapter 6). The default spacing is 1.5 times the character height. When you are finished entering text, enter ^D.

By entering values following pn or zn, you can position the annotation anywhere on the page. In 2D mode, if offset parameters are present, the annotation will begin *horz_offset* centimeters to the right and *vert_offset* centimeters down from the upper-left corner of the plot window. If one or both of the values entered for the offsets ends with the letter `u`, the offset values will be interpreted in data units corresponding to the axis numbering rather than in centimeters. Annotation text can be positioned outside the plot window.

In 3D mode, offsets not in data units are in box units (see `bo`, Chapter 5). Box units run from 0 to 1 for positions inside the 3D cube. At present, 3D annotation can only be placed in the *x-z* plane.

If offsets are absent, the annotation will be drawn or displayed starting at the default or the previous annotation starting point. Pen-plotter users may reposition the pen using the plotter controls. When the annotation is drawn based on the last-used starting position, that position is remembered in terms of its actual location in the plot window, not data units. The last-used position is forgotten when the window is changed (`wi`) or the plot is turned (`tu`).

The default starting position for annotation text in 2D mode is $h = 0.125 \times$ *x_length* and $v = 0.125 \times$ *y_length*, where *x_length* and *y_length* are the dimensions of the plot window. The default starting position for annotation text in 3D mode in box units is $x = 0.125$, $y = 0$ and $z = 0.125$.

Each line of input text starts a new line on the plot unless the input line ends with a `\`. Since the maximum length of an input line is 255 characters and an input line may include formatting characters that will not be drawn, you may need more than one input line to create a single line of text on the plot. Ending a line with a \ lets you string input lines together.

Beginning a line with `\C` will center the line horizontally in the window.

An initial \ is required to enter initial blank spaces from the keyboard because C-PLOT normally strips away blank spaces at the start of a line. Changing character size within a line does not change the line spacing. If you use special character sequences, the size and slant of the characters are reset to their original values after each input line.

pk, zk draw key

draws the key, entered using `gk`, in the upper-left corner of the plot window or in another position you specify.

```
pk
pk horz_offset vert_offset
pk xbox ybox zbox
or
zk [same options]
```

With no parameters, the key will be drawn starting at the default or the previously used position. You can use the plotter controls to move the pen to another position before drawing the key, or you can position the key by entering values after the pk or zk command.

In 2D mode, if offsets are given as arguments, the key will be drawn *horz_offset* centimeters to the right and *vert_offset* centimeters down from the upper-left corner of the window. If one or both of the values entered for the offsets end with the letter u, the numbers are interpreted in data units corresponding to the axis numbering rather than in centimeters.

In 3D mode, offsets not in data units are in box units (see bo, Chapter 5). Box units run from 0 to 1 for positions inside the 3D cube. At present, the 3D key can only be placed in the *x-z* plane.

The default starting position for the key in 2D mode is $h = 0.125 \times x\_length$ and $v = 0.125 \times y\_length$, where *x_length* and *y_length* are the dimensions of the plot window. The default starting position for the key in 3D mode in box units is $x = 0.125$, $y = 0$ and $z = 0.125$.

When the key is drawn based on the last-used position, that location is remembered in terms of its actual place in the plot window, not in data units.

The position used last is forgotten when the window is changed (wi) or the plot is turned (tu).

The spacing between lines can be adjusted using the *set parameter* command (see Chapter 6). The default spacing is 1.5 times the character height.

pd, zd place date in corner

puts the current time and date in a corner of the page.

```
pd
or
zd
```

No parameters are used. The current date and time in the format Tue May 26 22:19 1992 is drawn in the corner of the page. The location is fixed, though you can vary the size and style of the lettering using the *character size* (cs) command (see Chapter 6). The location is chosen so that if the plot is put into a standard notebook, the date will appear right-side up in the upper-right corner of the page, whether the plot is in landscape or portrait mode (see tu, Chapter 5).

Where to put arguments when stringing commands together

You can combine several drawing commands on one command line, but don't put spaces between the command characters. Any character following a space on the command line will be treated as an argument. *Annotate* and *draw*key described in this chapter, and *select pen velocity*, described in the next, can all take arguments. You must put the arguments after a single string of characters grouping all the commands together. For example,

```
    pvaltnk vel an_hor an_ver an_file key_hor key_ver
```

will supply the pen velocity, the position and file for the annotation and the position for the key. First come the commands, then come the arguments.

You have to insert place holders for arguments you don't want to supply. The place-holder character is a `-`. A single `-` will hold the place of all the position arguments to a command (two in 2D mode and three in 3D mode) for *annotate* and *draw key*. For instance,

```
    pvaltnk vel - an_file key_hor key_ver
```

will instruct the annotation command to act as it does when it is entered without position arguments.

You only need place holders if you have to put arguments past the place holders. For example, if you just want to set the pen velocity and supply the position for the annotation, you can enter

```
    pvaltnk vel an_horz an_vert
```

Chapter 8 More Plotter Commands

This chapter discusses special commands for:

- □ initializing the pen plotter
- □ selecting pens
- □ selecting the pen drawing speed

Commands covered

in	open and initialize pen plotter
rp	release pen plotter
p#	select pen
pv	select pen velocity
pw	don't move pen off page
px or **ps**	move pen off page

in open and initialize pen plotter

opens an input-output channel to the pen plotter. Once you have opened the plotter using this command, you will be able to use the drawing commands to make plots. Invoking this command when the plotter is already opened reinitializes the plotter.

```
in [device] [baud]
or
in m
```

Without an argument, `in` attempts to establish an input-output channel with the name assigned to the variable DEVICE in the *cplot_config* file. If the device is on a serial interface, the baud rate is also taken from the variable BAUD in that file. (See Appendix A for an explanation of the *cplot_config* site-initialization file.) You also can select the device name or baud rate or both by entering them as arguments after `in`. Once the device or baud rate is selected as an option to `in`, C-PLOT disregards the values from the *cplot_config* file.

If you enter `in m`, the program assumes the pen plotter is sharing a serial line to the computer with your terminal. This only works with pen plotters that implement eavesdropping mode and have been connected to the terminal and computer with a special Y-cable. (The switch on the back of plotter with the markings D and Y should be in the Y position for eavesdropping mode.)

When not using eavesdropping mode, the program treats serial and GPIB (General Purpose Interface Bus) interfaces differently. The interface used is established in the *cplot_config* file according to the value of the variable `GPIB`. With a serial interface, the program uses *ioctl()* system calls to set the baud rate, turn off echoing, turn on XON-XOFF flow control and so on for the line to the plotter. (The tty modes of the serial line are reset to their initial values when C-PLOT exits or the plotter is released.) The baud rate is taken from the value of `BAUD` in the *cplot_config* file. When the device is opened, the program sends out special HP-GL commands to configure the serial interface on the plotter. The program does not do anything to configure a GPIB interface.

The following paragraphs describe the sequence of events when C-PLOT opens and initializes the pen plotter. From this list, you may be able to determine what is going wrong when the program cannot open the plotter.

1) C-PLOT tries to open the special file whose name (for example, */dev/plotter*) is given in the *cplot_config* file or as an argument to the `in` command. The open can fail if the name doesn't exist, the name isn't a character-type special file, you don't have read and write permission for that special file or the device has been locked by another user (if your system uses the exclusive use method of locking).

2) C-PLOT checks to see if the plotter has been locked using the *lockf()* system call (if your system uses this method). If the plotter is locked, you will not be able to gain control until it has been unlocked by the other user. Otherwise, the program locks the plotter to keep other users off.

3) If the serial interface is used (but not in eavesdropping mode), the baud rate and line modes are set.

4) For serial devices, a reset message and the handshake parameters are sent.

5) A message is sent asking the plotter to output its error status. If the plotter doesn't respond within three seconds, C-PLOT assumes there is no plotter and tells you so.

6) C-PLOT asks the plotter for more information about itself, including the size of the maximum plotting area. Subsequent plots drawn using a graphics filter will be scaled so that the plotter's maximum plotting area will be mapped onto the filter's maximum plotting area.

rp release pen plotter

ends C-PLOT's connection with the plotter, allowing another user to gain control.

```
rp
```

In a multiuser system with locking or exclusive use in effect, only one user at a time can be using the plotter.

p# select pen

changes the pen used for drawing on the pen plotter. Since many pen plotters have more than one pen, you may have pens of different thicknesses or different colors installed.

```
p#
```

No arguments are used. For the # sign, enter an integer greater than or equal to zero and less than 1,000 to identify the pen position. (Higher pen numbers are interpreted by graphics filters to control other features, such as line width.) Refer to your pen plotter manual to see how pens are numbered. When selecting one or more pens as part of a string of plotting commands, insert the pen numbers in the string rather than including them as arguments. For instance, `p2alt1p` uses pen 2 to draw the axes, labels and title, and pen 1 to draw the points.

The `p#` command is comparable to the *select filter line style* command, `z#`, described in Chapter 9.

pv select pen velocity

controls the drawing speed on the pen plotter. The quality of the lines on the plotter depends on the drawing speed and acceleration of the pen. You may sometimes want to draw plots quickly, at the expense of well-inked lines, or you may want to slow the pen for better quality lines. This command has no effect on graphics filters.

```
pv velocity
or
pv −1
```

With no parameters, the current pen velocity is printed and you are prompted for a new value.

The numeric velocity parameter you select is sent literally to the pen plotter. Positive values give the drawing speed in centimeters per second. A negative value generally selects the maximum velocity of the pen plotter, but at higher pen acceleration. The default value for pen velocity is 10 centimeters per second.

pw don't move pen off page

leaves the pen-plotter pen positioned above the last commanded drawing location.

```
pw
```

Normally, after pen-plotter drawing commands are executed, C-PLOT commands the plotter to replace the pen in its home (often capped) position. If you are drawing a single plot with multiple commands, you can save time by having the pen remain over the plot.

The pw command is comparable to the *leave filter open* command, zw, described in Chapter 8.

px, ps move pen off page

returns the pen-plotter pen to its home position and, on plotters so equipped, ejects the current page and feeds a new one.

```
px or ps
```

Both px and ps have the same effect —each returns the pen to its home position if the plotter had been previously instructed to leave the pen over the page with the pw command. On plotters with automatic paper feed, px or ps may be used to force a new page. The *move pen off page* commands can be compared with the *close filter* commands, zx and zs, described in Chapter 9.

Chapter 9 Graphics Filter Commands

In this chapter you will find instructions for:

- □ initializing, opening and closing graphics filters
- □ erasing old plot material from graphics filters
- □ controlling the echoing of commands

Commands covered

zi	initialize graphics filter
zf	select filter
sc	select filter scaling factors
z#	select filter line style
ze	erase old plot
zE	erase current window
zq	don't write text to screen
zw	don't close filter yet
zx	close filter
zs	close filter, synchronous

Type `<return>` twice with `z` commands

All the `z` commands except `ze` require you to enter a second `<return>` at the keyboard before the `PLOT->` prompt will again be displayed. This feature accommodates filters displaying graphics on terminals that have a distinct graphics mode and that erase the graphics when put back into text mode. You can view the graphics display, then enter the second `<return>` to get back to text mode.

The current graphics filter isn't necessarily associated with the screen. If you are using a printer filter and enter a `z` command and `<return>`, nothing will appear on the screen and the plot won't be sent to the printer until you enter the second `<return>`.

`zi` initialize graphics filter

starts a process to interpret the `z` commands for graphics devices.

```
zi
or
zi filter [filename | @spool_opts] [filter_opts]
```

When no parameters are entered with `zi`, a list of the files in `$CPLOTHOME`/*filters* is displayed and you can enter the name of the filter program you want to use. The name of the last-used filter, or if there is none, the default filter, also will be shown. You may enter `<return>` to use that filter or enter the name of the desired filter.

Giving the file or device name *`filename`* as the first argument after *`filter`* switches the standard output of the filter to *`filename`*. (Note that *`filename`* can't begin with `-` or `@` character.) You might use this option to redirect video displays to another terminal or to put output destined for a printer into a file to be sent to the printer at a later time. For example,

```
    zi psfilter plot.ps
```

sends the output of *psfilter* to the file *plot.ps*.

Although filters that have output destined to be printed (or spooled) are usually already configured to send that output to a particular printer, you may override the default spooling commands by using the `@`*`spool_opts`* feature. For example,

```
    zi psfilter @lp @-dlaser @-s
```

sends the output of *psfilter* to the lp spooling command `lp` with the options `-dlaser -s`. Otherwise, the spooling command is taken from the file `$CPLOTHOME`/*filters*/*spoolers*. If there is none specified there, the compiled-in spooling command, if present, is used. If no spooling command is specified with any of those three methods, the filter output is written to the screen.

If you enter any of the `z` commands before initializing a filter with `zi`, you will be prompted for a filter name. Again, entering `<return>` will invoke the filter displayed, or you may enter a different filter name and then `<return>`.

Before a new filter process is begun, the program waits for the previous filter process (if any) to terminate. If you enter `zi` and C-PLOT is waiting for the previous filter process to terminate, typing a `^C` will return the program to the `PLOT->` prompt. Entering another `zi` will kill the previous filter process without waiting for it to terminate normally. You may want to terminate a filter to kill a misbehaving filter program or to abort a plot when you don't want to wait for it to be completed.

The filter also is killed when C-PLOT exits; the program is designed to clean up all subprocesses and temporary files it has created. To make sure a filter associated with a printer has finished processing its input, type `zi` before exiting the program. The program will wait until the current filter has finished and then prompt for a new filter name. You can hit `<return>` and then exit the plot program. Alternatively, use the `zs` command (described below) to synchronize filter termination with the plot program.

Some filters, such as those that generate PostScript or HPGL-2, recognize command line options. For example,

```
zi psfilter plot.eps -eps -color
```

places encapsulated color PostScript in the file *plot.eps*.

The filters reside in `$CPLOTHOME/`*filters*. If *`filter`* includes a `/`, the name is taken as an absolute or relative path name. The names of the filters are arbitrary, and each site is free to rename the installed filters.

`zf` select filter

lets you maintain up to eight active graphics filters.

```
zf
or
zf#
```

Entering `zf` alone selects the next filter in the sequence from 1 to 8. If the current filter is 8, filter 1 is selected. `zf1` selects filter 1, `zf2` selects filter 2, etc. If no filter is active when a `zf` command is entered, C-PLOT will prompt for a filter to be specified. Filter 1 is the default.

Once a filter is selected, it remains the target of z commands until another is chosen.

`sc` select filter scaling factors

informs C-PLOT of the exact size of the graphics filter device in use.

```
sc
or
sc short_side long_side
```

With no parameters, you are prompted to enter the lengths in centimeters of the short and long sides of the current graphics filter device's maximum plotting area. You can also give the two values on the command line. When the correct values are entered, the centimeter units used by the commands `cs`, `wi`, `pn`, `zn`, `pk` and `zk` will be accurate on the filter plot.

Entering values of 0 will restore the default scaling, which maps the entire pen-plotter plotting area to the maximum plotting area available on the graphics-filter devices.

If the graphics-filter drawing area is set smaller than the pen-plotter area, it is possible to generate plots that run off the graphics-filter page, as the window commands operate with respect to the pen-plotter plotting area.

z# select filter line style

changes the line width or color of lines drawn using certain graphics filters. The effect of the command depends on the particular graphics-filter program, although most adhere to the conventions described at the end of this chapter.

```
z#
```

No arguments are used. For #, enter an integer greater than or equal to zero. When selecting one or more line styles as part of a string of plotting commands, insert the numbers in the string rather than including them as arguments. For instance, `z2alt1p` uses style 2 to draw the axes, labels and title, and style 1 to draw the points.

The `z#` command is comparable to the *select pen* command, `p #`, described in Chapter 7.

ze erase old plot

clears from the graphics filter any plot, or parts of a plot, that have been drawn using the commands that begin with `z`. If the graphics filter is associated with a video screen, the screen will be erased.

```
ze
```

If you have already drawn a filter plot or any of its elements using `z`, `zz`, `zp`, `za` or related commands (see Chapter 7), you can use `ze` to clear the memory of the old plot elements and begin fresh with a new plot. Although `z` and `zz` will automatically clear away old plot material before drawing a fresh plot, the other filter commands simply overwrite the existing material.

zE erase current window

erases the area within the current window

```
zE
```

The position and size of the area to be erased can be set with the `wi` command. The color of the erased area is set to the current background color.

There is no comparable command for the pen plotter.

zq don't write text to screen

stops C-PLOT from echoing to the screen the characters you type at the keyboard, and it stops the program from printing any messages on the screen for as long the filter is open.

```
zq
```

The `zq`, or quiet-mode, command is useful with some filter programs that use the terminal as their display device. The command prevents C-PLOT messages from becoming interspersed with graphics commands sent by the filter process or, if applicable, it prevents the filter display from being overlaid with messages generated by C-PLOT.

In quiet mode, your terminal will not echo the characters you type at the keyboard. C-PLOT will not print messages on the screen; only the output of graphics filters will be displayed. The quiet-mode state is passed to user functions (see `fn`), and you can arrange for them not to write to the screen when quiet mode is on. However, output from a subshell (see `u`) will be written to the screen.

If the filter is being kept open with the `zw` command, once quiet mode is turned on, it will stay on until the filter is closed with the `zx` or `zs` commands, another filter is initialized with the `zi` command, you interrupt a command with `^C` or you exit the plot program.

Quiet mode is usually necessary with filters such as *4010* and *4014*. Quiet mode is not enforced when plotting on the pen plotter.

`zw` don't close filter yet

keeps the graphics device open while you build up a complex plot with multiple drawing commands.

```
    zw
```

Complex plots having multiple data sets, windows, annotations, etc. are put together by entering `zw` before the drawing commands. When all elements of the plot have been drawn, a *close filter* command, `zx` or `zs`, explained below, is used to complete the plot.

When the *do not close filter yet* option is used, the filter will stay open until a `zx` or `zs` command is entered, another filter is initialized with the `zi` command, you interrupt a command with a `^C` or you exit the plot program.

When `zw` is used in conjunction with printing devices, the plot will not be sent to the printer until a *close filter* command is issued. Display terminals that have distinct graphics and text modes will remain in graphics mode until a *close filter* command is entered.

The `zw` command is comparable to the *don't move pen off page* command, `pw`, described in Chapter 7.

zx close filter

tells the filter process that the current plot is finished.

```
    zx
```

If the filter is associated with a printer, zx will send the current plot to the printer. You generally use zx to close a filter after using zw to keep it open.

On video terminals that maintain separate displays for text and for graphics, zx is the recommended command for switching from text to graphics in order to view an existing plot. Typing a second <return> will return the screen to text mode.

zs close filter, synchronous

tells the filter process that the current plot is complete, like zx, but it also causes C-PLOT to pause until the filter is finished writing to your terminal.

```
    zs
```

The synchronization option is useful for filters that write graphics commands to your terminal over the same serial line as ordinary text. Graphics commands from the filter program may get garbled with text from the plot program. The synchronized close, zs, stops this intermingling of output. As an example, zqzkds draws the plot, key and date with quiet mode on and prevents the PLOT-> prompt from being issued until the last graphics command has been sent to the terminal. As with zx, a second <return> will return you to text mode.

Pen numbering

C-PLOT graphics filters are written to conform to the pen-numbering conventions described in this section. These conventions allow a command file to produce consistent plots on pen plotters, laser printers and mono or color video monitors.

The plot program itself doesn't distinguish among pen numbers (except to ignore numbers greater than 1000 when selecting the pen-plotter pen). The number entered with the `z #` command is simply passed on to a graphics filter. The code in each filter can use or ignore the pen number, depending on the device characteristics. Not all pen numbers will be functional on all filters.

Line-width values are interpreted in C-PLOT basic units, where there are 40 basic units in 1 C-PLOT millimeter. A line width of zero should select the thinnest line available on a device.

The following table shows the recommended correspondence between pen number and attribute:

0-999	drawing colors	4000-4999	line widths
1000-1999	*white*-fill fill colors	5000-5999	symbol outline widths
2000-2999	*black*-fill fill colors	9000-9999	flags
3000-3999	background colors	9100-9900	filter-dependent flags

The colors associated with pen numbers 1000 through 3999 modulus 1000 are to be the same as the colors associated with pen numbers 0 through 999.

Pens 0, 1000, 2000 and 3000 track the background color assigned with pens 3002-3999. By drawing with pen 0, you can erase previously drawn portions of the plots. Pens 1, 1001, 2001 and 3001 track the foreground color assigned with pens 10-999.

The following 9000-series flags have been defined:

9001	turns symbol filling on
9002	turns symbol filling off
9003	turns *black*-filled symbol outlines on
9004	turns *black*-filled symbol outlines off
9999	resets all attributes to startup default values

When symbol filling is on, the interior of the symbols is painted solidly with the fill color. When off, *white*-filled symbols show through what is underneath, and *black*-filled symbols are colored by drawing a grid of lines. The startup default is to have symbol filling on.

The *white*-filled symbols are those selected with the `sy` command using symbol codes 0, 1, 2, 3, 8, 12, 14-28. The *black*-filled symbols have symbol codes 4, 5, 6, 7, 10, 11 and 13. *White*-filled symbols are always outlined by the current line color. *Black*-filled symbols are outlined with the current background color or not at all if the no-outline flag has been selected. The startup default is to have *black*-filled symbol outlines drawn.

The following pen color assignments are standardized for color display devices:

0 —background	5 —yellow
1 —foreground	6 —cyan (blue-green)
2 —blue	7 —magenta (blue-red)
3 —red	8 —white
4 —green	9 —black

The colors for VGA and EGA graphics on 386 and 486 PC computers are as follows:

0 —black	9 —black
1 —bright white	10 —light blue
2 —blue	11 —light green
3 —red	12 —light cyan
4 —green	13 —light red
5 —yellow	14 —light magenta
6 —cyan	15 —brown
7 —magenta	16 —gray
8 —bright white	17 —white

The colors available on X Windows, SunView, color PostScript and HP-GL follow:

0 —white	38 —olive drab
1 —black	39 —medium sea green
2 —blue	40 —med. spring green
3 —red	41 —pale green
4 —green	42 —sea green
5 —yellow	43 —spring green
6 —cyan	44 —yellow green
7 —magenta	45 —dark slate gray
8 —white	46 —dim gray
9 —black	47 —light gray
10 —aquamarine	48 —gray
11 —med. aquamarine	49 —khaki
12 —blue	50 —magenta
13 —cadet blue	51 —maroon
14 —cornflower blue	52 —orange
15 —dark slate blue	53 —orchid
16 —light blue	54 —dark orchid
17 —light steel blue	55 —medium orchid
18 —medium blue	56 —pink
19 —med. slate blue	57 —plum
20 —midnight blue	58 —red
21 —navy blue	59 —Indian red
22 —sky blue	60 —medium violet red
23 —slate blue	61 —orange red
24 —steel blue	62 —violet red
25 —coral	63 —salmon
26 —cyan	64 —sienna
27 —firebrick	65 —tan
28 —brown	66 —thistle
29 —sandy brown	67 —turquoise
30 —gold	68 —dark turquoise
31 —goldenrod	69 —medium turquoise
32 —light goldenrod	70 —violet
33 —green	71 —blue violet
34 —dark green	72 —wheat
35 —dark olive green	73 —yellow
36 —forest green	74 —green yellow
37 —lime green	

Colors 75 to 175 are 101 shades of gray from black to white.

Chapter 10 Command Files

In this chapter you will find:

- □ instructions for making files that contain sequences of commands that will run C-PLOT
- □ an explanation of how to pass arguments to these command files how to use the command files to run the program in batch mode (in the background)
- □ how to make the program interpret graphics filter instructions as pen-plotter instructions and vice versa, using the same command file in both cases.

Commands covered

do	take commands from a file
mk	make a command file
em	end making a command file
sf	save current format
ch	change target of drawing commands
w	wait for user to enter `<return>`

do take commands from a file

instructs C-PLOT to take input from an ASCII file. This command file may contain frequently used or complicated sequences of commands. The text in the file is in the same format as if the commands had been typed from the keyboard.

```
do [cmd_file]
or
do cmd_file arguments ...
```

Command files contain characters and newlines in the same sequence that you would type at the keyboard. Command files can be nested —that is, a command file can invoke other command files — up to four deep. If the argument *cmd_file* is the single character . , the same file as last time is used.

Without parameters, do will print the name of the current command file and prompt for the name of a new command file. You will not be able to pass arguments if you specify the command file this way.

Instructing the program to execute a command file by typing do*cmd_file* changes the prompt from PLOT-> to plot1->. The number 1 indicates the level of nesting. The commands in the file *cmd_file* will then be executed in sequence. When the commands in the file are exhausted, program input reverts to the keyboard, except for the case of an ex command in the file, which will immediately terminate the plot program.

A command file can be invoked when entering the program by giving the name of the file as an argument in the shell command line. This feature lets you run the program as a background process. See "Running In Batch Mode" at the end of this chapter.

Command files can be created with *make command file*, mk; *save current format*, sf; or a text editor.

Which directory?

When a file name is entered for the command file, C-PLOT first looks for that file name in the current directory. If it can't find the file there, and if the file doesn't contain the / character, C-PLOT looks in the directory given by the shell variable CPLOT_DO_DIR in your environment. If that variable is not found in the program's environment, the directory $CPLOTHOME/*cmdfiles* is checked to see if it contains *cmd_file*. CPLOT_DO_DIR can contain a colon-separated list of directories, in which case C-PLOT will look for the file in each directory in turn.

Argument substitution

The argument-substitution feature lets you pass arguments to the command file, making it possible to vary parameters each time you use the file. As C-PLOT reads the file, strings in the command file of the form $1, $2 and so on will be replaced by arguments entered with the do command. For instance, if the command file named *cmd_file* contains the lines

```
gd 2 data.$1
wi $2 $3 5 5
```

and you type on the command line do cmds 32 2 3, the plot program will get data from the file *data.32* and place a 5cm × 5cm window 2cm from the left and 3cm from the bottom of the page.

Up to nine arguments may be passed to the command file on the command line. The variable $0 is replaced by the name of the command file. On the command line, arguments are separated by spaces, except that double quotes may be used to group several words as a single argument. Literal double quotes and dollar signs may be passed as \" and \$.

Points to remember

The control characters `^D` or `^N` should be represented in the command file by the combination of the printing characters `^` and `D` or `N` instead of the literal control characters. However, the two-character sequences are only interpreted as control characters if they appear as the first two characters on a line.

When using command files, remember that commands that toggle states or plotter functions (`eb`, `vt`, etc.) should be used to select a known state by giving an argument of `0` or `1`. Otherwise, the command will simply toggle the function to the opposite state, which may not be the one desired.

The `ra` and `ro` commands need differing numbers of input lines, depending on the prior input. For instance, if you use `ra x` to set the range of only the *x*-axis and there is data present, you will be asked if you wish to have the *y*-axis ranged for the included points. If there is no data present, the question won't be asked. Or, if you select user-defined tick spacing with `ro`, you will be prompted for the spacing specifications.

When the commands `wi`, `pn` or `pk` are used without arguments to specify the window size or pen position, the program will not pause to let you set those values on the pen plotter.

To create a file that includes the argument-substitution feature, you will probably have to use an editor rather than the `mk` command, since the characters `$1`, etc., will be treated literally by the plot program. If, for example, you enter `ty $1 $2 $3` at the keyboard, the program will prompt you for plot types since it won't recognize the arguments as valid numbers.

When the plot command `gd 7` is encountered during execution of a command file, input reverts to the keyboard. You may enter the appropriate commands for this *get-data* mode from the keyboard, and input will revert back to the command file when you exit the mode.

Likewise, when the *subshell* command, `u`, is encountered, input again reverts to the keyboard and stays there until you exit the subshell.

Ordinary user functions (but not necessarily fits) are unable to take input from command files, although there are provisions for passing information to the functions on the command line (see Chapter 11).

Typing a `^C` during execution of a command file or while making a command file will terminate the command or make-file mode and the `PLOT->` prompt will reappear. However, a `^C` during a type 5 (fitting) user function invoked from a command file will leave you within the fit interactive program. If the command file had turned on quiet mode with the `zq` command, the `^C` will not only leave you within the fit, but terminal echo mode will be off. Type `ex` or `^D` to exit the fit and return to the plot program. Echoing will be turned back on. Certain errors also will bring back the PLOT-> prompt, such as trying to initialize a nonexistent filter with zi.

Within user functions, you also can abort a command file by invoking the macro set_error() (see Appendix E). A command file can be run when C-PLOT is first invoked by giving the name of the file as an argument in the shell command line.

When invoking type 5 (fitting) user functions from a command file, you can arrange to have the fitting function read commands from *cmd_file* for a while and then have control returned to C-PLOT. If *cmd_file* contains the lines

```
fn fitfunc.5 –
...
ex
```

or (to use the same fitting function as last time)

```
fn –
...
ex
```

the commands between the fn and ex will be executed by the fitting function. Any commands following ex will be read by C-PLOT.

mk make a command file

provides a means for automatically creating a command file.

```
mk cmd_file
```

Typing mk *cmd_file* changes the prompt from PLOT-> to making->. Commands will be executed as they are entered.

Everything typed from the keyboard will be saved in *cmd_file*, with a few exceptions. Input to the gd 7 command will not be saved, nor will input to subshells or to user-defined functions, including fits, unless the fits are invoked with the —argument described above. When you enter any of these commands, nothing you type will be saved in the command file until you are finished with the command. For example, a u command will create a UNIX subshell, where you may manually execute normal processes. When you return to the plot program on exiting the shell, subsequent commands will continue to be saved in the command file.

It is not possible to do a command file while making another command file or to execute an mk command from a command file. A do command can be inserted into the file being made, but that command file will not be executed until the parent command file is executed.

Most users employ a text editor for creating and maintaining command files, but sometimes find the mk command useful for creating a first draft. The *save* *current format* command, described below, provides an alternative starting point for making command files.

em end making a command file

puts you back into normal mode when making a command file using the mk command.

```
em
```

A ^D also will return you to the normal mode. The em is inserted in the command file but ignored when the command file is read.

sf save current format

saves the current plot-format information in a file.

```
sf [cmd_file]
```

The command and formatting parameters of the current plot will be written to *cmd_file* in a form suitable for use as an input command file.

Typing sf by itself will list the current state of the plot-format parameters on the screen.

ch change target of drawing commands

causes all subsequent graphics filter z commands, whether entered from the keyboard or a command file, to be treated as pen plotter p commands, or vice versa.

```
ch
or
ch z|p|0
```

With no arguments, C-PLOT indicates whether or not the commands are being changed. A z argument causes HP-GL plotter p commands to be treated as graphics filter z commands. A p argument does the reverse. An argument of 0 makes the p and z commands work normally. When changing from z to p, the *erase* command, ze, the *window erase* command, zE and the *quiet* command, zq, are ignored. When changing from p to z, the *select pen velocity* command, pv, is ignored.

You may, for example, wish to use a graphics filter to display the plot on a video terminal while developing a command file. When the command file is fine tuned, you can use the *change target* command to make the plot program substitute pen-plotter p commands for the z commands when it executes the command file.

The `ch` command also can be useful for debugging a `z`-based command file, by changing the target of the drawing command to the pen plotter with no pen plotter initialized. In this case, nothing will be drawn, but you will be able to observe error messages on the screen that might otherwise be obscured by graphics. Also, quiet mode (invoked by the `zq` command) doesn't turn off printing of messages when the pen plotter is the drawing target.

w wait for user to enter `<return>`

is useful when running command files that produce multiple plots. The `w` command gives you a chance to make the program pause so you can view each plot in turn.

```
w
```

The `w` command sends a beep to the terminal, prompts with

```
Hit <return> to continue
```

and waits until you enter a `<return>` or newline. When C-PLOT detects that it is running in the background, the command does nothing. The `<return>` must come from the keyboard whether or not C-PLOT is taking commands from a command file.

Running in batch mode

Batch mode means C-PLOT is run as a background process taking input from a command file. Usage is:

```
cplot [-s] cmd_file [command_file_args] [</dev/null]
```

The `−s` option silences C-PLOT's output, except for error messages, which are still written to the standard error stream.

The optional `</dev/null` ensures that C-PLOT can detect it is running in the background. When C-PLOT is running in the background, the `w` command and other interactive commands are disabled.

Presumably you will want to direct the output somewhere other than to the screen of the terminal you are working on. From the Bourne shell (*/bin/sh*), you could type:

```
cplot ... >/dev/null 2>&1 &
```

This form directs the standard output to the UNIX sink (*/dev/null*), where it won't be seen again and directs the error output (attached to file descriptor 2) to join the standard output (file descriptor 1). Alternatively, you can direct both output streams to a regular file, or direct the streams to two different destinations.

From the Berkeley C-shell (*/bin/csh*), you can direct both output streams in the following fashion:

```
   cplot ... >&record_file &
```

In this example both streams are directed to a file.

If you invoke C-PLOT with a command file, but in the foreground, the program will revert to interactive mode when the command file is exhausted, unless the command file contains the *exit* command, `ex`. An `ex` in a command file always terminates the program.

Chapter 11 Using User Functions

This chapter describes the user function facility and includes instructions for writing and running your own functions.

Commands covered

fn	run user function 1
f#	run user function 1 to 8

How user functions work

User functions run as separate processes under the control of C-PLOT and exchange data and other information with the plot program. (See Appendix C for a list of standard user functions.)

You needn't know very much about programming to create your own functions. Prototype C-language files are included with the C-PLOT package, along with the required overhead modules. C-PLOT creates the files, invokes your favorite editor and runs the compiler for you. For ordinary functions you need only type in the lines of C code that describe your calculation. Appendix E contains additional information on writing user functions.

There are five types of user functions. The type number becomes part of the function name. The types are:

Type	Arithmetic	Description
1	$y = f(x)$	Simple —You give the range for x.
2	$x = f(t)$ $y = g(t)$ $r = u(t)$ $s = v(t)$	Parametric —You give the range for t. (In 3D mode, change r to z.)
3	$x = f(x, y, r, s)$ $y = g(x, y, r, s)$ $r = u(x, y, r, s)$ $s = v(x, y, r, s)$	Operation —Transforms current data. (In 3D mode, change r to z.)
4		Other —You supply data (current data is available).
5		Non-linear fitting —See Chapter 12.

Where user functions reside

Standard locations for user functions are the public function directory, `$CPLOTHOME`/*functions*, and your private function directory. The latter is set by the environment variable `CPLOT_FN_DIR` or by using the `set` fn_dir command within `CPLOT_FN_DIR`. If not explicitly set, the default private function directory is `$HOME`/*functions*, or if `HOME` is also not set, ./*functions*.

When you specify a function name in the commands described below, the program searches for the function according to the following rules.

1) If the function name contains a `/`, the function path is implied in the name, as in /*users*/*moe*/*func.1* or ./*func.2*.

2) If the function exists in your private function directory that version will be used.

3) If there is an executable by the chosen name in the public function directory, that function will be used.

4) Otherwise, a new function will be created in your private function directory.

Public functions

Public functions are located in the directory `$CPLOTHOME`/*functions*. The functions in the public directory can't be edited or compiled in the usual way. Either executables can be copied to the public directory or the functions can be invoked according to rule 1 above.

Editors

C-PLOT will automatically invoke an editor when you create or modify a user function. If you have the environment variable EDITOR set, that editor is used. Otherwise, the default editor, *vi*, is used.

How does it work?

For each type of function, a prototype file is copied from $CPLOTHOME/*prototypes* and given the name you selected. The prototype contains skeleton C subroutines and help information appropriate for the function type you selected. You are then put into your chosen editor. On exiting the editor, the shell script $CPLOTHOME/*bin*/*makefunc* is run from the function directory. That file normally invokes the C compiler to compile your module and link it with the appropriate overhead modules. If your user function contains the string cplot_compile: followed by commands to compile your function, those commands are used instead of the default commands from *makefunc*. The commands can refer to the *make* utility or invoke the C compiler directly. Possible ways of including the information in a function source file are:

```
/*
*   cplot_compile:  make my_func.5
*/
```

or

```
#if 0
    cplot_compile:  make my_func.5
#endif
```

Compiling user functions outside of C-PLOT

The C-PLOT package includes a shell script named *newfunc* that lets you compile functions outside the plot program. When you type newfunc to the shell without arguments, all the files in your current directory that end in the characters .1.c through .5.c will be compiled and linked with the proper overhead modules and libraries to create executable user functions. You also can specify files as arguments, giving either *name.#* or *name.#.* c file names.

The *newfunc* script invokes *makefunc*, described above. You can examine the *makefunc* shell script to see the flags, modules, include directories and make libraries needed to compile user functions if you are interested in constructing *makefile*s to maintain your user functions.

fn, f# run a user function

User functions provide a general purpose interface to read in data, generate new data or modify current data. With user functions, you also can control many of the features of the plot, including values of the axis ranges and the text used for plot labels.

In the simplest case, you give the program the name of a function and its type, and the program leads you through the creation, editing and compiling of the function.

```
fn
fn name.# [start finish intervals]
fn name.#.c
fn .
fn c
fn e
fn k
fn ?
f1 (same options)
or
f# (same options)
```

With no arguments, these commands prompt you for the function type and the function name. If the function is not found using the search rules described above, you are asked if you wish C-PLOT to create a new prototype. If the prototype already exists, but not in an executable version, you are asked if you wish to edit the prototype.

If the executable version exists, but the date of the prototype is more recent than the executable, you are asked if you wish to recompile the prototype. Otherwise, the executable version is run.

If, when the user function returns, it is found to be not compatible with the current version of C-PLOT, you will be informed. You should then recompile the prototype.

If you wish to first edit an existing function, specify the name with the `.c` extension and the program will invoke the editor.

If you already have a function running, typing `fn .` will return you to it. The program will automatically start up the editor for the current function if you type `fn e`. If you type `fn c` the program will automatically recompile the current function.

If a function dies unnaturally on receipt of a signal (floating point exception, segmentation violation, etc.), C-PLOT will print a message informing you of that fact along with the number of the signal.

Type 1 and 2 functions require a range on which to calculate the data. If you don't enter the starting value, finishing value and number of intervals on the command line, you will be asked for them. Note that the number of points generated is one more than the number of intervals. For type 1 functions, the range is in the independent variable, *x*. For type 2 functions, the range is for the parametric variable, *t*. In either case, if you specify a negative number for the number of intervals, the spacing of the points over the range will be logarithmic.

When type 1, 2 and 3 functions are executing, typing a `^C` will cause each point to be printed out. Typing a second `^C` will cause an immediate return from the function. A single `^C` makes a type 4 function return immediately.

When you type `fn` *`name.#`*, the function starts executing from scratch.

C-PLOT lets you keep eight functions going independently. The `fn` command is synonymous with `f1`. The `f2` command refers to the second function. The `f3` command refers to the third function, etc., up to `f8`. Typing `fn k` kills the current function process. The `?` option displays the function number, process ID number and name of each active user function.

See Appendix E for more information on writing type 1 through 4 functions. For examples of user functions, see the source code for C-PLOT's standard included functions in `$CPLOTHOME/`*functions* .

Type 5 functions, the nonlinear least-squares fits, are described in detail in Chapter 12.

Chapter 12 Fitting

In this chapter you will find a summary of the features of the non-linear least squares fitting package. The last section of the chapter explains in detail how to create your own fitting function.

Commands summary

The fit functions are interactive processes in their own right. Some of the commands recognized by the fits work the same way as the corresponding C-PLOT commands. Those unique to the fit process are explained in this chapter. A table of all the available commands is presented on the following page.

As in the plot program, all fit commands are one- or two-letter mnemonics. In table that follows, italic parameters are to be replaced with the appropriate characters for the desired instruction. Optional parameters appear in square brackets. When several parameters are shown separated by vertical lines, you use only one of them with the command. For commands that simply indicate *options*, consult the detailed command description for an explanation of the syntax.

Command	Description
2d|3d	Select data exchange mode
cd [*directory*]	Change directory[1]
ch [p*#*=*value* ...]	Calculate chi squared
do [*cmdfile*]	Take commands from file[1]
em	End making a command file[1]
er	Erase the video screen[1]
ex	Exit the fit program[1]
fc	Set fit criteria and fit options
fi [*options*]	Fit the data points[3]
fv [*value*]	Set fit verbosity
gd [*options*]	Get data points[2]
gp	Get parameter values
h	Get on-line help[1]
lm [*#* [*low high*]]	Set parameter constraints
md [*x* [*y*]] [o] [p*#*=*v* ...]	Make data
mk cmdfile	Make a command file
mr [*x* [*y*]] [/] [p*#*=*v* ...]	Make residuals
pg [*x* [*y*]]	Get points from the plot program[4]
ps [*x* [*y*]]	Send points to the plot program[4]
ra [0|*#*]	Select range to fit
rp [*file*|. [*#*]]	Read parameters from a file
sa [*file*[a|w]]	Save data points[1,3]
sA [*file*[a|w]]	Save plot points[3]
sf [*file*[a|w]]	Save full parameters[1,3]
sF [*file*[a|w]]	Save full parameters and errors[3]
sp [*file*[a|w]]	Save parameters[3]
sP [*file*[a|w]]	Save parameters and errors[3]
u [*command*]	Create a subshell[1]
vp	Select parameters to vary
wt [i|s|n|u]	Select how to weight data points

[1] The commands cd, do, em, er, ex, h, sa and u use the same syntax as the plot program. Consult the previous chapters for descriptions.

[2] The gd command uses the same syntax as the plot program but only implements modes 1, 2 and 3. Provisions are made for reading additional independent variables if the user's function requires them.

[3] The commands fi, sA, sP, sF, sa, sf and sp can have their output directed to a file in addition to the terminal screen. The syntax is

```
cmd cmd_options > filename
cmd cmd_options >> filename
```

The first example will open *filename* for writing, and the second will append to *filename*. In either case, if *filename* is the single character . the same file is used as before, and the output is appended.

[4] The commands pg and ps are only available when the fit process is run as a user function from the plot program.

Introduction to the fitting package

The fits are normally run as type-5 user functions, and you should refer to the general description of user functions (the `fn` command) in Chapter 10. The command-file facility in fit is identical to that in C-PLOT. When invoking fits from plot command files, you can have a separate command file for the fit function, in which case you would enter from the plot program:

```
fn fitfunc.5 cmd_file cmd_file_args ...
```

The command file should terminate with an `ex` to ensure that control passes back to the plot program at the end of the command file.

Alternatively, you can include the fit commands in the C-PLOT command file using the syntax:

```
fn fitfunc.5 ->
...
ex
```

All the commands between the `fn` and `ex` will be executed by the fit function. To continue running the current function you can use:

```
fn -
...
ex
```

The fits also can be run as independent processes from the shell.

An explanation of the nonlinear least-squares fitting algorithm used in this software, the Marquardt algorithm, can be found in chapter 11 of the book Data Reduction and Error Analysis for the Physical Sciences by Philip R. Bevington (New York, McGraw-Hill, 1969).

2d, 3d select data exchange mode

sets how the fit data points will be exchanged with the C-PLOT data points.

```
2d
3d
```

Two sets of data points are maintained: those used for fitting and those exchanged with C-PLOT and used for plotting. When the fit is configured for more than one independent variable, these commands affect how the fit data points are assigned to C-PLOT data when using the `md`, `mr` and `ps` commands and how the C-PLOT data is assigned to the fit data with the `pg` command.

If the fit is configured with only one independent variable, only 2D exchange mode is available.

In 2D exchange mode, the default independent variable for transferring data for the `pg`, `ps`, `md` and `mr` commands is *x1*. To use a different variable, specify its number on the command line. The `ps`, `md` and `mr` commands set C-PLOT *x*, *y* and possibly (with `mr` and `ps`) *y* error-bar values —C-PLOT *x* error bars are set to 0. For the `pg` command, only values for one independent variable are read in. The values of the other independent variables, if any, are left unchanged. When there is more than one independent variable, the `md` command increments each from the entered starting value to the entered finishing value.

In 3D exchange mode, the default independent variables for exchanging data are *x1* and *x2*. To use different variables, specify their numbers on the command lines to `pg`, `ps`, `md` and `mr` The `ps`, `md` and `mr` commands set the C-PLOT *z* data from the fit dependent variable. For the `md` command, you can set the range and the number of intervals for the two independent variables to be transferred to the C-PLOT *x* and *y* data. You enter constant values for any other independent variables. Depending on the ranges and number of intervals give for the independent variables, the trajectory in the *x* - *y* data sent to C-PLOT can form a grid or a line. When making a grid of data, line-control information is added to that data so that when the data points are drawn with `lc` mode on and a line symbol selected, the line will be restarted each time $y = y_{min}$.

ch calculate chi-squared

lets you calculate a value for *chi*-squared based on the current parameters without fitting the data.

```
ch [p#=value ...]
```

The calculated value will be displayed when you enter any of the *save-parameter* commands.

fc set fit criteria and fit options

lets you change certain details concerning how the program does the fits and presents its output.

```
fc
```

When you enter the `fc` command, the program asks you a number of questions. Entering `<return>` after a question retains the default value, which is printed in parentheses. Otherwise you can enter new information. Entering `^D` returns to the interactive prompt.

Weighting the data

The first question asks you how to weight the data. The possible ways to weight the data are given in the table below.

Mode	Weight
Statistical	$w_i = 1/\sqrt{y_i}$
Instrumental	$w_i = 1/\sigma_i$
Other	$w_i = 1/y_i$
None	$w_i = 1$

Remember, you must re-enter the data whenever you change the method of weighting. (You also can select how to weight the data using the `wt` command.)

Number of iterations

Next, the `fc` command prompts for the maximum number of iterations to allow before stopping the fit. The number of iterations also can be set on the command line to `fi` with the `n=`*`value`* option.

Style of printout

The next question asks if you wish to change the printout verbosity. You can choose to have the parameter correlation matrices printed out at each iteration, only after the last iteration or not at all. You also can choose to have the support plane and non-linear confidence limits calculated and displayed after the last iteration. These calculations can be obtained at any time using the undocumented `fp` command whether or not you select the option here. The style of printout also can be set with the `fv` command.

Taylor series fits

If you wish to have the fit only use the Taylor series expansion, you can choose that here. This mode bypasses the entire Marquardt algorithm.

Convergence and statistical parameters

Finally, you are asked if you wish to change the statistical criteria. These variables are concerned with certain details of the fitting algorithm and don't normally need to be altered.

The first variable you can change is the starting value of the Marquardt algorithm compromise parameter, λ. This parameter controls how much of the gradient search versus how much of parabolic expansion is used to determine the direction of the search. The larger λ is, the more the algorithm uses the gradient search.

The next variable is the minimum value allowed for λ. It won't be permitted to go lower than the value selected here.

When the program calculates the derivative of the fitting function numerically with respect to a fitted parameter b_i, it uses values of the function at b_i and at $(1 + \delta)b_i$. You can enter here a value to use for δ. The default value is 0.00001.

The next two values, ε and τ, are important in deciding at what point the fit has converged. They are used in a convergence test that is satisfied if for each parameter b_i and parameter increment δb_i,

$$|\delta b_i| / (\tau + |b_i|) < \varepsilon \,.$$

The parameter increments are the changes in the value of each parameter from the prior iteration. τ mostly functions to prevent division by zero. The default value for both is 0.001.

The parameter γ measures the angle between the direction of steepest descent of the χ^2 hypersurface and the direction of the current iteration. When γ falls below the critical value set here, the course of the fitting algorithm changes as described below under `fi`.

The value of the matrix singularity variable lets you adjust how small the determinant of the parameter correlation matrix can be before giving up on the fit.

The last two variables, *ff* and *tt*, are concerned only with the quantities on the final printout obtained with the `fp` command.

fi fit the data points

starts the fitting algorithm using the current data.

```
fi [[+|-] # ...] [p#=value ...]
   [n=value] [f#=value] [t#=value]
   [L#=value|none ...] [U#=value|none ...]
```

All the options to `fi` can be selected using other commands.

Single numbers, represented by `#`, indicate which parameters to vary. In addition, `+#` means add parameter # to the ones being fitted, while `-#` means don't fit parameter `#`. The `vp` command also can be used to select the parameters to vary. Values for fixed parameters and initial values for fitted parameters can be set using the `p#=value` syntax, where the # represents the parameter number. The `gp` command can be used to enter the same information.

The maximum number of iterations is set using `n=value`. This value also can be set using the `fc` command.

You can set the range of the independent variables using the `ra` command or you can choose the range for each independent variable by entering `f#=value` to set the lower limit for independent variable number # and `t#=`*`value`* to set the upper limit. You can set only the from-value or only the to-value if you like. If # is not specified, independent variable number 1 is used.

Parameter constraints can be set with the `lm` command or by entering `L#=`*`value`* as an argument to `fi` where # is the parameter number. Replace *`value`* with `none` to turn off constraints. Use `U#=`*`value`* or `U#=none` for the upper constraint.

Description of the fit output

For the example that follows, the sample data is listed under the `sa` command below. The fitting function is the fit to a line that appears in the prototype module, described at the end of this chapter.

Before the first iteration of the fit, the program prints information in the following format:

```
July 6, 1992
Fitting 11 points.  2 parameters varied.  0 fixed.  Phi = 514.9.

Number             Name  Is fit?  Deriv?  Initial Value
0         Constant term    YES      YES      0.1
1           Linear term    YES      YES      0.9
```

Phi (Φ) is the sum of the squares of the differences between the *y*-values of the data points and values calculated with the current parameters. The table lists the parameter number, name, whether or not that parameter is being fit, whether or not the user function supplies an analytic derivative for that parameter and the starting value for that parameter.

A typical fit produces output in the form:

```
Iteration #1        Phi = 10.02  Chi-Squared = 1.114
11:01pm (0.2u 0.1s) Lambda = 0.01 Gamma = 23.68 Step = 17.38

0       Constant term = 15.5308 delta = 15.4308
1       Linear term = 3.63904   delta = 2.73904

Iteration #2        Phi = 9.792  Chi-Squared = 1.088
11:01pm (0.3u 0.3s) Lambda = 0.001 Gamma = 64.35 Step = 2.964

0       Constant term = 10.4579 delta = -5.07289
1       Linear term = 4.00195   delta = 0.362915

Iteration #3        Phi = 9.792  Chi-Squared = 1.088
11:01pm (0.5u 0.4s) Lambda = 0.0001 Gamma = 1.533 Step = 0.1285

0       Constant term = 10.2319 delta = -0.226027
1       Linear term = 4.01728   delta = 0.0153272

Convergence by epsilon test in 4 iterations.
```

```
Iteration #4        Phi = 9.792  Chi-Squared = 1.088
11:01pm (0.6u 0.6s) Lambda = 1e-05 Gamma = 0.0179 Step = 0.00056

0       Constant term = 10.2309 delta = -0.000984272
1       Linear term = 4.01735   delta = 6.6703e-05
```

Chi-squared (χ^2) is simply Φ divided by the number of degrees of freedom (the number of fitted points minus the number of fitted parameters). The numbers in parentheses after the time of day are the cumulative user and system CPU times for this fit.

Lambda (λ) is the current value of the Marquardt compromise parameter. *Gamma* (γ) represents the angle between the direction of the gradient at the current point on the χ^2 hypersurface and the direction taken during this iteration. *Step* is the distance in scaled parameter space traveled during this iteration.

At each iteration for each fitted parameter the current value of the parameter is printed along with the change in its value from the last iteration.

The iterations cease when either the iteration limit is reached or one of the convergence criteria is satisfied. The possible convergence messages are:

Convergence by epsilon test. The ε test is the usual convergence test and is performed after each iteration where Φ has decreased for the given value of λ. The test is satisfied if, for each parameter b_i and parameter increment δb_i, $|\delta b_i|/(\tau + |b_i|) < \varepsilon$, where the values of τ and ε are set with the `fc` command.

Convergence by gamma-epsilon test. If Φ hasn't decreased but γ has fallen below the critical value established with the `fc` command, the parameter increments are continually halved and the ε test is performed. If the test is satisfied, you will see this message. If Φ has increased with the halved increments, the previous values of the parameters will be maintained and you will see the message "Correction vector for last iteration not used". If the ε test is not satisfied and Φ decreases with the halved increments, the fit will proceed to the next iteration.

Convergence by gamma-lambda test. If Φ has decreased but γ has exceeded 90 degrees and λ has exceeded 1, this message is printed. The message might more appropriately read "non-convergence", since the conditions causing this message generally indicate a pathological situation possibly caused by errors in the model equation or inappropriate values for the current parameters or the data.

Typing `^C` while fitting halts the fit and restores parameters to their values as of the last completed iteration.

`fv` set fit verbosity

controls how much information is displayed while fitting.

```
fv [mode]
```

Without arguments, you are asked what information you want displayed during the fitting. The numerical value displayed after you have made your selections can be entered as an argument to `fv` the next time to make the same selections. The fit verbosity also can be set using the `fc` command.

`gp` get parameter values

is one way to enter new values or restore old values for the parameters.

```
gp
```

You can use `gp` to change the values of parameters one by one, as prompted by the program. The current values and initial guesses (as compiled into your C-module) are printed for each.

Before being prompted for new values, you are asked if you wish to restore either the compiled-in initial values or the previous values, that is, the values used before the last attempt at fitting with the `fi` command.

`lm` set parameter constraints

lets you set or vary lower and upper constraints on the parameters during fitting.

```
lm
or
lm [# [low high]]
```

With no arguments, you are prompted for lower and upper constraints for each of the adjustable fit parameters. You can enter information for just one parameter by giving the parameter number as an argument and, optionally, the constraints, on the command line. Entering the literal characters `none` for the lower or upper constraint removes the constraint at that end of the range.

The default values for the parameter constraints are set in the C code for your fitting function. You also can enter values for the parameter constraints on the `fi` command line.

During fitting, a parameter with constraints will not be allowed to move past the constrained value. If the fit converges with a parameter at a constrained limit, an error message is included in the fit results.

md make data

generates points over a selected range, or one-to-one with your data, and sends the points to the plot program.

```
md [x [y]] [o] [p#=value ...]
```

Without arguments, you enter starting and finishing values and the number of intervals as prompted by the fit program. Entering a literal `min` or `max` sets the starting or finishing value to the data minimum or maximum. Points are generated that will become the plot program's current data. If you include the `o` argument, you are not asked for a range and points are calculated at the current values of the independent variables for each data point.

You can select which independent variable (if there is more than one) to send to the plot program by entering a value for *x* for transferring 2D data or values for *x* and *y* for transferring 3D data.

You also can set values for parameters using the `p#=value` notation, where the # represents the parameter number.

In 2D exchange mode, when there is more than one independent variable, each is incremented from the entered starting value to the entered finishing value. In 3D exchange mode, you set the range and number of intervals for two independent variables. Depending on the values entered, the trajectory in the *x-y* plane can be a line or a grid. In the case of a grid, line control information is added to the generated data, so that if drawn with `lc` mode on and a line symbol, the pen will be lifted between each y_{max} and y_{min}. Any other independent variables in 3D exchange mode are held at constant values that you also can enter.

When running the fit as a stand-alone process, the generated points can only be used with the `sA` command, described below.

mr make residuals

lets you send to the plot program data representing the scatter in the data points left after subtracting the fitted values.

```
mr [x [y]] [/] [p#=value ...]
```

The values the *make residuals* command sends to the plot program are the differences between fit's current data points and points calculated from the model equation using the current values for the parameters. If you include the `/` argument, each difference is divided by the calculated value of your model at each point.

You can select which independent variable (if there is more than one) to send to the plot program by entering a value for *x* for transferring 2D data or values for *x* and *y* for transferring 3D data.

You can set values for parameters using the p#=*value* notation, where the # represents the parameter number.

When running the fit as a stand-alone process, the generated points can only be used with sA command.

pg get points from the plot program

copies C-PLOT's current data points to fit's data points.

```
pg [x [y]]
```

C-PLOT has many more ways of reading in data than the fit function. Entering pg replaces the current set of points in the fit function with those from the plot program, setting the weights according to the current weight mode.

Only as many points as will fit will be transferred. You select how many fit points are allowed at the top of the prototype C-module for your fit. (See the section " Adding your model equation to the prototype" at the end of this chapter.)

You can select which independent variable (if there is more than one) receives the plot *x* data by entering a value for the *x* argument in 2D exchange mode. In 3D exchange mode, you can enter *x* and *y* arguments to indicate which independent variables receive the plot *x* and *y* data.

The pg command is only available when the fit process is run as a type-5 user function from C-PLOT.

ps send points to the plot program

copies the current set of fit points to C-PLOT.

```
ps [x [y]]
```

You can use ps to send the points you have been fitting to the plot program. Values for the error bars are calculated from the current weights. You can select which independent variable (if there is more than one) to send to the plot program by entering a value for *x* for transferring 2D data or values for *x* and *y* for transferring 3D data.

The ps command is only available when the fit process is run as a type-5 user function from C-PLOT.

ra select range to fit

lets you fit a portion of the current data. You can enter minimum and maximum values for each of the independent variables.

```
ra [0|#]
```

When you enter ra, the program determines the minimum and maximum values for each independent variable, prints them, and then prompts for minimum and maximum values for the fit range.

The command ra 0 sets the ranges to include all the current data. When you have multiple independent variables, you can range just one of them by using the # option.

Once set, the ranges stay in effect even after new data is read in.

rp read parameters from a file

lets you restore the values of parameters that were saved to a file using the sp, sP, sf or sF commands.

```
rp
rp filename [set_num]
or
rp . [set_num]
```

You normally use one of the save-parameter commands (sp, sP, sf or sF) described below to create the file that is to be read. The names of the parameters in the file must match the names in the fit you are running, although values will be read and assigned until a discrepancy in the names is found.

With the . option, the parameters will be read from the same file you used the last time you entered the rp command.

If there is more than one data set in the parameter file, you can select which will be read by entering the set number as an argument.

If you are going to edit a save-parameter file and then use rp to read it back in, you must not change the characters in the parameter names. Initial and trailing spaces around the parameter name are ignored, however, in the comparison of the current parameter names and the names in the file.

sA save plot points

lists the current plot points to the screen or to a file.

```
sA
or
sA filename [a|w]
```

The plot points are those created by the commands ps, md and mr and copied to the fit data points with pg. With this command you can, for example, create data sets from your model equation with different parameter values using md and save the data to a file without having to return to the plot program. The syntax of sA is the same as that for sa described next. Three or four columns of data are displayed depending on whether 2D or 3D exchange mode is in effect.

sa save data points

lists the current data points to the screen or to a file, just as its does in the plot program. There is a difference in the format, though, when the points are written to the screen.

```
sa
or
sa filename [a|w]
```

Refer to the description of sa in Chapter 3 for an explanation of this command. The difference in screen format from the plot program version of sa is shown in the following example. (The data presented here is used in the straight-line fit examples elsewhere in this chapter.)

```
x      y          weight     yfit       residual
10     48.9343    0.142953   50.4044     0.210153
11     56.32      0.13325    54.4217    -0.252944
12     53.9749    0.136114   58.4391     0.607637
13     67.6994    0.121537   62.4564    -0.637217
14     73.7787    0.116422   66.4738    -0.850451
15     80.0557    0.111765   70.4911    -1.06898
16     69.4587    0.119988   74.5085     0.605915
17     64.9841    0.12405    78.5258     1.67984
18     72.0632    0.117799   82.5432     1.23454
19     99.7246    0.100138   86.5605    -1.31823
20     98.2006    0.100912   90.5779    -0.769229
```

The second to last column is the value calculated using the model equation and the current parameters. The last column is the weighted residual, $w_i(y_i - yfit_i)$.

The sa command in the plot program produces the following for the same data.

```
10     48.9343    6.99531
11     56.32      7.50469
12     53.9749    7.34678
13     67.6994    8.22795
14     73.7787    8.58944
15     80.0557    8.94734
16     69.4587    8.33417
17     64.9841    8.06126
18     72.0632    8.48903
19     99.7246    9.98622
20     98.2006    9.90962
```

Here the third column is the error-bar value rather than the weight (which is the inverse of the error-bar value).

`sp`, `sP`, `sf`, `sF` save parameters

The *save parameters* commands let you write out the current parameters to the screen or to a file in several formats.

```
sp
sp filename [a|w]
or
sP, sf and sF (same options)
```

With no arguments, `sp`, `sP`, `sf` and `sF` all write the current set of parameters to the screen in the same format as the last iteration of the fit. The meaning of the values printed is explained under the command `fi`. The commands `sP` and `sF` print out the calculated errors in the fitted parameters, rounding the fitted parameter values to two significant figures of the error. The commands `sp` and `sf` don't print the errors, but do print out six significant figures of each parameter.

With a *`filename`* argument, the format produced by these commands is quite different and is intended to be used with the annotation feature of the plot program. The command `sp` *`filename`* writes to the file:

```
\H@13@\W@-\(*x\u\s2\b\l\d @\(*x\u\s2\b\l\d = 1.088
\H@13@\W@-Constant term@Constant term = 10.2309 \(lh
\H@13@\W@-Linear term@Linear term = 4.01735 \(lh
```

which, when interpreted by C-PLOT by typing `pn` *`filename`*, writes on the plot:

χ^2 = 1.088
Constant term = 10.2309 ☜
Linear term = 4.01735 ☜

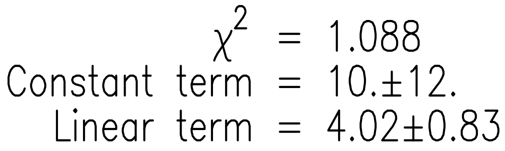
The hand symbols indicate which were the fitted parameters.

The formatting sequences on the parameter lines will make the equals signs line up when used with the annotation command, even with nonproportional fonts. The delimiter character `@` used in the formatting sequences cannot be used in your parameter names, without causing problems for the `rp` command and for the user function *fitpar.4* described in Appendix C.

The command sP *filename* produces:

```
\H@13@\W@-\(*x\u\s2\b\l\d @\(*x\u\s2\b\l\d = 1.088
\H@13@\W@-Constant term@Constant term = 10.\(+-12.
\H@13@\W@-Linear term@Linear term = 4.02\(+-0.83
```

which, when interpreted by C-PLOT by typing pn *filename*, writes on the plot:

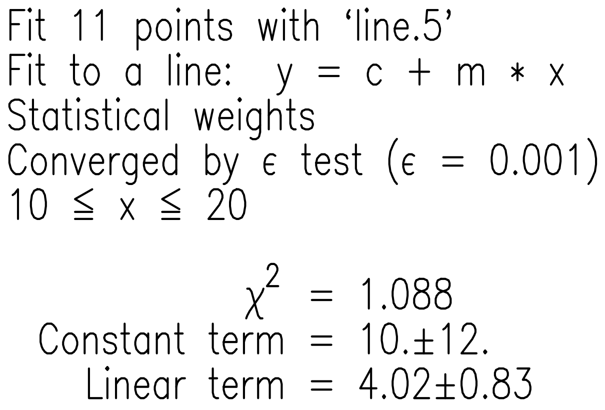


χ^2 = 1.088
Constant term = 10.±12.
Linear term = 4.02±0.83

The command sF *filename* produces:

```
Fit 11 points with 'line.5'
Fit to a line:  y = c + m * x
Statistical weights
Converged by \(*e test (\(*e = 0.001)
10 \(<= x \(<= 20
\H@13@\W@-\(*x\u\s2\b\l\d @\(*x\u\s2\b\l\d = 1.088
\H@13@\W@-Constant term@Constant term = 10.\(+-12.
\H@13@\W@-Linear term@Linear term = 4.02\(+-0.83
```

which, when interpreted by C-PLOT by typing pn *filename*, writes on the plot:



Fit 11 points with 'line.5'
Fit to a line: y = c + m * x
Statistical weights
Converged by ϵ test (ϵ = 0.001)
10 ≦ x ≦ 20

χ^2 = 1.088
Constant term = 10.±12.
Linear term = 4.02±0.83

The first line includes the name of the fitting function. The second line is a comment set in the prototype file. The third line indicates the method of weighting. The fourth line shows how the fit terminated. The next line or lines show the range of each of the independent variables. (In this example there is just one.)

The output from sf *filename* is similar, but the errors are not printed with the parameters.

By indicating a or w after the file name, you tell the program either to append to the file or to write over the current contents of the file. If the file already exists, you must explicitly indicate you wish to write over the file, otherwise the file will not be changed.

The output format that appears on the screen also can be directed to a file using the *>filename* syntax, as explained in the notes to the list of commands at the beginning of this chapter.

The public user function *fitpar.4*, described in Appendix C, lets you read a file containing many parameter sets created using any of the save-parameter commands to produce plots of any parameter versus any other.

`vp` select parameters to vary

provides one way to choose which parameters are to be fit.

```
vp
```

The `vp` command lets you choose one by one which parameters to fit. You also can select which parameters to vary on the command line to the `fi` command.

`wt` select how to weight data points

provides a quicker way to choose the method of weighting data than the `fc` command.

```
wt
or
wt i|s|u|n
```

The choices available for the `wt` command are explained under `fc`. If you don't choose the weighting mode on the command line, the current weighting mode is printed, and you are prompted for a new mode.

Adding your model equation to the prototype

To use the fitting portion of the C-PLOT package, you have to do a bit of C programming to enter the code for your fitting function. You don't need to be an expert programmer, since most of the programming work has been done for you.

In this section, the code from the prototype C module from which you start is presented with explanations of what the code does.

Introduction

Your fitting function can be expressed as

$$y = f(b_1, b_2, \ldots b_m; x_1, x_2, \ldots)$$

where the b_i are the parameters to be fit and the x_i are the independent variables. (Most often, users have only one independent variable.) The fitting algorithm requires derivatives of the fitting function with respect to each fitted parameter. That is, values of

$$\frac{\partial\, y = f(b_1, b_2, \ldots\, b_m; x_1, x_2, \ldots\,)}{\partial\, b_i}$$

for each b_i are needed. If you don’t supply the derivatives analytically, the program will calculate them numerically. It is generally more efficient if you supply the derivatives, since numerical evaluation requires an additional call of your fitting equation for each derivative for each point. You can supply derivatives for some of the parameters, though, and let the program calculate them for others.

Initial definitions

These first lines are included at the beginning of all fitting C modules.

```
#define VERSION 5

#define NUM_XS  1       /* Independent variables */
#define MAXPAR  2       /* Parameters */
#define MAXPTS  1024    /* Most points for fitting */
#define MAXPLT  1024    /* Most for pg ps md mr sA */

#include <math.h>
#include <p_fitsize.h>
```

Lines that begin with a `#` are directives for the C preprocessor. In these instances, either a manifest constant is defined or the contents of another source file are to be included. `VERSION` indicates the version of the prototype file you are using. Its value is used by other preprocessor directives in the included file *p_fitsize.h* and should not be changed.

You select values for the next four definitions. `NUM_XS` selects the number of independent variables you wish to use. Most often its value is one, but you may have as many as you like. Define `MAXPAR` to be your total number of parameters. Its value must agree with the number of elements in the structure `initial` defined below. `MAXPTS` will be the maximum number of data points you can fit at a time. `MAXPLT` limits the number of in-core points of the total (up to 65,636) you can send to or receive from the plot program with the commands `pg`, `ps`, `md`, `mr` and `sA`.

Initial code

In the next lines of code you put in a title and a comment, and you have the opportunity to enter some code that will get called once.

```
char  *title = "Fit to a Line:   y = c + m * x";
char  *comment;
setup() {
        /*
        *  Shown is optional user initialization
        *  of prompts. Second argument must point
        *  to static storage.
        */
        set_prompt(0, "LINE");  /* Main prompt */
        set_prompt(1, "line");  /* Command-file prompt */
}
```

The character-pointer variable `title` should be initialized to the address of a string containing a short message that identifies the function. You can have embedded newlines within the string. The title will be printed each time the fit starts and will be displayed when you type the *print version* `V` command. The character pointer `comment` may be set to point to a string that will be included in the information printed with the `sF` command

The function `setup()` gets called shortly after the fit process starts. You can put one-time initialization code within. For example, the default prompt `FIT>` can be changed by the optional calls to the function `set_prompt()`. In this example the primary prompt is set to `LINE>` and the secondary prompt to `line>`.

Initializing the parameters

The following structure is to be initialized with the parameters you will use in your model equation. The example is for a two-parameter fit to a straight line.

```
struct init_4 initial[] = {
/* Name         Deriv? Fit? Initial Limit? Low High */
{"Constant term",   1,   1,    0.1,     0,  0,   0},
{"Linear term",     1,   1,    0.9,     0,  0,   0},
};
```

The number of lines must agree with the number of parameters set by `MAXPAR` above. The seven elements in each line of the structure are: 1) a character string that identifies the parameter; 2) an integer flag that, if nonzero, indicates your model supplies the analytic derivative of the fitting functions with respect to the corresponding parameter; 3) an integer flag that indicates if this parameter is to be fit initially; 4) the default initial value for this parameter; 5) an integer value indicating whether to constrain the limits —1 for the lower limit, 2 for the upper limit and 3 for both; 6) the value of the lower constraint; and 7) the value of the upper constraint. The latter five can be easily changed while the program is running.

Not all parameters you give must be fitted or even used directly in the fitting equation. For instance, a parameter might be put in only to be used for plotting with the *fitpar.4* user function (see Appendix C).

Referring to parameters

The internal storage of parameters and related values are in the structure array named `fpar`, defined in *p_fitsize.h*. The next lines show how to refer to the parameters in your model equation.

```
#define CONST    fpar[0].p_b
#define fCONST   fpar[0].p_fit
#define dCONST   fpar[0].p_p

#define LINEAR   fpar[1].p_b
#define fLINEAR  fpar[1].p_fit
#define dLINEAR  fpar[1].p_p
```

The purpose of the above definitions is to make referencing the current values of the parameters convenient. The structure member `fpar[i].p_b` indicates the current value of the *i*-th parameter. The other two members are relevant generally only when you are supplying the analytic derivatives for the corresponding parameter. In such a case `fpar[i].p_fit` indicates the parameter is currently being fit and `fpar[i].p_p` is the variable to which you assign the value of the derivative. Their use is made clear below.

Referring to independent variables

These next definitions show how to refer to the current value of the independent variable(s) to be used in the current evaluation of the fitting equation.

```
#define X       (M_flag? Make_x[0]:dp->d_xx[0])
#define X1      (M_flag? Make_x[0]:dp->d_xx[0])
#define X2      (M_flag? Make_x[1]:dp->d_xx[1])
#define X3      (M_flag? Make_x[2]:dp->d_xx[2])
/* etc. */
```

The integer variable `M_flag` is nonzero when evaluation of the fitting equation is being performed on behalf of the *make data* command, `md`, or the *make residuals* command, `mr`, and zero when the evaluation is being performed during fitting. The C-language ternary operator `(?:)` within parentheses evaluates to the expression on the left of the colon if `M_flag` is nonzero and the expression on the right of the colon otherwise.

Notice that the definitions of `X` and `X1` are the same. The latter is for the case of multiple independent variables. You can of course change the defined names to make your fitting equation more readable.

The fitting equation

In the next part of the code you perform the actual evaluation of your fitting equation. The simple example shown is for fitting to a straight line.

```
double  model(deriv_flag)
int     deriv_flag;
{
        double  x, yfit;

        x = X;
        yfit = CONST + LINEAR * x;
        if (deriv_flag) {
                if (fCONST)
                        dCONST = 1;
                if (fLINEAR)
                        dLINEAR = x;
        }
        return(yfit);
}
```

The argument deriv_flag is nonzero when the routine model() must calculate derivatives for the parameters currently being fit. Of course, the routine only need calculate these derivatives if the structure initial indicates they are to be user-provided. If the derivatives are to be user-provided and deriv_flag is set, you still only calculate derivatives if the parameter is currently being fit. In this example, that is the case when fCONST or fLINEAR are nonzero.

Although you could use the definition X for the current value of the independent variable each time you refer to it, the code will be more efficient if it refers to X just once. That is the purpose of the line x = X.

Adding your own commands

Within this C module you can add custom subroutines callable by two-letter mnemonics you enter in response to the interactive prompt.

```
int     prefilter(), postfilter();
struct  user_cmds {
        char    c_one;
        char    two;
        int     (*c_func)();
} user_cmds[] = {
        {'p', 'o', postfilter},
        {'p', 'r', prefilter},
        0,
};
prefilter() {
}
postfilter() {
}
```

Simply add the letters you intend to use and the name of the function in the initialization of the structure `user_cmds`. Also, declare your function name as a function returning an integer as is done for `postfilter()` and `prefilter()` above (although no return value is required). These two subroutines are simply illustrative of the method and can be removed from your C-module or be replaced by your own subroutines.

If the mnemonic you choose conflicts with any of the built-in fit program commands, you will be informed when the program starts and your subroutines will be inaccessible.

If you are interested in the entire line that the user has typed in response to the interactive prompt, the function

```
char *get_cmdbuf()
```

returns a pointer to a buffer containing those characters. You may examine it or parse its contents as you like.

Referring to data within your subroutines

In the file *p_fitsize.h* is the definition of the data structure:

```
struct  f_data {
        unsigned d_flags;   /* Used internally */
        float   d_w2;       /* The square of the weight */
        double  d_y;        /* The dependent variable */
        double  d_ys;       /* Used internally */
        float   d_xx[1];    /* Independent variable(s) */
};
extern  struct  f_data  *f_data, *dp;
#define INC(d)  (d = (struct f_data * ) (((char *) d) + dpsize))
```

The pointer `f_data` is the base of an array of these structures. The pointer `dp` contains the address of the array element associated with the current point when fitting. The macro definition `INC(d)` illustrates how to increment a pointer to an element in the `f_data` array by one element. Such machinations are necessary since the size of an array element depends on the number of independent variables configured.

A fragment of code with which you might loop through all the data points follows:

```
register int  i;
register struct f_data  *d;

for (d = f_data, i = 0; i < npts; i++, INC(d)) {
        if (d->d_flags&D_DONT_FIT)
                continue;
        d-d_xx[0] =
        d-d_xx[1] =
        d-d_xx[2] =
        d-d_y     =
        d-d_w2    =
}
```

This example assumes three independent variables. Of course, the example still requires the right-hand side of the assignment expressions. The externally defined integer `npts` contains the current number of data points. The data flags have the `D_DONT_FIT` bit set if the points are not in the current fit range.

Useful routines for interactive running

Several routines are available to provide easy interaction with the user of your fitting function while maintaining compatibility with quiet mode (used with plot filters) and with command files.

For printing to the screen, use the routine `msg()` rather than `printf()`. The routine `msg()` suppresses output when quiet mode is in effect. Otherwise, its behavior is identical to the standard `printf()`.

For reading input from the keyboard (or command file) the following four routines are available:

```
get_dnum(prompt, num)   /* Input double from user */
char    *prompt;
double  *num;

get_inum(prompt, num)   /* Input integer from user */
char    *prompt;
int     *num;

get_snum(prompt, num)   /* Input string from user */
char    *prompt, *num;

yesno(prompt)           /* Check for positive response */
char    *prompt;
```

In each of these routines, if `prompt` is nonzero, the string it points to is printed, and in the first three routines, the current value of the double, integer or string pointed to by `num` also is printed. In each case, a line of text is then read from the keyboard (or command file) and scanned for something to stuff into the location pointed to by `num`. If no appropriate value is found on the line of text, the contents of `num` remain unchanged.

The return values of the first three functions are: 1 if the user simply enters `<return>`(`num` unchanged); 0 if the user entered something; and −1 if there was an end-of-file (the command file finished or the user entered a `^D`). (Also, the externally defined integer `eofflag` is nonzero on an end-of-file condition after attempting input.)

The return values of the routine `yesno()` are: 1 if the input from the keyboard (or command file) begins with `y`, `Y` or `1`; `0` if nothing is entered; and −1 for anything else. Examine the value `eofflag` to distinguish an end-of-file from a negative response.

Here is a sample code fragment showing how these routines might be used.

```
char    file[128] = "good_data";
double  norm = 1;

new_options() {
        if (yesno("Change options (NO)") > 0) {
                if (get_snum("New file name", file) == 0) {
                        /* open the file */
                }
                if (eofflag)
                        return;
                if (get_dnum("Normalization", &norm) == 0)
                        return;
        }
        if (eofflag)
                return;
        /* and so on */
}
```

A typical dialogue produced by the above might look like:

```
Change options (NO)? y
New file name (good_data)? something_else
Normalization (1)? 876.1
```

Notice how the current values are given in parenthesis. If the user enters a `^D` as a response to any of the prompts, the function will return to the calling routine.

Optimizing

In version 4 of the C-PLOT package, there is a second variation on the fitting algorithm. This second variation is only relevant when you do not provide analytic derivatives for all the fitted parameters. For certain fitting functions you may be able to make your calculations more efficient with this new variation (at the expense of much more memory usage). Both variations produce the same fit results.

The differences relate to the order in which the model() function is called. While doing the fit iterations in the original method, for each data point in turn, the model() equation is called once with the current parameter set and then an extra time for each parameter that did not have analytic derivatives provided.

The new method always calls model() for all the data points each time there is a change of parameter values. During a fit iteration, model() will be called for each data point in turn with the current parameter set. Then, for each fitted parameter that does not have analytic derivatives provided, model() will be called for each data point in turn with the varied value of the fitted parameter. For users who need to do lengthy calculations that change for each parameter set but are the same for each data point, this new method makes it possible to speed the fitting process.

The only drawback to the new method is that it allocates additional memory equal to MAXPTS * MAXPAR * sizeof(double).

The default behavior is to use the old calling sequence. To use the new calling sequence, add the line:

```
#define VECTORIZE 1
```

to your prototype before *p_fitsize.h* is included.

In order to take advantage of the new variation, you need to know at what point in the fitting algorithm the model() function is being called. That information can determined from the arguments to model().

The model() function is called with four integer arguments:

```
model(deriv_flag, m_mode, par_num, point_num)
```

The argument deriv_flag has already been described. It is set if model() must calculate the analytic derivatives during this call.

The argument m_mode is set to one of the constants defined in *p_fitsize.h* to indicate from where in the code model() is being called. These constants are:

```
                          /* Called from ... */
#define FIT_VALUE       2 /* ... "fi" during fitting */
#define FIT_PARTIAL     3 /* ... "fi" to calculate partial */
#define FIT_SUMSQ       4 /* ... "fc" or "ch" */
#define CALC_DATA       5 /* ... "sa" */
#define MAKE_RESIDUALS  6 /* ... "mr" */
#define MAKE_DATA       7 /* ... "md" */
```

The argument par_num is normally −1. When model() is called with m_mode equal to FIT_PARTIAL, the parameter described by fpar[par_num] has been incremented (for calculating the partial derivative) from its value when model() was last called with *mode* equal to FIT_VALUE.

The argument point_num is the index into the f_data array. If the fit range does not include all the data, point_num will not necessarily have a value of zero at any time. Use code similar to the following to check for the first point:

```
model(deriv_flag, m_mode, par_num, point_num) {
        static  int     prev_point = 32000;  /* a large number */
        int     first;

        if (point_num < prev_point)
                first = 1;      /* first in loop */
        else
                first = 0;      /* not so */
        prev_point = point_num;
        ...
}
```

Note that if you can't provide analytic derivatives, you could still improve the performance of the computations by calculating the partial derivatives yourself. When deriv_flag is set, find the value of the fitting function at

```
y0 = f(a[j] = v)
```

and at

```
y1 = f(a[j] = v + del)
```

and set the parameter's derivative to (y1 - y0) / del. By calculating the derivatives for all the parameters within model(), you can avoid redundant computations and speed up the fitting substantially.

Appendix A Setting up the Site

In this appendix you will find information on setting up site files and user files for C-PLOT. It shows which shell variables are taken by C-PLOT from each user's environment and how C-PLOT uses them. There is an explanation of how to set up the proper compiler options for C-PLOT's user functions, and there are instructions for setting up filters for your printers and display terminals. The appendix also includes a list of the standard files that are part of the plot package and a description of the configuration files that are read by C-PLOT each time it starts up. Also, the appendix explains how to optimize the terminal capabilities database to make the most of C-PLOT's PseudoGraphics feature.

Installing the software

The C-PLOT distribution is generally supplied on magnetic media in *tar* format. The amount of disk space required for installation varies from computer to computer, but is usually 1.5 to 2.5 megabytes. The directory hierarchy on the tar media has the same structure as the final installation, so you need only choose the place in your file system hierarchy to locate the C-PLOT files. C-PLOT expects to find its auxiliary files either in the directory */usr/cplot* or in the directory */usr/local/cplot*. You can locate the C-PLOT files somewhere else, however, and make */usr/cplot* (or */usr/local/cplot*) a symbolic link to that location (if symbolic links are available with your version of UNIX) or have each user set the environment variable `CPLOTHOME` to that location.

Once you have decided where to put C-PLOT, make that directory, change to it if necessary and then extract the contents of the tar tape or disk. The distribution media usually has the *tar* command arguments you need for your system printed on the label.

If extracting from a floppy disk, the disk will contain a compressed tar file of the C-PLOT distribution. After extracting the file, named *cplot.tar.Z*, you need to type the commands

```
zcat cplot.tar.Z | tar xvf -
```

You can then remove the *cplot.tar.Z* file. (If *zcat* is not available on your system, contact Certified Scientific Software for assistance.) You need to make */usr/cplot/bin* (or `$CPLOTHOME`/*bin*) part of your `path` (or `PATH`) environment variable so that UNIX shell programs can find C-PLOT. One way to do this is by adding

```
set path=(/usr/cplot/bin $path)
```

to your *.login* file if you use */bin/csh* or

```
PATH=/usr/cplot/bin:$PATH
```

to your *.profile* file if you use */bin/sh*.

Standard files

The files and directories that make up the standard distribution are located in `$CPLOTHOME`, as shown in the following list:

- □ *bin* (C-PLOT executables)
- □ *cmdfiles* (place for system command files)
- □ *cplot_config_*
- □ *demos* (demo command files)
- □ *filter_tools* (filter-building modules)
- □ *filters* (installed filters)
- □ *fonts* (font data files)
- □ *functions* (public user functions)
- □ *help* (help files)
- □ *help_tools* (tools for formatting help files)
- □ *include* (header files for user functions)
- □ *overhead* (modules to be linked with user functions)
- □ *prototypes* (prototype user functions) All the above are directories, except for the prototype site-initialization file, *cplot_config_*, described next.

The site initialization file

The distribution contains a file called *cplot_config_*. You should rename this file *cplot_config* the first time you install C-PLOT. Each time C-PLOT starts it reads in the values of a number of variables from the file *cplot_config*. You may wish to customize these parameters for your installation.

Name	Default	Explanation
`BAUD`	`2400`	Baud rate of pen plotter; can be argument to `in` command
`CFLAGS`	`-O`	Compiler flags for user functions[1]
`CLIBS`	`-lm -ltermlib`	Libraries to be linked with user functions[1]
`DEVICE`	`/dev/plotter`	Default device for pen plotter; can be argument to `in` command
`FILTER`	`x11`	Initial graphics filter; can be argument to `zi` command[1]
`FILTER1`	`x11`	Same as `FILTER`[1]
`FILTER2`	`testfilter`	Initial second graphics filter[1]
`GPIB`	`0`	Nonzero if plotter is on GPIB interface
`NPTS`	`8,192`	Maximum number of in-core points[2]

[1] Defaults vary depending on the platform.

[2] Additional data points (up to 65,635) are retained in temporary files. (An unlimited numbers of points may be plotted.)

Edit the file by hand if you want to make changes. Values in a *.cplot_init* file in a user home directory take precedence over values in the *cplot_config* file. Not all the variables need be set in each file. The value used for a variable is the one obtained from the last file read.

What plot gets from the environment

Several shell variables are normally set when you log on to the computer — some automatically by the login process, others from the contents of your *.profile* files if you are using the Bourne shell, */bin/sh*, or your *.login* file if you are using */bin/csh*.

C-PLOT looks at the following environment variables:

Variable	Default	Description
CPLOT_DO_DIR	$CPLOTHOME/*cmdfiles*	Alternate command file directory[1]
CPLOT_FN_DIR	$HOME/*functions*	User's function directory
CPLOT_GD_DIR	.	Alternate data directory[1]
CPLOTHOME	*/usr/cplot*	Location of C-PLOT auxiliary files
EDITOR	*vi*	user function editor
HOME	.	User's home directory
SHELL	*/bin/sh*	Program to spawn with u command
TERM	vt100	Terminal name or type
TMPDIR	*/tmp*	Directory for temporary files

[1] The value may be a colon-separated list of directories.

The last four names are standard variables used by many programs. The first four are particular to this package. To set an environment variable in a *.profile* file appropriate for */bin/sh*, you would put lines in the file similar to:

```
CPLOT_DO_DIR=/usr/joe/cmdfile
export CPLOT_DO_DIR
```

For a *.login* file appropriate for */bin/csh*, would put a line similar to:

```
setenv CPLOT_DO_DIR /usr/joe/cmdfiles
```

When you create subshells from C-PLOT, the variable CPLOTLOCK is added to the environment. Any time an instance of C-PLOT starts up, it prints a warning message if CPLOTLOCK is present in the environment, to help you avoid inadvertently running nested versions of C-PLOT.

Help files

There are a number of files in the help directory. The contents of the files called *news* and *motd* are printed each time C-PLOT starts. The file *news* is included in the distribution and will be overwritten when C-PLOT is updated. You also can write your own help files. Any text file in the help directory is accessible through C-PLOT's help facility. The help file *help_fmt* includes instructions for setting up help files so they will be formatted for your screen or window size and will print properly. The formatting syntax is based on the *nroff*/*troff* utilities;

nroff/*troff* macros and a *Makefile* are included in the *help_tools* directory to assist in producing hard copies of the files. See the comments in the *Makefile* and *head.man* files, in the *help_tools* directory, for more information on printing.

Installing graphics filters

A number of precompiled graphics filters are included in the *filters* subdirectory. For those that produce output for a printer, you should create a *spoolers* file as described in the file *filter_tools*/*spoolers*. You also may wish to compile additional filters from the files in the *filter_tools* directory. Consult the *README* file in the *filter_tools* directory for more information. Compiled filters should be moved or copied to the *filters* directory.

Optimizing terminal capabilities for PseudoGraphics

Most installations of UNIX operating systems include a facility for describing the capabilities and special character sequences of terminals in a uniform way. On some versions of the operating system, the file /*etc*/*termcap* contains the capabilities. On other systems, files in the directory /*usr*/*lib*/*terminfo* are used. The following *termcap*/*terminfo* capabilities are used for PseudoGraphics, where the two-letter code is the conventional name of the capability.

Code	Description
`as`	Start alternate character set
`ae`	End alternate character set
`cl`	Clear screen and home cursor
`ho`	Home cursor
`cm`	Cursor motion
`up`	Cursor up
`md`	Begin bold mode
`so`	Begin stand-out mode
`se`	End stand-out mode
`ti`	Initialize terminal
`te`	Reset terminal
`li`	Lines
`co`	Columns

Values for the number of lines and columns on the tty screen are taken from the environment variables `LINES` and `COLUMNS`, rather than the *termcap* or *terminfo* databases, if available. If these values aren't present, the defaults of 24 lines and 80 columns are used. If the `TIOCGWINSZ` *ioctl()* mode is present on windowing systems, that command is used to obtain the window size.

Stand-out mode is used in the C-PLOT package when highlighting data points in the video-display version of the command `gd 7` (see Chapter 3), as well in highlighting text displayed with the help command.

The capabilities `as` and `ae` are often missing from standard data bases. Their function is to start and end translation to the terminal's alternate character set. The PseudoGraphics features use characters from the alternate character set, if available, to outline the axis, show the tick marks and plot the points. The codes for the alternate characters themselves are not generally part of the terminal capabilities databases. Thus, the codes for these characters are built into the plot program, but only for a selected number of terminals.

There are five alternate character sets built into the program. One of these character sets is automatically chosen when the plot program starts up based on the value of the `TERM` variable. The recognized names are shown in the following table. The first entry is used when `TERM` doesn't match any of the other names.

Special Characters	TERM names	Other name
none	dumb	plot0
VT100	vt100 vt52 v132 xterm	plot1
Zenith	h19 Z19 z29 Z29	plot4
IBM PC	ibmpc pc PC	plot5
Televideo	tvi950	plot6

If one of the built-in character sets might be appropriate for your terminal but the `TERM` name you use isn't built into the program, you can use the `gr` command to select the appropriate name from the last column of the table. If the plot program can't find the name you give with the `gr` command in the terminal-capability database, it will switch to the alternate characters associated with that name but will continue to use the control sequences it had been using.

Compiling user functions

C-PLOT automatically invokes the *makefunc* script in `$CPLOTHOME/`*bin* to create executable files from the user's code. By default the *makefunc* script will invoke the C compiler. The flags that should be passed to the compiler and the libraries that need to be loaded with the code vary from operating system to operating system. The flags and libraries are taken from the *cplot_config* initialization file.

Normally, the flags supplied with your distribution of the software are appropriate for your site. You may want to make some changes, however. For instance, you may want to have the name list stripped from user function executables to conserve disk space. On the other hand, you may want to keep the name list so you can run a debugger on your functions.

Sample user functions and demos

Each distribution comes with an assortment of user functions in the public function directory. Some of these are of general utility and some are provided as entertainment or examples of how to write user functions. Descriptions of many of these user functions are given in Appendix C.

When you have finished installing C-PLOT, you may want to try running some of the demonstration files included with the distribution. Change to the *demos* directory and examine the *README* file for instructions. Copies of the demo files are in Appendix D.

Appendix B Plot Fonts

About the fonts

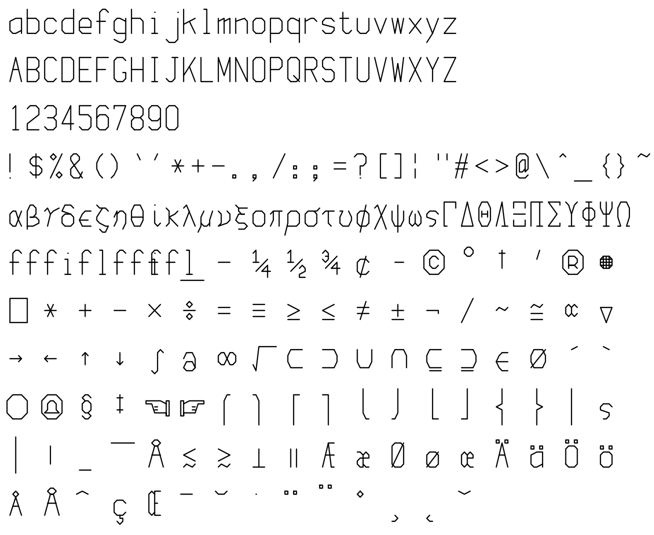
This appendix contains samples of the different fonts available for drawing text on the pen plotter and the graphics filter devices.[1]

The character sequences used to access the Greek, technical and other special characters are described in the last section of Chapter 6. All fonts contain all 94 printable ASCII characters. As of this writing, only fonts 0 and 2 contain all the special characters.

The samples presented here were drawn using the PostScript graphics filter (psfilter) on an Apple LaserWriter. The character size was set to 4mm using the cs command (see Chapter 6).

Font 0

Font 0 is the default font and is built into the program. The characters are drawn on a 6×8 grid.


abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $%&()`'*+-.,/:;=?[]¦"#<>@\^_{}~
αβγδεζηθικλμνξοπρστυφχψωςΓΔΘΛΞΠΣΥΦΨΩ
ff ffi fi fl ffl ſt ſſ ı _ – ¼ ½ ¾ ¢ - © ° † ′ ®
□ ∗ + − × ÷ = ≡ ≥ ≤ ≠ ± ¬ / ~ ≅ ∝ ∇
→ ← ↑ ↓ ∫ ∂ ∞ √ ⊂ ⊃ ∪ ∩ ⊆ ⊇ ∈ ∅ ´ `
○ @ § ‡ ☜ ☞ ⎛ ⎞ ⎡ ⎤ ⎝ ⎠ ⎣ ⎦ { } | ς
| ı _ ‾ Å ≲ ≳ ⊥ ∥ Æ æ Ø ø œ Ä ä Ö ö
Å Å ^ ç Œ ¯ ˘ · ¨ ˝ ˚ ¸ ˛ ˜

[1] Fonts 1 through 8 consist partly of characters from the Hershey Fonts originally created by Dr. A.V. Hershey while working at the U.S. National Bureau of Standards.

Font 1

Font 1 is a medium-resolution font with characters drawn on a 13×13 grid. The thick strokes consist of two lines.

abcdefghijklmnopqrstuvwxyz

ABCDEFGHIJKLMNOPQRSTUVWXYZ

1234567890

!$%&()‘’*+−.,/:;=?[]|"#<>@\^_{}~

αβγδεζηθικλμνξοπρστυφχψωςΓΔΘΛΞΠΣΥΦΨΩ

ff fi fl ffi ffl – ° † ′

+ − × ÷ = ≡ ≧ ≦ ≠ ± ∼ ∝ ∇

→ ↑ ← ↓ ∫ ∂ ∞ √ ⊂ ⊃ ∪ ∩ ∈ ´ `

○ § ‡ | ς

| | ‖

Font 2

Font 2 is a high-resolution font drawn on a 21×21 grid. The characters are drawn with strokes one line thick.

abcdefghijklmnopqrstuvwxyz

ABCDEFGHIJKLMNOPQRSTUVWXYZ

1234567890

!$%&()‘’*+−.,/:;=?[]|"#<>@\^_{}~

αβγδεζηθικλμνξοπρστυφχψωςΓΔΘΛΞΠΣΥΦΨΩ

ff fi fl ffi ffl _ — ¼ ½ ¾ ¢ – © ° † ′ ® ●

□ * + − × ÷ = ≡ ≧ ≦ ≠ ± ¬ / ∼ ≅ ∝ ∇

→ ← ↑ ↓ ∫ ∂ ∞ √‾ ⊂ ⊃ ∪ ∩ ⊆ ⊇ ∈ ø ´ `

○ 🔔 § ‡ ☜☞ ⎛ ⎞ ⎡ ⎤ ⎝ ⎠ ⎣ ⎦ ⎨ ⎬ | ς

| | _ ‾ Å ≲ ≳ ⊥ ‖ Æ æ Ø ø œ Ä ä Ö ö

Å Å ˆ ç Œ ¯ ˘ ˙ ¨ ¨ ˚ ¸ ˛ ˇ

Font 3

Font 3 is a high-resolution font drawn on a 21×21 grid. The thick strokes consist of two lines.

abcdefghijklmnopqrstuvwxyz

ABCDEFGHIJKLMNOPQRSTUVWXYZ

1234567890

!$%&()‘’*+−.,/:;=?[]|"#<>@\^_{}~

αβγδεζηθικλμνξοπρστυφχψωςΓΔΘΛΞΠΣΥΦΨΩ

ff fi fl ffi ffl ° † ′

+ − × ÷ = ≡ ≧ ≦ ≠ ± ∼ ∝ ∇

→ ← ↑ ↓ ∫ ∂ ∞ √ ⊂ ⊃ ∪ ∩ ∈ ´ `

○ 🔔 § ‡ | ς

| | ⊥ ‖

Font 4

Font 4 is a high-resolution font drawn on a 21×21 grid. It is the fanciest of the Roman fonts, with thick strokes consisting of three lines. In addition, many short strokes are used to contour the serifs.

abcdefghijklmnopqrstuvwxyz

ABCDEFGHIJKLMNOPQRSTUVWXYZ

1234567890

!$%&()‘’*+−.,/:;=?[]|"#<>@\^_{}~

αβγδεζηθικλμνξοπρστυφχψωςΓΔΘΛΞΠΣΥΦΨΩ

ff fi fl ffi ffl ° † ′

+ − × ÷ = ≡ ≧ ≦ ≠ ± ∼ ∝ ∇

→ ← ↑ ↓ ∫ ∂ ∞ √ ⊂ ⊃ ∪ ∩ ∈ ´ `

○ 🔔 § ‡ | ς

| | ⊥ ‖

Font 5

Font 5 is a high-resolution, double-line stroke, sans serif font drawn on a 21×21 grid.

abcdefghijklmnopqrstuvwxyz

ABCDEFGHIJKLMNOPQRSTUVWXYZ

1234567890

!$%&()‘’*+−.,/:;=?[]|"#<>@\^_{}~

αβγδϵζηθικλμνξοπρστυϕχψωςΓΔΘΛΞΠΣϒΦΨΩ

ff fi fl ffiffl_ — ¼ ½ ¾ ¢ − © ° † ′ ® ●

□ ❄ + − × ÷ = ≡ ≧ ≦ ≠ ± ¬ / ~ ≅ ∝ ∇

→ ← ↑ ↓ ∫ ∂ ∞ √ ⊂ ⊃ ∪ ∩ ⊆ ⊇ ∈ ø ´ `

O 🔔 ‡ ☜☞ ⎛ ⎞ ⎡ ⎤ ⎝ ⎠ ⎣ ⎦ ⎨ ⎬ | ς

| | _ ‾ ⊥ ‖ Ø ø Ä ä Ö ö

Å

Font 6

Font 6 is a high-resolution, triple-line stroke, Gothic font drawn on a 21×21 grid.

abcdefghijklmnopqrstuvwxyz

ABCDEFGHIJKLMNOPQRSTUVWXYZ

1234567890

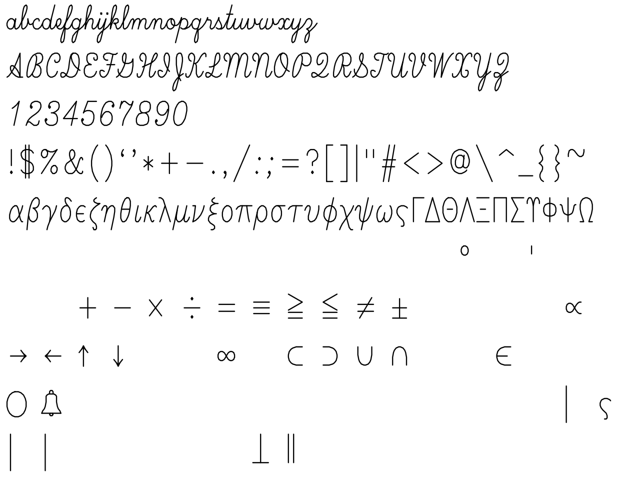
!$%&()‘’*+−.,/:;=?[]|"#<>@\^_{}~

° ′

×

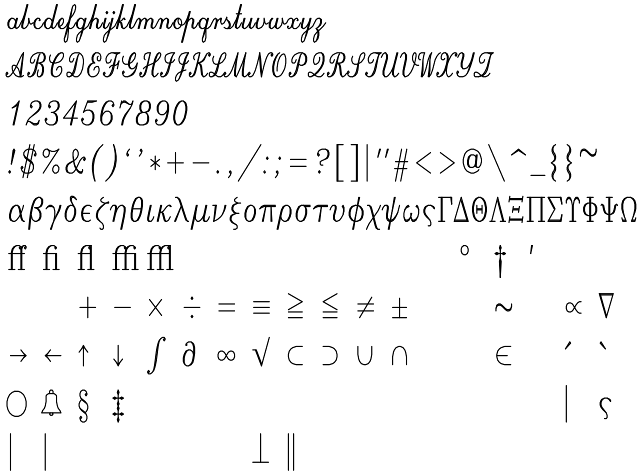
Font 7

Font 7 is a high-resolution, single-line, script font drawn on a 21×21 grid.



Font 8

Font 8 is a high-resolution, double-line, script font drawn on a 21×21 grid.



Appendix C Standard User Functions

This appendix contains descriptions of the standard user functions supplied with C-PLOT. The functions and source code (if provided) reside in the public function directory `$CPLOTHOME`/*functions*.

User functions covered

Name	Description	Purpose	Source
calc.4	Data calculator	Utility	No
chaos.1	Generates data that bifurcates	Fun	Yes
contour.4	Generates contour plots	Utility	Yes
curves.2	Generates geometric curves	Fun	Yes
fft.4	Does fast Fourier transforms	Utility	Yes
fitpar.4	Plots fit parameter files	Utility	Yes
hairs.4	Positions cursor over points	Utility	No
hist.4	Histograms frequency of data	Utility	Yes
psych.4	Generates psychedelic curves	Fun	Yes
scans.4	Reads scan data from files	Utility	Yes
shell.4	Runs any program to filter data	Utility	Yes
smooth.4	Smooths using boxcar average	Utility	Yes
sort.4	Sorts data	Utility	Yes
spline.4	Interpolates using cubic spline	Utility	Yes

calc.4

Calc.4 is a general-purpose user function for manipulating your current data or creating new data.

```
fn calc.4
fn calc.4 expression [; expression ...]
or
fn . .
```

Calc.4 uses a grammar consisting of vector names (*x*, *y* etc.), variable names (upper-case letters), arithmetic operators (+, −, =, etc.), and function names (`sin()`, `sqrt()`, `log()`, etc.). The arithmetic expressions you enter are performed for each point in the current data. For instance, if you enter an expression such as `y = 2 * x`, the value of each *y* in your data set will be replaced with twice the value of the corresponding *x*. Syntax rules and examples follow.

Lower-case letters

The letters `n` and `i` have special meaning. By assigning to `n` you can set the number of points. When `i` is used on the right side of an equation, its value is the index number of the data point. (Index numbers start at 0.)

Valid vector names are `x`, `y`, `z`, `r`, `s`, `t`, `t0`, `t1` and `t2`. Both `r` and `z` refer to the same data vector, representing x error bars in 2D mode and z data in 3D mode. The vector `t` refers to a data vector internal to *calc.4* that can be used to hold intermediate results. This data vector retains its values through subsequent calls to *calc.4*. The vector `t0` refers to the same storage as `t`, while `t1` and `t2` are additional temporary vectors.

You can assign values to vectors by placing them on the left side of an equals sign. You can use their current value by placing them on the right side of the equals sign.

Upper-case letters

The upper-case letters `A` to `Z` represent single-valued variables. They retain their values on subsequent calls to *calc.4*. Variables can appear on either side of an equation. For instance, you may have:

```
fn calc.4  A = 1.4; B = 4.4e-2; C = 3.22e-4
fn .  y = A*x + B*x*x + C*x*x*x
```

or

```
fn calc.4  n = 101; W = i*pi/50; x = cos(W); y = sin(W)
```

Assignment to variables only occurs if the number of current points is nonzero. If you have no current points, you may use `n = 1` to force evaluation of your expressions.

Operators

The arithmetic operators are, in order of precedence:

parentheses for grouping	:	`(  )`
unary plus and minus	:	`+  -`
exponentiation	:	`^`
multiply, divide, modulus	:	`*  /  %`
addition and subtraction	:	`+  -`
assignment	:	`=`
assignment with operator	:	`+=  -=  *=  /=  %=  ^=`

As in the C language, the operation x += 1 means $x = x + 1$.

Functions

The following math functions are available:

exp(), exp10()	:	powers of e and of 10
log(), log10()	:	logarithms base e and base 10
abs() or fabs()	:	absolute value
sqrt()	:	square root
sin(), cos(), tan()	:	sine, cosine, tangent
asin(), acos(), atan()	:	inverse sine, cosine, tangent
sinh(), cosh(), tanh()	:	hyperbolic sine, cosine, tangent
rad()	:	degrees to radians
deg()	:	radians to degrees
j0(), j1()	:	Bessel functions of first kind
y0(), y1()	:	Bessel functions of second kind
erf()	:	error function
erfc()	:	1 – erf()
int()	:	returns integer part of argument
rand()	:	returns random number between 0 and the argument
step()	:	returns 0 if *arg* < 0, 1 otherwise

All trigonometric functions use radians. The C-library `rand()` routine supplies the random numbers —thus the numbers returned cannot be relied on for meaningful statistical analysis.

Other things

The name `pi` is special. It represents the constant 3.14159....

Only one line of input is allowed, but you can have multiple expressions on that line separated by semicolons. When you have multiple expressions, the whole group is evaluated for the first data point ($i = 0$), then for the next data point ($i = 1$) and so on. If you have no points, the expressions won't be evaluated.

An expression preceded by the word `once` will only be evaluated for the first point ($i = 0$).

You can enter the expressions on the command line from C-PLOT or when prompted. If you enter just a dot as the expression, as in `fn .` ., the same input line is used as the last time.

Spaces aren't significant in the expressions.

Error handling

Syntax errors from bad grammar are noted with the offending character indicated. On the following error conditions —dividing by zero, taking the square root of a negative number, taking the logarithm of 0 or a negative number, raising 0 to a non-positive exponent and raising a negative number to a non-integral exponent —an error message is printed showing the index number of the current point. The offending operation is not performed and calculations continue. Other error messages may be produced by the math library routines if they have problems with your numbers.

Examples

Create a circle of 101 points, radius 3.

```
fn calc.4 n = 101; x = 3 * cos(pi*i/50); y = 3 * sin(pi*i/50)
```

Invert x, set error bars to the square root of y.

```
fn . x ^= −1; s = y ^ .5
```

or

```
fn . x = 1 / x ; s = sqrt(y)
```

Create a cosine curve of 201 points with x running from −10 to 10 radians.

```
fn . n = 201; y = cos(x = −10 + 20 * i / 200)
```

Increment the variable A, and increment each y by 0.2 times A.

```
fn .  once A += 1; y += A * 0.2
```

Note the `once` means A gets incremented once. You can now enter `fn .` to increase the y values by another step, perhaps repeating this operation to plot a series of curves offset by constant increments.

chaos.1

Chaos.1 is a simple user function that implements the recursive relation $y_{n+1} = x y_n (1 - y_n)$.

```
fn chaos.1
```

You choose the range for x and enter a value for y_0.

Of course, x_0 had better be non-zero. Appropriate values are $y_0 = 0.5$ and 0<x<4. You should draw the resulting data with symbol 9 (a dot).

contour.4

The *contour.4* user function generates contour plots.

```
fn contour.4  [options]
```

Valid options are:

.	:	use same options as last time
`+g` or `-g`	:	data are (or aren't) a perfect grid
`+n` or `-n`	:	data are (or aren't) a near grid
`+s` or `-s`	:	data are (or aren't) properly sorted
`+r` or `-r`	:	reuse (or don't) previous data
`+v` or `-v`	:	show (or don't) function progress
#	:	number of contour intervals (default 10)
`zmin=#`	:	*z* value of first contour
`zmax=#`	:	*z* value of last contour
`xcol=#`	:	number of columns of data
`xmin=#`	:	use this value to window *x* data
`xmax=#`	:	use this value to window *x* data
`ymin=#`	:	use this value to window *y* data
`ymax=#`	:	use this value to window *y* data

The default options are: `+gsv -r 10`.

The default minimum and maximum of *x*, *y* and *z* are the data minimum and maximum. In the default case, the number of columns of data is estimated automatically. In 2D mode the *z* values are taken from the *x*-error-bar column. The numbers returned can be plotted in 2D or 3D mode.

By indicating the data are a " perfect" or " near" grid and/or " properly" sorted, you can save time otherwise used in unnecessary preprocessing of the data.

A perfect grid means that if there are *N* columns and *M* rows, the total number of points is *N* times *M*. In addition, the *x* and *y* values each need to be equally spaced. In the current version of *contour.4*, a different algorithm is used for data in a perfect grid than for the other case. The perfect-grid algorithm returns smoother contours than does the alternate algorithm. Also, the data points for each contour are in consecutive order, so that they may be plotted with patterned lines. With the alternate algorithm, the data points are returned as line segments that form columns on the page.

A near grid requires the *x* data to be in columns, although not all columns must have the same number of points.

Properly sorted data means the data are sorted by increasing values of x. Data with the same x values are sorted by increasing values of y. If x values within a given column of data are not identical, data points within that column must be sorted by y, with the sort by y taking priority over the sort by x.

If the data are a grid, the function determines the number of columns from the number of points in the first column, which ends when the y values first turn over. If the data are not a grid but are sorted, or if the data are a " near" grid, the function counts the number of columns by counting the number of times the y values turn over.

If the data are not a grid and are not presorted, and if you don't specify the number of columns of data, the function estimates that number using the following rules:

After the function sorts the data by x, it counts the number of times the y values turn over. If this number is less than the maximum number of columns (1024) and less than one-third the number of points, that value is used. Otherwise, the number of columns is simply set to the square root of the number of points rounded to the nearest integer.

The *contour.4* function will produce an approximation of a contour plot even if the input data are randomly scattered over the x-y plane.

In 3D mode, *contour.4* returns new x, y and z values. In 2D mode, the contour (or z values) are returned in the x-error bar column. One could use various UNIX utilities to extract data points along the same contour, place them into separate files, read them back and plot them with unique symbols or pen colors.

Reusing the previous data can save time if you are interested in generating different contour ranges or intervals with the same data set. Only changes to the contour minimum, maximum or number of contour intervals have any effect with the *reuse* option.

Entering a negative number for the number of contour intervals selects logarithmic contour spacing. The minimum and maximum contour values selected must, of course, be positive numbers.

curves.2

Curves.2 is a user function that generates a number of geometric curves.

```
fn curves.2
or
fn curves.2 [range], # [parameters ...]
```

When invoked without arguments, you are presented with a list of names of curves. Select one of the curves and then select values for its parameters.

Since this is a type 2 function, both *x* and *y* are functions of a parametric variable. Most of the curves in *curves.2* are expressed in polar form, with the radius *r* given as a function of polar angle *z*, and it is *z* (in degrees) that is treated as the parametric variable. A range from 0 to 360 is appropriate for most of the sample curves. (For the spirals, of course, you may want to go around many more times.)

You can give the index number of the curve *#* and, optionally, new parameters, on the command line from C-PLOT. You must precede these parameters with a comma to separate them from the position held by the range for the parametric variable (see `fn`). If you only enter a new index number, the previous parameters will be used.

fft.4

Fft.4 does a fast Fourier transform of the current data.

```
fn fft.4
fn fft.4 options
or
fn . .
```

With no arguments, you are prompted for the type of transform and type of data in which you are interested. If you are supplying complex input data or asking for complex output data, the real part is stored in the *x* data column and the imaginary part is stored in the *s* (error-bar) data column. You also can supply options on the command line. Valid options are:

.	:	use same options as last time
+f or −f	:	forward (or reverse) transform
+r or −r	:	real (or complex) input data
+o or −o	:	real (or complex) output
+a or −a	:	take absolute value (or don't)
+s or −s	:	take absolute value squared (or don't)
+m or −m	:	sort and merge the data (or don't)
+i or −i	:	interpolate input data (or don't)
n=*#*	:	set number of points
f=*#*	:	set *from* value for interpolation
t=*#*	:	set *to* value for interpolation

The default options are:

```
+fro  −asmi  n=npts f=first_x t=last_x
```

Here *npts* , *first_x* and *last_x* refer to the number of points and the range of the current data. Interpolation, if used, is by a cubic spline. (See *spline.4*). Naturally, points in *x* must be equally spaced for the transform to make sense.

Transforming more points than the number of in-core points, as set in the $CPLOTHOME/*cplot_config* or $HOME/*.cplot_init* file is extremely slow because of the large number of file accesses required.

When entering options interactively, it is possible to turn on a verbose mode that will show the status of the transform as it works through the data.

fitpar.4

Fitpar.4 makes it easy to plot from a catenated group of save-parameter files created with a type 5 fitting user function. Any of the parameters or the *chi*-squared value may be plotted against any other. In addition, axis labels may be taken from the parameter names.

```
fn fitpar.4
fn fitpar.4 filename x_index y_index [label_flag]
or
fn . .  x_index y_index  [label_flag]
```

Use the `sp`, `sP` or `sf` commands in the fits to save parameter sets to a file. For *fitpar.4* the parameter sets need to be gathered consecutively in a single file. You can accomplish this from the fit by saving parameters with the append option, or from the shell by combining files with the UNIX *cat* utility.

When invoked without arguments, *fitpar.4* will ask for the name of the parameter file, let you view a list of the parameter names, prompt you for the parameter index numbers to use for *x* and *y*, and ask if you want to have the axis labels taken from the parameter names. If the parameters were saved with errors, those values will be put in the error-bar vector. You can assign the value of *chi*-squared to *x* or *y* by entering `-1` as the parameter number.

You can pass both the parameter file name and parameter numbers to the function on the command line from plot. A single dot in place of the file name means to use the same file as before.

For example,

```
    fn fitpar.4 params 0 4
```

will initialize the function with file *params* and return parameter 0 in *x* and parameter 4 in *y*. A subsequent call

```
    fn . . 0 3
```

will return parameter 0 as *x* and parameter 3 as *y*.

You might find it useful to include in your fit functions parameters that aren't used in the fitting but can be used in this function as one of the axes.

The optional fourth command-line argument turns on and off the feature that assigns axis labels from parameter names. A zero turns the feature off, a nonzero number turns it on.

hairs.4

Hairs.4 lets the user position a cursor over the high-resolution display of a plot and read off the cursor position in a variety of units.

```
fn hairs.4
```

When you use *hairs.4* to position cross hairs over each current point on a screen plot, the coordinates of each point will be read off in data units or the position of the cross hairs will be displayed in window or annotation units. The *hairs.4* function is only available on PCs using VGA, EGA or Hercules display adapters in native mode.

The following commands are available in *hairs.4:*

Command	What it does
`c`	step through available cursors
`C`	step through available colors
`d`	toggle among *user*, *window*, *annotate* coordinates
`f`	toggle between fonts
`m`	toggle between *scan* and *roam* motion modes

`q, ˆD, ˆC`
`<return>`
	quit

The keyboard moves the cursor according to the following diagram:

```
     Motion Keys                 Multipliers
   (h j k l y u b n)
                                 Roam   Scan
    \  /                  alt:     20      5
     y u     |          shift:     50     10
   <--h   j  k  l-->  control:    100     50
     b n  |
    /   \
```

Using the *alt*, *shift* or *control* keys with one of the motion keys multiplies the motion as indicated in the table above.

In *scan* mode, the cursor stays on the current data points. In *roam* mode, the cursor can be moved anywhere on the display.

User units display the cursor position in the units of the current data points. Window units display the cursor position in terms of C-PLOT centimeters, measured from the lower-left corner of the display, and are the appropriate numbers to enter for the `wi` command. Annotation units display the cursor position in terms of C-PLOT centimeters measured from the upper-left corner of the plot axis window and are the appropriate numbers to enter for the `zn`, `pn`, `zk` and `pk` commands.

hist.4

Hist.4 is simple user function that takes your data and generates new points to draw a histogram.

```
fn hist.4
or
fn hist.4 bins [min max]
```

The histogram will be a plot of the frequency of occurrence of the *x* values within equally spaced intervals (or bins). You can choose the starting value of the first bin, the end value of the last bin and the number of bins.

The function first sorts your data by the *x* values. You are then presented with the minimum and maximum. You may select different values —usually you will want values rounded to an integer.

By default, the program selects the number of bins to be the difference between the minimum and maximum rounded to an integer (or the difference times increasing powers of 10, until the result is greater than 0). You can use that value or enter another.

You can enter just the number of bins on the command line, in which case the data minimum and maximum values will be used, or you can enter the number of bins and the minimum and maximum values on the command line.

When the function returns, draw the current points with a solid line (symbol `L`) to obtain the histogram. Of course, in the process of generating the histogram your data points are lost from memory.

psych.4

Psych.4 generates a random psychedelic pattern.

```
fn psych.4
fn psych.4 arg
fn psych.4 par1 par2
or
fn . [arg or par1 par2]
```

The patterns *psych.4* generates are determined by two numbers. You can select the numbers or you can let the computer select random values. If you invoke *psych.4* with no arguments, you will be asked if you want it to use random numbers. Whenever you give two numerical arguments, those values are used to create the pattern. A single numerical argument *arg* makes the function use random numbers for subsequent invocations.

When the program is selecting random numbers, values for *par1* are between 0 and 10, and values for *par2* are between 0 and 20.

scans.4

reads data from ASCII scan files.

```
fn scans.4 [options] [scan_numbers]
```

The *scans.4* function reads in files of ASCII data according to a modest set of conventions.

When used with X-ray scattering data, *scans.4* can perform scan averaging, background subtraction, data-normalization and error-bar calculation. However, *scans.4* works well with any kind of data file that follows the conventions described below.

Command line options

`.`	use same options as last time
`-i`	initialize, used to start up function and return
`-f` *`file`*	select scan-file name
`-p`	print scan-file contents
`+d` or `−d`	collect (or don’t) 3 columns of data
`+e` or `−e`	calculate (or don’t) error bars from statistics
`+M` or `−M`	use (or don’t) special MCA data convention
`+n` or `−n`	normalize (or don’t) data points
`+o` or `−o`	sort (or don’t) data points
`+q` or `−q`	don’t (or do) print messages (quiet)
`+r` or `−r`	rerange (or don’t) plot axes for each new data set
`+s` or `−s`	sort and merge (or don’t) data by *x* values
`+v` or `−v`	print (or don’t) each line of scan file (verbose)
`+I` or `−I`	use (or don’t) `#I` intensity normalization
`+S` or `−S`	retrieve scans by scan (or file) number
`x=#`	set column for *x* values
`y=#`	set column for *y* values
`z=#`	set column for *z* values, turn on `+d` flag
`m=#`	set column for monitor normalization, turn on `+n` flag
`t=#`	set column for time normalization, turn on `+n` flag
`x=M`	stuff MCA channel numbers in *x* in 3D mode
`y=M`	stuff MCA channel numbers in *y* in 3D mode

The default settings correspond to the following options in 2D mode

```
   −f data  +eosSn  −drvIqM  x=1  y=−1 m=−2
```

and

```
   −f data  +deosSn  −rvIqM  x=1  y=2  z=−1 m=−2
```

in 3D C-PLOT mode.

Specifying scans

Scans can be retrieved by entering either the scan number (option `+S`, the default) or the file position number (option `−S`).

Scan numbers are determined by the `#S` lines in the file (see below). The file position number is the sequence position of the scan in the file, irrespective of scan number.

When selecting by scan numbers, if there is more than one scan with the same number, the last of them is retrieved. You can specify which instance of a repeated scan number to retrieve by appending a decimal point and an index number to the scan number. For example, selecting scan number `10.3` retrieves the third scan from the start of the file that has scan number 10.

Negative numbers count back from the end of the file and are always considered to be file-position numbers. For example,

```
    fn . −1
```

will always return the last scan in the file.

You can enter multiple scan numbers to select the scans you are interested in. Scan numbers that end with `b` are used as background scans. For example,

```
    fn . 12b 13 14b 15b 16 17b
```

Data in the background scans will be subtracted from the data in the non-background scans that has corresponding *x* values. Choosing a background scan will force the data to be sorted by *x* values.

You can read in a group of consecutive scans with

```
    fn . 3−7 10−14
```

This command would read in scans 3 through 7 and 10 through 14.

File conventions

The scan files contain control lines, data lines and blank lines. Control lines contain a `#` character in the first column followed by a command word. Data lines generally contain a row of numbers. Special data lines containing MCA data begin with an `@` character followed by a row of numbers. These data lines are ignored unless the use MCA data option 0.> is selected.

The control conventions used by *scans.4* are as follows:

`#S` *`N`* − starts a new scan. Here, *`N`* is the user’s numbering scheme and is the number used when retrieving by scan number (`+S`). Most often the scan number is the position of the scan in the file.

`#M` *`N`* − indicates data was taken counting to *`N`* monitor counts.

#T *N* — indicates data was taken counting for *N* seconds.

#N *N* [*M*] — indicates there are *N* columns of data. If *M* is present, it indicates there are *M* sets of data columns on each line. When collecting data from a multi-channel analyzer, for example, the data might be arranged with 16 points per line in the file to make the file easier to scan by eye. In such a case, the control line would be #N 1 16.

#I *N* — is for an optional multiplicative intensity-normalization factor.

#@MCA — indicates the scan contains MCA data. If the +M option is selected, *x* (2D or 3D) or *y* (3D only) values will be calculated automatically. In three-column mode, whether it is *x* or *y* depends on whether the x=M or y=M command line option is selected or on which interactive response was given. Data in the lines starting with @A will be stuffed into the *y* (2D) or *z* (3D) data array.

#@CALIB *a b c* — gives calibration factors for MCA data. The *x* (2D or 3D) or *y* (3D only) values will be calculated using the formula

$$x_i = a + b * i + c * i * i$$

where *i* is the point number, starting from zero. Calibration factors can be changed within the data portion of a scan for subsequent MCA data by the line

```
@CALIB a b c
```

Before each scan is read by *scans.4*, the calibration parameters are initialized to zero.

The following control lines are not commands but are printed out as they are encountered while reading a scan:

#C — is a comment line.

#D — is followed by the date and time the scan was taken.

#L *label1 label2* ... — is the data-column labels, with each label separated from the next by *two* spaces.

For example, a very simple file might have:

```
#S 1
#N 3
#L Temperature  Voltage  Counts
23.4 1.01 30456
23.6 1.015 24000

#S 2 etc.
```

Data columns

When running C-PLOT in 2D mode, the default behavior is to take *x* values from the first column and *y* values from the last column. If in 3D mode, *x* values are taken from the first column, *y* values from the second and *z* values from the last column. If normalizing the data, the default behavior is to use the `#T` or `#M` values. If neither appear, you must enter a column number for the normalization values.

When entering column numbers, a negative number counts backward from the last column. If the column for *x* is zero, the value put in for *x* is just the index number of the point.

Entering options

If you give a dot `.` as the command-line argument or in response to `Scans/options`, the same argument or option string will be used as last time. That is, the string is remembered, not the options chosen interactively using `Change modes?` For instance, if you enter a long sequence of scan numbers and read in the scans, then change something via `Change modes?`, you can simply enter a dot in response to `Scans/options` and recover the previous sequence of scan numbers.

When you do enter a string of flags and scan numbers, the modes set by the flags only apply to the scans that follow the flags, not tp the preceding scans.

The index file

Reading a long ASCII data file takes time. When *scans.4* first opens a file, it scans the whole file and saves a directory of the scans in a binary-format index file. The name of the index file is formed by appending `.I` to the original data-file name.

As long as the index file is more recent than the data file, *scans.4* will use the information in the index file.

Normalization and error bars

Data can be normalized to either monitor counts or time. When normalizing to monitor counts, the error bars will include the uncertainty in the counting statistics of the monitor counts. Otherwise, there is no difference between specifying time or monitor counts.

By default, *scans.4* normalizes data to monitor counts, with the second to last data column used for the monitor count values. Use the 0.> flag to turn off normalization. If a column number is selected using the `m=`*`col`* or `t=`*`col`* arguments, normalization is set to monitor or time mode, respectively, using the column number specified. If the column number in either case is given as zero, the normalization mode and value given by the #M or #T directives for a particular scan in the data file are used. It is an error for normalization mode to be on, for the normalization column to be set to zero and for no normalization directives to be present for a scan.

The normalization modes selected remain in effect for subsequent scans.

The values returned as error bars are those due to counting statistics (the square root of the number of counts). When the counts are derived from the algebraic combination of detector, background and monitor counts, the error bars are calculated using the appropriate " propagation of errors" formalism. See the source code for details.

If the +I option is selected, the counts for each point are multiplied by the value given by the #I control line in the scan header. If the +I option is selected, the counts for each point are multiplied by the value given by the #I control line in the scan header. If the +I option is selected and the scan header doesn't contain a #I control line, the counts are not changed.

Number of points

Earlier version of *scans.4.c* had built-in limits to the number of scans or raw data points that could be handled. Those limits no longer exist.

shell.4

Shell.4 uses the Bourne shell (*/bin/sh*) to run a command that takes the current data as input and produces new data from its output.

```
fn shell.4
fn shell.4 command
or
fn . .
```

With no arguments, you are prompted for a command. Otherwise the command is taken from the command line. If the command is a single dot, the same command as last time is used.

Input

The current points are sent to the subprocess with the *x* and *y* values on one line, with one line per data point. If error-bar and/or line-control modes are on, the corresponding values also are included on the line. If there are no current points, a single newline is written to the command subprocess.

If the subprocess is not interested in the current data, you can save time and system resources by erasing the current data using the gd 15 command.

Output

The output of the subprocess is scanned for *x* and *y* values with the new number of points determined by the number of lines of data that are read. If error-bar and/or line-control modes are in effect, corresponding values also are scanned for. It is not considered an error for either of these values to be missing, but those values that are missing from the input are set to 0.

Any line read from the subprocess that doesn't contain valid data is printed on the screen.

Execution

The Bourne shell (*/bin/sh*) is used to invoke the command. Each time *shell.4* is invoked, the previous instance of the command is killed. All signals for the subprocess running the command are restored to their default values, with the result that a `^\Z` (or your system's quit-signal character) will likely cause the subprocess to do a core dump.

Examples

Use *awk* to multiply the *y* values by 3:

```
fn shell.4 awk '{print $1, 3 * $2}'
```

Use *sort* to rearrange the data,

```
fn shell.4 sort -n
```

This usage sorts the data by *x* values. (Beware, *sort* doesn't recognize scientific notation.) Use *cat* to extract data from an ASCII file,

```
fn shell.4  cat data
```

This usage ignores the current data points.

smooth.4

is a simple boxcar data smoother.

```
fn smooth.4 [bin_size]
```

The function *smooth.4* uses a simple smoothing algorithm that averages some number of adjacent points (given by `bin_size`) to produce a new point. The `bin_size` parameter is rounded up to an odd number.

The input data should be presorted. The number of points used in the average decreases at the end points.

sort.4

Sort.4 sorts the current data by *x*, *y* or *s*, the error-bar column (which doesn't have to contain error-bar values). Data can be sorted by increasing (forward) values or decreasing (reverse) values.

```
fn sort.4
or
fn sort.4 [x|y|z|s] [f|r]
```

With no command-line options, you are prompted for the data column to sort by, and whether to sort in the forward or reverse sense.

The command-line arguments can be entered together as one word, or separately. Only the last valid command letters are considered. The default each time is to sort in the forward sense by *x*.

spline.4

Spline.4 creates data points at evenly spaced intervals using a cubic spline algorithm to interpolate new points from the old. The interpolated points lie on a cubic polynomial between each pair of original points, and each polynomial section is joined continuously to the next with continuous first and second derivatives.

```
fn spline.4
fn spline.4 [+s|−s] [[n=]npts] [f=from] [t=to]
or
fn . .
```

The default `from` and `to` values are the first and last values of *x* in the original data. If you don't enter arguments, you will be prompted for the number of points and the range to interpolate over.

The `+s` and `−s` options turn the automatic sorting and merging of data on and off. Data must be in increasing order, by *x*, for the spline to succeed. Also, *x* values must be distinct. Turning the sort off saves time for splining presorted data. The default is no sort.

Spline.4 turns off error-bar mode and error-bar values are ignored in the sorting.

Appendix D Demo Files

The C-PLOT distribution includes several standard demonstration files showing the program's capabilities. This section includes samples of the graphics they produce, along with the contents of the command files used to make the graphics. Users may find it helpful to review these files as an aid to understanding the operation of C-PLOT's command file facility.

The sample plots here were made using C-PLOT's PostScript filter.

Each of these command files uses graphics filter `z` commands to draw the plots. You can produce the graphics on your own display device or printer by initializing the appropriate graphics filter using the `zi` command. To display output on a pen plotter, first issue the `in` command to initialize the plotter, then type `ch p`, which instructs C-PLOT to translate the `z` commands to pen-plotter `p` commands.

Each command file begins with `zeq9999w` followed by a blank line. If you haven't already initialized a filter with the *zi* command, the blank line will cause the default filter to be used. The `e` erases the current plot. The `q` sets quiet mode. (*Quiet mode* is necessary with certain display devices, such as a video terminal on a serial interface, to prevent the plot program's text stream from mingling with the filter program's graphics stream. When plots are directed to the pen plotter, quiet mode has no effect.) The `9999` resets the colors in the graphics filter. The `w` causes the filter to wait for more plotting instructions before restoring text mode or printing the page.

Each command file also issues a reset command, `re`, at the beginning to put the plot program into a standard state. An additional command file, *doall*, will run all the demos consecutively.

Plane curves

The command file *curves* displays 12 curves, each in a small window. The data for the curves is supplied by the user function *curves.2*, which generates plane curves using formulas taken from *Standard Mathematical Tables* (Cleveland, CRC Press, 1976).

The command file *curves* calls another command file, *curves.p1*, which is nested and is called with four arguments. The first two values specify the location of the small windows. The next two arguments, each of which is enclosed in quotes, set the parameters for the function *curves.2*. The third argument gives the range of the type 2 user function parametric variable, while the fourth selects and parameterizes the particular curve.

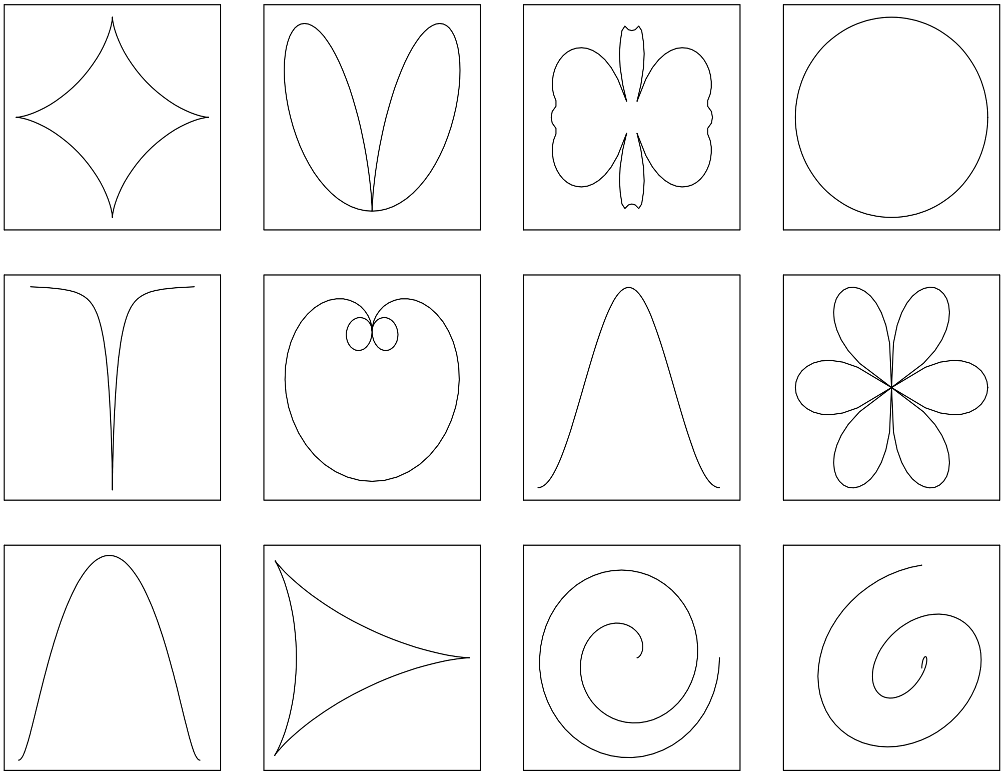
curves

```
#  @(#)curves     4.3  30 Jun 1993 CSS
#  Certified Scientific Software’s C-PLOT

# Erase, turn on quiet mode, reset filter,
# ...  keep in graphics mode.
zeq9999w

re
# Choose symbol
sy L
# Turn off numbering
ty +16 +16 .
# Turn off tick marks
ty +512 +512 .
# Initialize function
fn curves.2  0 360 120 , 1 1
# Make little plots
do curves.p1   1  13     "0 360 120"   "1 1"
do curves.p1   7  13     "0 180 120"   "2 1 0"
do curves.p1  13  13     "0 360 120"   "3 1 .5"
do curves.p1  19  13     "0 360 120"   "4 1"
do curves.p1   1   7   "-85  85 120"   "5 1 90"
do curves.p1   7   7  "-360 360 120"   "6 1 -90"
do curves.p1  13   7     "0 360 120"   "7 1"
do curves.p1  19   7     "0 360 120"  "15 1 3 0"
do curves.p1   1   1     "0 360 120"   "9 1 .5"
do curves.p1   7   1     "0 360 120"  "10 1"
do curves.p1  13   1     "0 720 120"  "11 1 4"
do curves.p1  19   1     "0 720 120"  "12 1 0"
# Synchronized close
zs
```

curves.p1

```
#  @(#)curves.p1  4.1  19 Sep 1992 CSS
#  Certified Scientific Software’s C-PLOT

# Set window
wi $1 $2 5 5
# Get points
fn . $3 , $4
# Set new axis
np
# Draw the points and axis
zap
```

Here are the 12 tiny plots that *do.curves* produces:



Psychedelic patterns

The file *psych* produces randomized patterns in 12 small windows on the page, using command files very much the same as those in the previous example.

psych

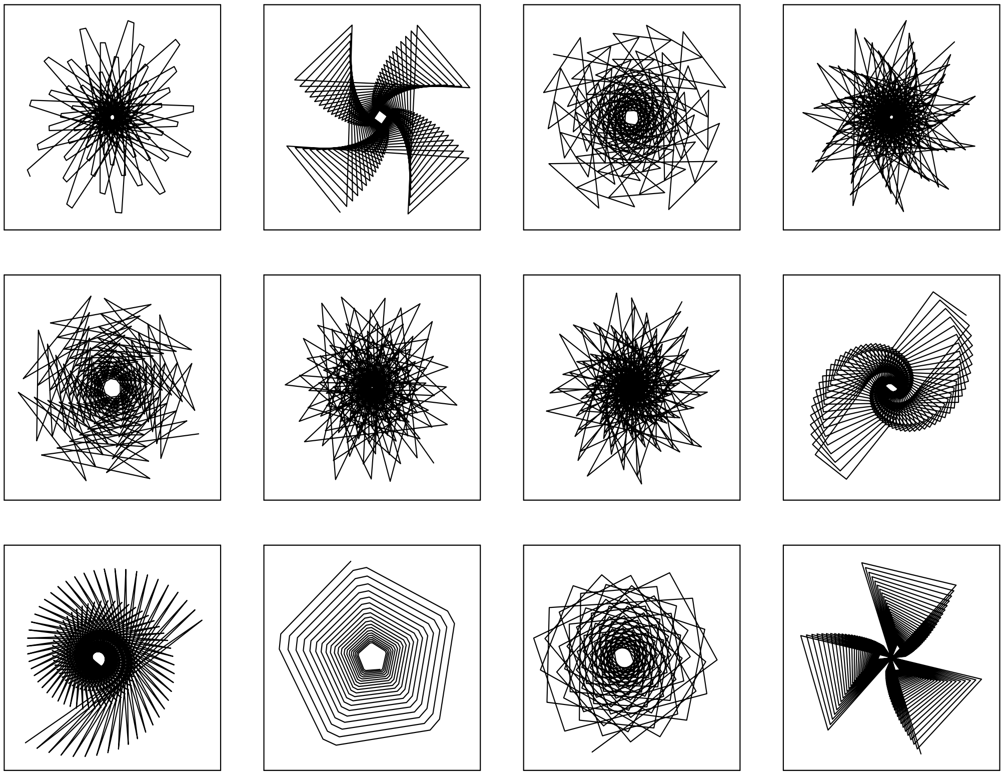
```
#  @(#)psych      4.2  03 May 1993 CSS
#  Certified Scientific Software’s C-PLOT

# Erase, turn on quiet mode, reset filter,
# ...  keep in graphics mode.
zeq9999w

re
# Choose symbol
sy L
# Turn off numbering
ty +16 +16 .
# Turn off tick marks
ty +512 +512 .
# Initialize function
fn psych.4 1
# Make little plots
```

```
do psych.p1   1  13
do psych.p1   7  13
do psych.p1  13  13
do psych.p1  19  13
do psych.p1   1   7
do psych.p1   7   7
do psych.p1  13   7
do psych.p1  19   7
do psych.p1   1   1
do psych.p1   7   1
do psych.p1  13   1
do psych.p1  19   1
# Synchronized close
zs
```

psych.p1

```
#  @(#)psych.p1   4.1  19 Sep 1992 CSS
#  Certified Scientific Software’s C-PLOT

# Set window
wi $1 $2 5 5
# Get points
fn .
# Set new axis
np
# Draw the points and axis
zap
```

Here are the psychedelic patterns that result:



Fourier transforms

The command file *fft* uses the user function *calc.4* to generate two different data sets. The user function *fft.4* is invoked to generate the Fourier transform of each of those sets. Note that the 1,024 data points generated for each curve go far past the 128 points plotted. The wide range in real space gives high resolution to the frequency-space Fourier transform.

The first data set generates data that is the sum of three cosine curves of differing amplitudes and frequencies within a Gaussian envelope:

$$y = [2\,cos\,(2x) + cos\,(x/2) + 3\,cos\,(x)]\ e^{-(x/64)^2}\ .$$

The Fourier transform of this data shows three broadened peaks at the appropriate frequencies with amplitudes in the correct ratios.

The second data set is simply

$$y\ =\ \frac{sin\,(x)}{x}\ .$$

Its Fourier transform is a boxcar function.

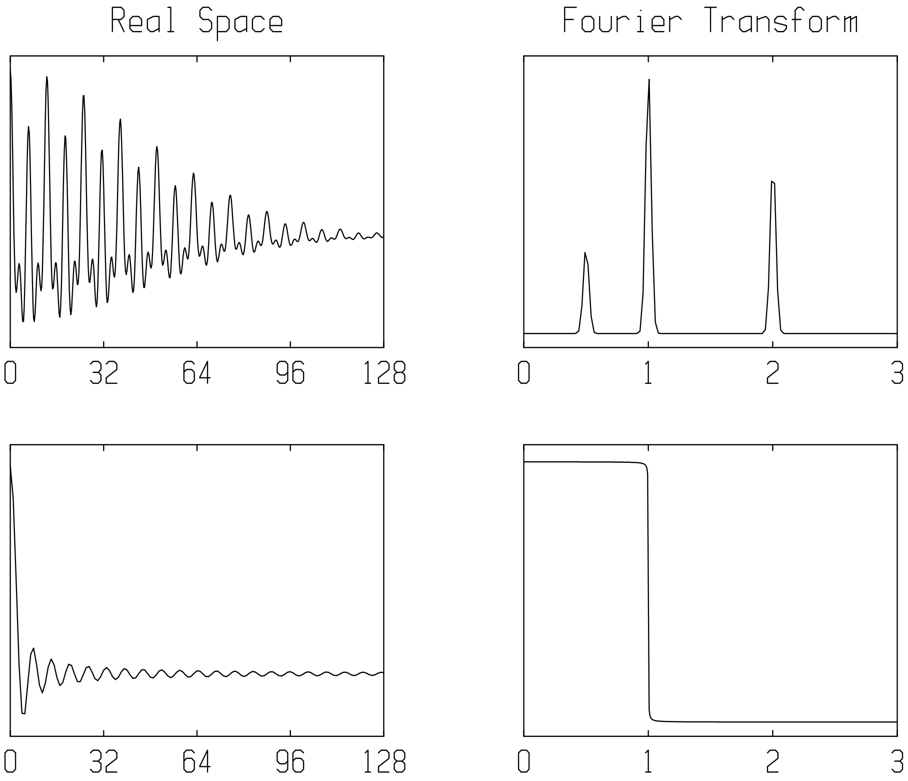
The range-options command for the x axis, `ro x`, is used four times within the command file to set the user-defined tick spacing option. Six lines of input follow each time, representing values for the minimum and the maximum, a choice not to autorange the *y* axis, a choice of exact ranges, a choice of user-defined tick spacing and values for the number of major tick intervals and intermediate tick marks.

fft

```
#  @(#)fft  4.2  03 May 1993 CSS
#  Certified Scientific Software's C-PLOT

# Erase, turn on quiet mode, reset filter,
# ...  keep in graphics mode.
zeq9999w

re
# Choose symbol
sy L
# Don't draw y-axis number or tick marks
ty 0 528 0

# Set Title
tx t Real Space

# Calculate some "real space" data that consists
# of three sinusoids broadened by a Gaussian.
fn calc.4 n=1024; x=i/4; \
 y=(2*cos(x*2)+cos(x/2)+3*cos(x))*exp(-x*x/(64*64))
# Set x-axis range and user-defined tick numbering
ro x
0
128
n
y
y
4 0
# Autorange y-axis
np y
# Set first window
wi 3 11 8 6
# Draw plot
zapt
# Set Title
tx t Fourier Transform
# Now do FFT
f2 fft.4 n=1024
# Set x-axis range and user-defined tick numbering
ro x
0
3
n
y
y
3 0
```

```
# Autorange y-axis
np y
# Set second window
wi 14 11 8 6
# Draw plot
zapt

# Calculate another set of "real space" data
fn . n=1024; x=i+.00001; y = sin(x)/(x)
ro x
0
128
n
y
y
4 0
np y
wi 3 3 8 6
zap
# Now do FFT
f2 . n=1024
ro x
0
3
n
y
y
3 0
np y
wi 14 3 8 6
zap
# Synchronized close
zs
```

Here are the real-space plots and their Fourier transforms.


Real Space

Fourier Transform

Contour plots

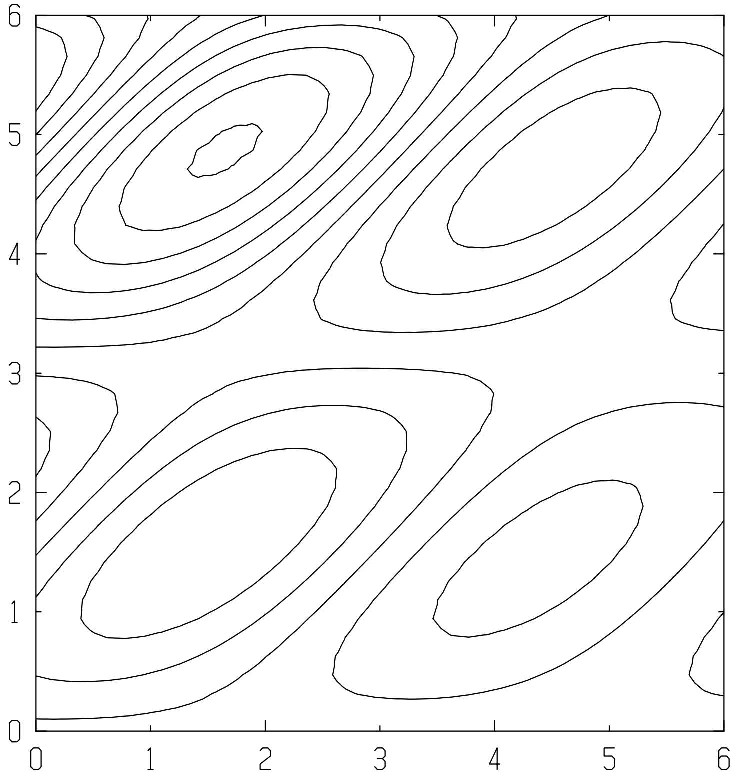
The simple command file *contour* first generates data triples using the user function *calc.4*. The calls to the user function are done twice in the example just for readability. The command file then invokes the *contour.4* user function to create a data set that forms the contour plot when drawn.

contour

```
#  @(#)contour    4.2  03 May 1993 CSS
#  Certified Scientific Software’s C-PLOT

re
# Calculate x and y grid
fn calc.4 n=1600; once G=40; once W=2*pi/G; \
 x=int(i/G)*W; y=(i%G)*W;

# Calculate z values
fn . z=sin(y)*cos(x-y)*exp((-x*x+y*y)*(W*W))
# Calculate contour
f2 contour.4 +sgv 15 zmin=-2 zmax=2.5
# Use a square window
wi 1
```

```
# Select tighter ranges
ra 0 6 0 6
# Use exact ranges
ty 2 2 0
sy L
# Plot it
zeq9999w

zzs
```

Here is the resulting contour plot:



Feature demo

The command file *demo* demonstrates several of C-PLOT's plot and text formatting features.

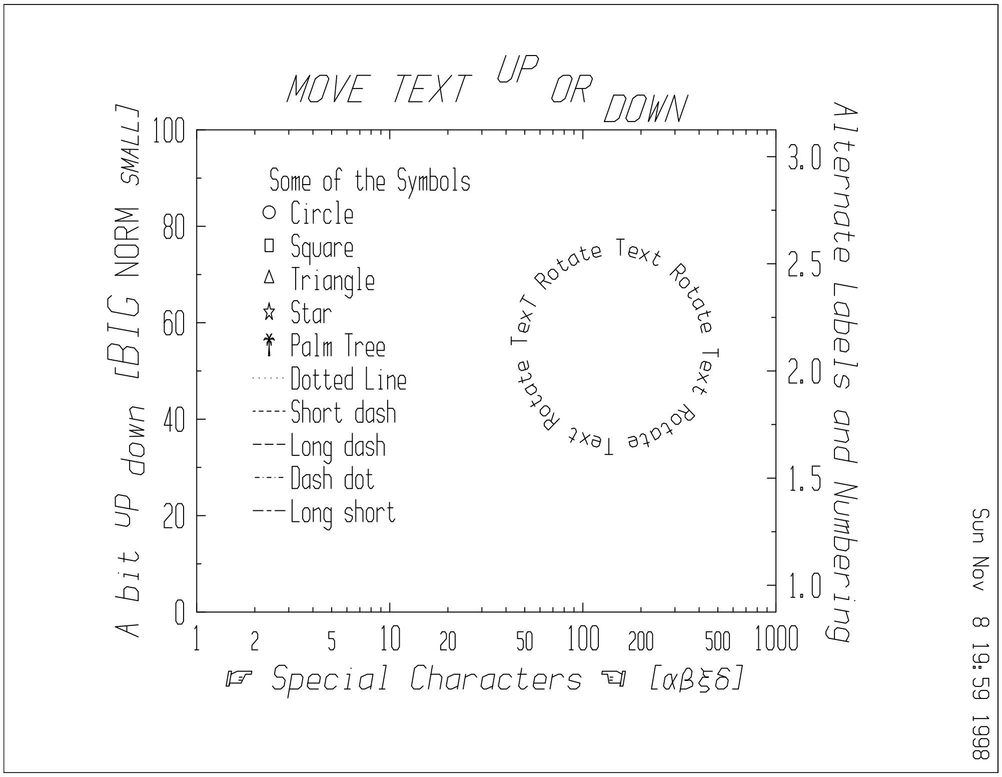
demo

```
#  @(#)demo 4.2  03 May 1993 CSS
#  Certified Scientific Software's C-PLOT

# Erase, turn on quiet mode, reset filter,
# ...  keep in graphics mode.
zqe9999w

re
```

```
sy L
ra 1 1000 0 100
# Select types for exact ranges, log x axis
# and to draw box around plotting area
ty 10 2 32
wi 5 4 15 12
# Select title and labels
tx
MOVE TEXT \uUP\d OR \dDOWN\u
\(rh Special Characters \(lh
\(*a\(*b\(*c\(*d
A bit \v’-3’up\v’5’ down\v‘-2‘
\lBIG\s \T’0’NORM\T’15’ \sSMALL\l
# Select key
gk
z Some of the Symbols
0 Circle
1 Square
2 Triangle
12 Star
23 Palm Tree
A Dotted Line
B Short dash
C Long dash
D Dash dot
E Long short
ˆD
# Characters sizes
cs
7 2 20
6 2 10
6 3 0
5 2 0
5.5 3 0
4 2 0
# Select type to not draw right y axis
ty . . +256
zz

ra y 1 3
tx
\
\
\
Alternate Labels and Numbering
\

# Select types to draw only right axis
ty . -3 -256
ty . . +640

zaldk

cs k 4 2
```

```
zn 9 4
\R’-45’Ro\R’-30’ta\R’-15’te\R’000’ T\R’+15’ex\
\R’+30’t \R’+45’Ro\R’+60’ta\R’+75’te\R’+90’ T\
\R’105’ex\R’120’t \R’135’Ro\R’150’ta\R’165’te\
\R’180’ T\R’195’ex\R’210’t \R’225’Ro\R’240’ta\
\R’255’te\R’270’ T\R’285’ex\R’300’T
^D

zs
```

Here is the resulting graphic:



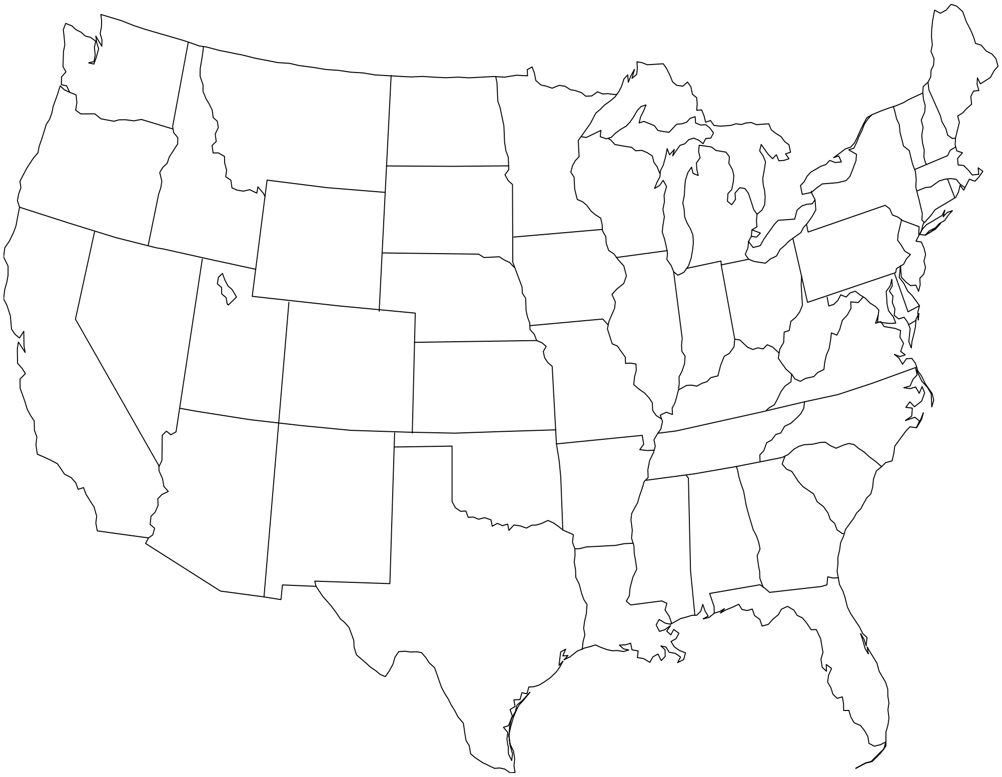
USA map

The data points in the file *usa.data* include line-control information that tells C-PLOT whether to continue the line from the previous point or to start a new line at the current point. The command `lc 1` turns the line-control mode on so that line-control information will be read in with the `gd` command and used when plotting the points.

usa_map

```
#  @(#)usa_map    4.1  19 Sep 1992 CSS
#  Certified Scientific Software’s C-PLOT

zeq9999w

re
ty 2 2 0
wi 9
sy L
lc 1
gd 2 usa_map.dat
zps
```

Here is the map:



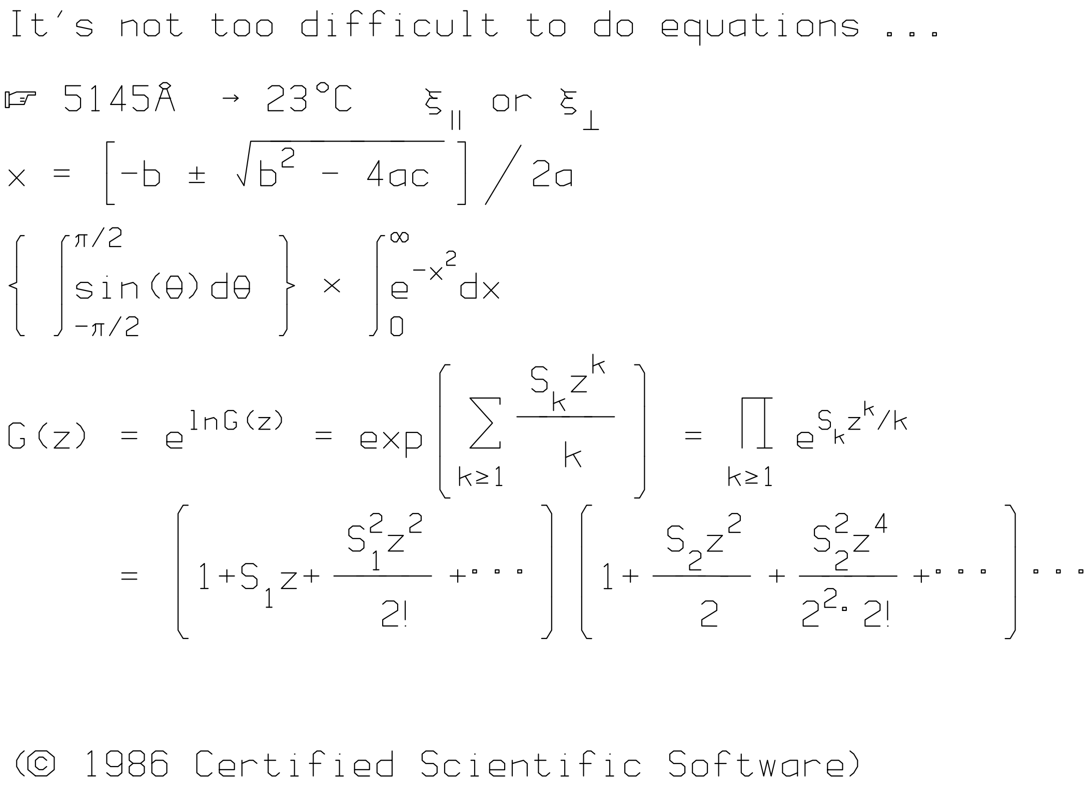
Mathematical text

The file *eqn* demonstrates the special text-formatting sequences available with C-PLOT. These sequences resemble the standard *nroff*/*troff* sequences, for the most part.

eqns

```
#  @(#)eqns 4.2  03 May 1993 CSS
#  Certified Scientific Software’s C-PLOT

zqe9999w

re
wi 9
cs k 5.5 1.6
zns 1 1
It’s not too difficult to do equations ...

\(rh 5145\(an  \(-> 23\(deC\
\   \(*c\d\(pa\u or \(*c\d\(pe\u

x = \u\(lc\b\d\d\(lf\u\
-b \(+- \l\l\(sr\(rn\(rn\(rn\(rn\
\W’-\(sr\(rn\(rn\(rn\(rn’\s\s\ \|b\u\s2\l\d - 4ac\
\ \u\(rc\b\d\d\(rf\u\d\l\l\l/\s\s\s\u2a


\u\u\(lt\b\d\d\(lk\b\d\d\(lb\u\u\
\ \u\u\(lt\s\u\(*p/2\d\W’-\(*p/2’\l\b\d\d\
\(bv\b\d\d\(rb\
\s-\(*p/2\W’--\(*p/2’\l\u\usin(\(*h)d\(*h\
\ \u\u\(rt\b\d\d\(rk\b\d\d\(rb\u\u\ \(mu\
\ \u\u\(lt\s\u\(if\d\W’-\(if’\l\b\d\d\(bv\b\d\d\(rb\
\s0\b\l\u\ue\u\s-x\u\s2\l\d\l\ddx



G(z) = e\u\sln\|G(z)\l\d = \
exp\l\u\u\(lt\b\d\d\(bv\b\d\d\(lb\u\u\s\
\|\|\S’50’\(*S\b\S’-50’\d\d\s\b\|\|\|\|k\(>=1 \
\l\u\u\u\u\|\uS\d\sk\l\uz\u\sk\l\d\d\b\b\b\b\
\(ru\(ru\(ru\(ru\b\b\d\d\dk\u\
\  \l\u\u\(rt\b\d\d\(bv\b\d\d\(rb\u\u\s\
\ = \|\|\S’50’\(*P\b\S’-50’\d\d\s\b\|\|\|k\(>=1\
\ \l\u\ue\u\sS\d\sk\l\uz\u\sk\l\d/k



\     = \l\u\u\(lt\b\d\d\(bv\b\d\d\(lb\u\u\s\
1+S\d\s1\l\uz+ \u\uS\d\s1\b\l\u\u\s2\l\dz\u\s2\l\
\d\d\b\b\b\b\(ru\(ru\(ru\(ru\b\b\d\d\d2!\u\u\
\ +\v’-5’...\v’5’\
\l\u\u\(rt\b\d\d\(bv\b\d\d\(rb\u\u\s\
\l\u\u\(lt\b\d\d\(bv\b\d\d\(lb\u\u\s\
1+ \u\uS\d\s2\l\uz\u\s2\l\d\d\
\b\b\b\b\(ru\(ru\(ru\(ru\b\b\d\d\d2\u\u\
\  + \u\uS\d\s2\b\l\u\u\s2\l\dz\u\s4\l\d\d\
\b\b\b\b\(ru\(ru\(ru\(ru\b\b\b\b\
\d\d\d2\u\s2\l\d\v’-5’.\v’5’2!\u\u\
\|+\v’-5’...\v’5’\
\l\u\u\(rt\b\d\d\(bv\b\d\d\(rb\u\u\s\
\v’-5’...\v’5’\



(\(co 1986 Certified Scientific Software)
ˆD
```

Here are the equations produced by the code contained in this command file:



It's not too difficult to do equations ...

☞ 5145Å → 23°C $\xi_{\parallel}$ or $\xi_{\perp}$

$$x = \left[-b \pm \sqrt{b^2 - 4ac}\right] \Big/ 2a$$

$$\left\{ \int_{-\pi/2}^{\pi/2} \sin(\theta)\,d\theta \right\} \times \int_0^{\infty} e^{-x^2} dx$$

$$G(z) = e^{\ln G(z)} = \exp\left(\sum_{k\geq 1} \frac{S_k z^k}{k} \right) = \prod_{k\geq 1} e^{S_k z^k/k}$$

$$= \left(1+S_1 z+ \frac{S_1^2 z^2}{2!} +\cdots \right) \left(1+ \frac{S_2 z^2}{2} + \frac{S_2^2 z^4}{2^2 \cdot 2!} +\cdots \right) \cdots$$

(© 1986 Certified Scientific Software)

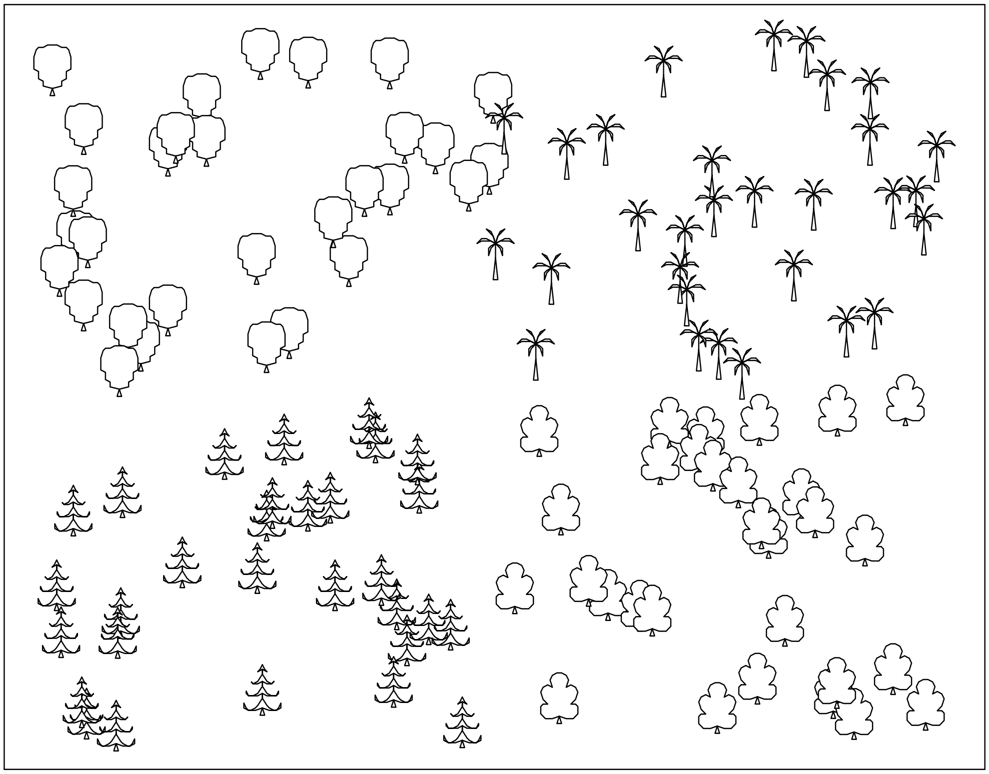
Forests

The command file *trees* uses the random number function in *calc.4* to simulate one tropical palm grove, two temperate forests with deciduous foliage and one conifer-populated forest.

trees

```
#  @(#)trees      4.1  19 Sep 1992 CSS
#  Certified Scientific Software’s C-PLOT

zeq9999w

re
# Select no ticks and no numbers
ty 528 528 0
cs s 10
ra 0 10 0 10
za
# Generate 30 random points between 0,0 and 5,5
fn calc.4 n=30; x=rand(5); y=rand(5);
sy 24
zp
fn . x=rand(5)+5; y=rand(5);
sy 25
zp
fn . x=rand(5); y=rand(5)+5;
sy 26
zp
fn . x=rand(5)+5; y=rand(5)+5;
sy 23
zp
zs
```

Here is the resulting graphic:



Pen numbers

C-PLOT's conventions for numbering pens is described in the next section. The conventions allow command files to produce consistent plots with any conforming C-PLOT graphics filters.

The command file *pens* uses a range of pen numbers to change filled-symbol coloring and line widths.

pens

```
#  @(#)pens 4.2  03 May 1993 CSS
#  Certified Scientific Software's C-PLOT

zeq9999w

re
zw
wi 9
sy 0
ra 0 9 .2 10
ty 2 2 0
cs k 8
ft 2
```

```
zn 0 10u
\C\T’20’Pen Numbering Demonstration
ˆD
zn 0 9.1u
\s "White" Filled Symbols   \h’6’Line Widths\
\  \u"Black" Filled Symbols
\s\h’435’\u+ Outline Widths
ˆD

do pens.p1 1000 0 8u
do pens.p1 1001 0 7u
do pens.p1 1002 0 6u
do pens.p1 1003 0 5u
do pens.p1 1004 0 4u
do pens.p1 1005 0 3u
do pens.p1 1006 0 2u
do pens.p1 1007 0 1u
do pens.p1 1008 0 0u

do pens.p1 1009 2 8u
do pens.p1 1010 2 7u
do pens.p1 1011 2 6u
do pens.p1 1012 2 5u
do pens.p1 1013 2 4u
do pens.p1 1014 2 3u
do pens.p1 1015 2 2u
do pens.p1 1016 2 1u
do pens.p1 1017 2 0u

do pens.p2 4000 4 8u
do pens.p2 4005 4 7u
do pens.p2 4010 4 6u
do pens.p2 4020 4 5u
do pens.p2 4030 4 4u
do pens.p2 4040 4 3u
do pens.p2 4060 4 2u
do pens.p2 4080 4 1u
do pens.p2 4100 4 0u

z9999

do pens.p3 2001+5000 6 8u
do pens.p3 2001+5010 6 7u
do pens.p3 2001+5040 6 6u
do pens.p3 2005+5000 6 5u
do pens.p3 2005+5010 6 4u
do pens.p3 2005+5040 6 3u
do pens.p3 2002+5000 6 2u
do pens.p3 2002+5010 6 1u
do pens.p3 2002+5040 6 0u

zs
```

pens.p1

```
#  @(#)pens.p1    4.1  19 Sep 1992 CSS
#  Certified Scientific Software’s C-PLOT

z$1n $2 $3
\v’3’\S’+50’\[00\S’-50’\v’-3’$1
^D
```

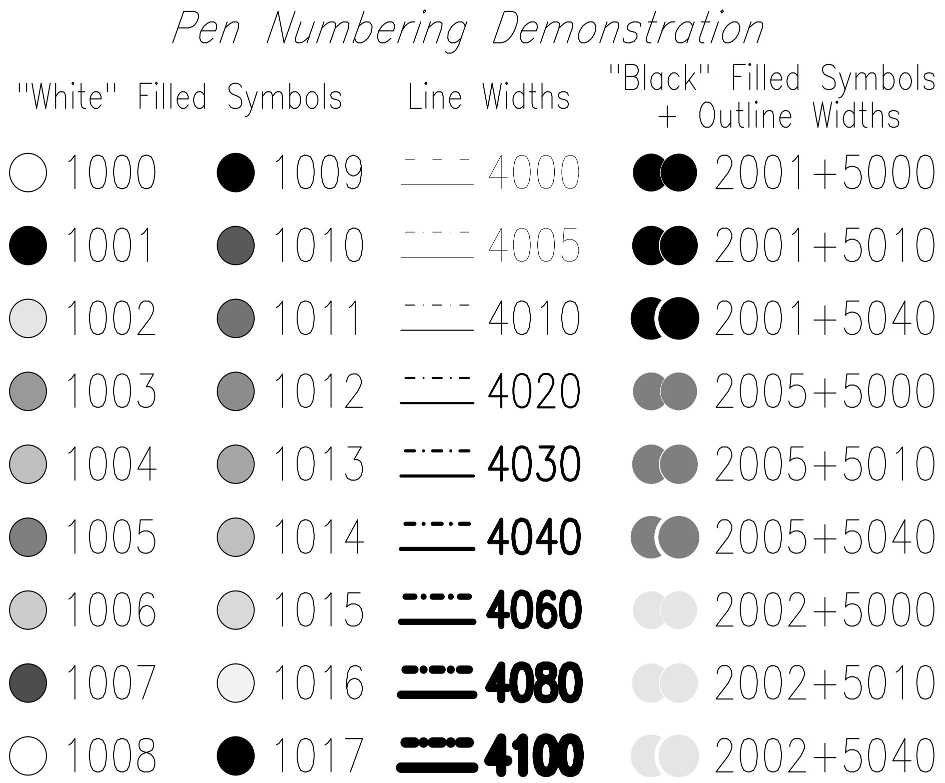
pens.p2

```
#  @(#)pens.p2    4.1  19 Sep 1992 CSS
#  Certified Scientific Software’s C-PLOT

z$1n $2 $3
\h’-6’\v’-3’\*D\v’+3’\h’6’$1
\r\h’-6’\v’+3’\*L\v’-3’\h’6’
^D
```

pens.p3

```
#  @(#)pens.p3    4.1  19 Sep 1992 CSS
#  Certified Scientific Software’s C-PLOT

z$1n $2 $3
\v’3’\S’+50’\[04\h’-9’\[04\S’-50’\v’-3’$1
^D
```

The appearance of the plot drawn with this command file will vary depending on the graphics filter being used. Not all pen numbers are functional on all graphics filters. The following picture is produced by running the command file with the PostScript graphics filter:


Pen Numbering Demonstration
"White" Filled Symbols
Line Widths
"Black" Filled Symbols
+ Outline Widths
1000
1009
4000
2001+5000
1001
1010
4005
2001+5010
1002
1011
4010
2001+5040
1003
1012
4020
2005+5000
1004
1013
4030
2005+5010
1005
1014
4040
2005+5040
1006
1015
4060
2002+5000
1007
1016
4080
2002+5010
1008
1017
4100
2002+5040

Appendix E Writing User Functions

Rules for writing type 1 to 4 C-PLOT user functions are explained in this appendix. Rules for writing type 5 user functions, the fits, are discussed in the second half of Chapter 12.

For type 1 user functions, you provide a routine to produce values for the dependent variable *y* as a function of the independent variable *x*. The values of *x* passed to your routine are determined by the ranges you enter with the `fn` command. The values for *x* error bars, *y* error bars and *z* are set to 0. Type 1 users functions are not useful in 3D mode. In type 2 user functions, you provide routines to produce values for all four C-PLOT variables, *x*, *y*, *r* and *s* in 2D mode and *x*, *y*, *z* and *s* in 3D mode, as a function of the parametric variable whose range you also supply with the `fn` command. For type 3 user functions, no range information is used. Instead, you provide routines to calculate new values for the four variables based on their current values.

For the first three types of user functions, you provide routines to calculate values for a single data point coordinate at a time. These routines are repeatedly called by the overhead module routines. With type 4 user functions, the overhead module makes one call to the routine you provide. You supply the code necessary to create or modify the entire data set.

With each of these four types of user functions, you can set many of the display options for the plot, including the title, labels, symbol, key and axis ranges. Each of these user function requires you to provide a routine named `setup()` that will be called once. For type 1 to 3 user functions, this is where the display options would be set.

Compiling user functions

Normally you run the shell script $CPLOTHOME/*bin*/*makefunc* from the function directory to compile your user functions. That file usually invokes the C compiler to compile your module and link it with the appropriate overhead modules. If your user function contains the string `cplot_compile:` followed by commands to compile your function, those commands are used instead of the default commands from *makefunc*. The commands can refer to the *make* utility or invoke the C compiler directly. Possible ways of including the information in a function source file are:

```
/*
*    cplot_compile:  make my_func.5
*/
```

or

```
#if 0
    cplot_compile:  make my_func.5
#endif
```

Accessing and modifying the data structure passed to and from the plot program is simplified by a set of macros that is brought into your program with the line of code,

```
#include <p_funct.h>
```

This *include* file also includes another, *p_plot.h*, that describes the above-mentioned data structure. Both include files are found in the directory $CPLOTHOME/*include*.

Text and labels

The following macros set the text that would be entered interactively using the `tx` command.

Name	What it Does
`set_title(`*s*`)`	Assigns string *s* to title
`set_xlabel(`*s*`)`	Assigns string *s* to x-axis label
`set_xunits(`*s*`)`	Assigns string *s* to x-axis units
`set_ylabel(`*s*`)`	Assigns string *s* to y-axis label
`set_yunits(`*s*`)`	Assigns string *s* to y-axis units
`set_zlabel(`*s*`)`	Assigns string *s* to z-axis label
`set_zunits(`*s*`)`	Assigns string *s* to z-axis units

For example,

```
set_title("Plot Of Very Interesting Results");
set_xlabel("Radius");
set_xunits("cm");
set_ylabel("Intensity");
set_yunits("");                /* No y units */
```

Strings not explicitly assigned retain their previous contents. The manifest constant TEXT_LEN is defined in the include files and contains the maximum number of characters allowed in any text strings.

Error-bar, line-control and orientation modes

Error-bar, line-control and orientation modes can be turned on or off, or you can determine if they are already on or off, with these macros. Within the plot program, the commands eb , lc and tu control these modes.

Name	What it Does
xb_on()	Turn *x* error-bar mode on
xb_off()	Turn *x* error-bar mode off
is_xb()	Value is nonzero if *x* error-bar mode is on
yb_on()	Turn *y* error-bar mode on
yb_off()	Turn *y* error-bar mode off
is_yb()	Value is nonzero if *y* error-bar mode is on
z_on()	Turn *z* error-bar mode on
z_off()	Turn *z* error-bar mode off
is_zb()	Value is nonzero if *z* error-bar mode is on
lc_on()	Turn line-control mode on
lc_off()	Turn line-control mode off
is_lc()	Value is nonzero if line-control mode is on
tu_on()	Turn portrait mode on
tu_off()	Turn portrait mode off (plot in landscape mode)
is_tu()	Value is nonzero if portrait mode is on

These macros set the current plotting symbol and create or modify the symbol and text in the key. The commands sy and gk perform the same function within the plot program.

Name	What it Does
set_sym(*sym*)	Assign code in *sym* to plotting symbol
set_key(*i*, *sym*, *s*)	Assign *i*th line of the key the symbol *sym* and the string *s*

The symbol is coded in the low-order 16 bits of an (unsigned) integer. In the following description, these bits are numbered from 0 to 15, with bit 0 being the least significant (ones) bit.

If bit 7 is set, the code is for a line symbol and bits 0-6 contain the ASCII value of the corresponding upper-case letter.

```
set_sym(0x80|’L’);      /* Solid line */
set_sym(0x80|’A’);      /* Dotted line */
```

If some of bits 0-6 are set but none of bits 8-14, the code contained in bits 0-6 is one greater than the corresponding code for one of the special symbols.

```
set_sym(1+0);   /* Code 0, an open circle */
set_sym(1+23);    /* Code 23, a palm tree */
```

If some of bits 8-14 are set but none of bits 0-6, the code contained in bits 8-14 (shifted down 8 bits) is the ASCII value of the character to be used as the symbol.

```
set_sym(’+’<<8);        /* A plus sign */
set_sym(’X’<<8);        /* An upper-case x */
```

If both some of bits 0-6 and some of bits 8-14 are set, the two ASCII characters represented are the two-letter code for one of the special characters used by C-PLOT.

```
set_sym(’*’|’a’<<8);    /* The \(*a, alpha character */
set_sym(’d’|’d’<<8);    /* The \(dd, double dagger */
```

Finally, when assigned to a symbol used in the key, if bit 15 is set, the program pauses when drawing the key on a pen plotter to allow the user to change pens. (See the c option in gk .)

The set_key() macro can be used to assign each element of the key. The second argument contains the coded symbol using the conventions described above. The second argument should be set to zero to mark the last element of the key.

```
set_key(0, 1+0, "Data");        /* 0, open circle */
set_key(1, 0x80|’L’, "Fit";     /* ’L’, solid line */
set_key(2, 0, "");              /* mark end with sym=0 */
```

The manifest constant KEY_LEN is defined to be the maximum length of the key labels (including the terminating null byte). The constant NUMKEYS is the number of key entries allowed. If using all NUMKEYS entries, it is not necessary to mark the last entry with a null symbol.

Plot type and logarithmic axes

The plot-type parameters control many of the features of the plot and are set within the plot program using the ty command.

Name	What it Does
set_xtype(*t*)	Assigns *t* to the *x*-axis plot type
set_ytype(*t*)	Assigns *t* to the *y*-axis plot type
set_ztype(*t*)	Assigns *t* to the *z*-axis plot type
set_gtype(*t*)	Assigns *t* to the overall plot type
xlog_on()	Make the *x*-axis logarithmic
xlog_off()	Make the *x*-axis linear
is_xlog()	Value is nonzero if *x*-axis is logarithmic
ylog_on()	Make the *y*-axis logarithmic
ylog_off()	Make the *y*-axis linear
is_ylog()	Value is nonzero if *y*-axis is logarithmic
zlog_on()	Make the *z*-axis logarithmic
zlog_off()	Make the *z*-axis linear
is_zlog()	Value is nonzero if *z*-axis is logarithmic

The values to assign to the plot types in the above macros are the same values used within the plot program. The macros to set or unset logarithmic axes simply add the appropriate values to the plot type.

Axis ranges

The minimum and maximum values of each axis can be set from within the user function just as with the *range axis* command, `ra`. You also can have either or both axes auto-ranged by the plot program in a manner similar to the *new points* command, `np`.

Name	What it Does
`new_points()`	Axis ranges set from current data
`new_xpoints()`	x-axis ranges set from current data
`new_ypoints()`	y-axis ranges set from current data
`new_zpoints()`	z-axis ranges set from current data
`set_xmin(`*x*`)`	Set x-axis minimum to *x*
`set_xmax(`*x*`)`	Set x-axis maximum to *x*
`set_ymin(`*y*`)`	Set y-axis minimum to *y*
`set_ymax(`*y*`)`	Set y-axis maximum to *y*
`set_zmin(`*z*`)`	Set z-axis minimum to *z*
`set_zmax(`*z*`)`	Set z-axis maximum to *z*
`get_xmin()`	Value is x-axis minimum
`get_xmax()`	Value is x-axis maximum
`get_ymin()`	Value is y-axis minimum
`get_ymax()`	Value is y-axis maximum
`get_zmin()`	Value is z-axis minimum
`get_zmax()`	Value is z-axis maximum

The automatic ranging of the axes using the first three macros won't take effect until the function returns to the plot program, so the macros to retrieve the axis extremes will return the values contained on entry to the user function or set in the current invocation of the user function.

Plot and page window coordinates

The coordinates of the plot window are available in both internal and data units. You might use the internal units to obtain the aspect ratio of the plot window, useful, for example, if you are writing a user function to generate data points to draw arrows.

The macros only provide the page-size coordinates in internal units. The corresponding data-unit values can be readily obtained using ratios with the values for the plot window coordinates.

None of the values for these coordinates can be changed within a user function. The values for the plot window coordinates are changed with the `wi` command within the plot program. The page-window coordinates may change only if other than the default HP-GL plotter is initialized with the `in` command.

Name	What it Does
`get_dx0()`	Plot-window *x* minimum in data units
`get_dy0()`	Plot-window *y* minimum in data units
`get_dz0()`	Plot-window *z* minimum in data units
`get_dx1()`	Plot-window *x* maximum in data units
`get_dy1()`	Plot-window *y* maximum in data units
`get_dz1()`	Plot-window *z* minimum in data units
`get_xdel()`	Plot window *x* range in data units
`get_ydel()`	Plot window *y* range in data units
`get_zdel()`	Plot window *z* range in data units
`get_wx0()`	Plot-window *x* minimum in internal units
`get_wy0()`	Plot-window *y* minimum in internal units
`get_wx1()`	Plot-window *x* maximum in internal units
`get_wy1()`	Plot-window *y* maximum in internal units
`get_px0()`	Page-window *x* minimum in internal units
`get_py0()`	Page-window *y* minimum in internal units
`get_px1()`	Page-window *x* maximum in internal units
`get_py1()`	Page-window *y* maximum in internal units

The following sample code generates data coordinates *x* and *y* associated with arbitrary internal unit coordinates *cx* and *cy*, taking into account whether the plot window is in portrait or landscape mode and whether or not the axes are using logarithmic scaling. Such a transformation would be useful when writing a user function that moves cross hairs over a screen containing a plot drawn by a C-PLOT filter.

```
if (is_tu()) {
    x = get_dx0() + get_xdel() * (cy - get_wy0()) / (get_wy1() - get_wy0());
    y = get_dy0() + get_ydel() * (get_wx1() - cx) / (get_wx1() - get_wx0());
} else {
    x = get_dx0() + get_xdel() * (cx - get_wx0()) / (get_wx1() - get_wx0());
    y = get_dy0() + get_ydel() * (cy - get_wy(0)) / (get_wy1() - get_wy0());
}
if (is_xlog())
    x = pow(10., x);
if (is_ylog())
    y = pow(10., y);
```

If the range-changing macros are used to modify the range values within a user function, the above macros will continue to report the original values of the plot window data coordinates until the function is reinvoked after returning to the plot program.

Number of points

You must explicitly set the total number of points if it is changed in a type 4 user function. Use the following macro to do so. Don't try to change the number of points in types 1 to 3 functions.

Name	What it Does
`set_npts(`*n*`)`	Set the number of points to *n*
`get_npts()`	Value is the current number of points

Miscellaneous macros

These macros provide some status information and allow you to break the plot program out of a command file if that is appropriate.

Name	What it Does
`is_bgnd()`	Nonzero if running in the background
`is_quiet()`	Nonzero if quiet mode is on (see `zq`)
`get_fun_num()`	Value is function number, as in `f1`, `f2`, `f3`
`set_error()`	When function returns, plot program resets to command level

The last macro sets a flag that makes the plot program behave just as if the user typed `^C` at the keyboard, but the action doesn't occur until the user function returns control to the plot program.

For example, if the user function is reading a data file and has reached the end of that file, you might have code such as,

```
user4() {
        ...
        if (fgets(buf, 128, fd) == NULL) {
                printf("Reached end of file.\n");
                set_error();
                return;
        }
        ...

}
```

Setting and retrieving data points

The following macros are only relevant with type 4 user functions. Since C-PLOT may keep data points in temporary files, access to the points is through functions rather than a static array. The functions, described later, require the address of the C-PLOT data structure, which is named `Point`. Within your C code, you declare storage for one of these structures and access it through the following macros, which also require the address of the storage of the `Point`.

Name	What it Does
`set_x(`*p*`, `*x*`)`	Assign *x* to `Point` whose address is *p*
`set_y(`*p*`, `*y*`)`	Assign *y* to `Point` whose address is *p*
`set_z(`*p*`, `*z*`)`	Assign *z* to `Point` whose address is *p*
`set_r(`*p*`, `*r*`)`	Assign *r* to `Point` whose address is *p*
`set_s(`*p*`, `*s*`)`	Assign *s* to `Point` whose address is *p*
`set_up_pen(`*p*`)`	Assign *up* line control to `Point` whose address is *p*
`set_down_pen(`*p*`)`	Assign *down* line control to `Point` whose address is *p*
`get_x(`*p*`)`	Value is *x* of `Point` whose address is *p*
`get_y(`*p*`)`	Value is *y* of `Point` whose address is *p*
`get_z(`*p*`)`	Value is *z* of `Point` whose address is *p*
`get_r(`*p*`)`	Value is *r* of `Point` whose address is *p*
`get_s(`*p*`)`	Value is *s* of `Point` whose address is *p*
`get_pen(`*p*`)`	Value is non-zero if `Point` has *up* line control

The following functions provide the interface between your code and the routines in the overhead modules.

Data generation

These are the routines you provide that will be called by the overhead module routines.

Name	What it Does
`setup()`	Called once each time user function is invoked
`user(`*x*`)`	Returns value for *y* in a type 1 function
`user_x(`*t*`)`	Returns value for *x* in a type 2 function
`user_y(`*t*`)`	Returns value for *y* in a type 2 function
`user_sx(`*t*`)`	Returns value for *r* in a type 2 function
`user_sy(`*t*`)`	Returns value for *s* in a type 2 function
`user_x(`*x*`, `*y*`, `*r*`, `*s*`)`	Returns value for *x* in a type 3 function
`user_y(`*x*`, `*y*`, `*s*`)`	Returns value for *y* in a type 3 function
`user_sx(`*x*`, `*y*`, `*s*`)`	Returns value for *r* in a type 3 function
`user_sy(`*x*`, `*y*`, `*r*`, `*s*`)`	Returns value for *s* in a type 3 function
`user_4()`	Called once in a type 4 function

In 3D mode, replace *r* with *z*.

All the functions except `setup()` return values of type `double`. The arguments are all of type `double`. All but `user4()` are called once for each data point. The argument `t` is the parametric variable in the type 2 user functions. Otherwise, the arguments are the values of the appropriate coordinate of the current data point. In type 2 and 3 user functions, if you wish to keep the current value of a coordinate, simply return the argument.

Getting command line options and keyboard input

Several subroutines are provided to simplify dialogue with the user of your C-PLOT user functions. Type 1 to 4 user functions do not read from a C-PLOT command file. To control the options within a user function when running from command files, extract fn command-line arguments using the get_cmdbuf() or get_args() routines provided. When the user function is run interactively the three get_?num() functions provide a convenient way of obtaining options from the keyboard.

Name	What it Does
get_cmdbuf(*b*)	Copy command line to character buffer *b*
get_args(*fmt* [, *ptr* ...])	Scan command line for arguments
get_dnum(*prompt*, *dptr*)	Input double from keyboard
get_inum(*prompt*, *iptr*)	Input integer from keyboard
get_snum(*prompt*, *sptr*)	Input string from keyboard

In the above function calls, *b*, *fmt*, *prompt* and *sptr* are type (char *), *dptr* is type (double *) and iptr is type (int *). The optional arguments to get_args() are all pointers with their type depending on the contents of the fmt string. There can be no more than 26 of these pointers.

The size of the character buffer b used in get_cmdbuf() should be at least CMD_LEN bytes, where CMD_LEN is a manifest constant that is declared in the include files.

To clarify, here are examples of the usage of these functions. For get_args(), the rules for fmt, the optional pointers and the return value are just as they are for the standard sscanf() routine in the C library. If a function is invoked from the plot program as

```
fn name.1 arg1 arg2 arg3 arg4 ...
```

the scanning specified by format begins at arg1. Here is an example

```
setup() {
        int     args;
        double  p1, p2;

        args = get_args("%lf %lf", &p1, &p2);
        ...
}
```

For each of the last three routines in the above table, prompt, if nonzero, should point to a string that will be printed along with the current value of what is pointed to by the second argument. When the routine is called, a line of text will be read from the keyboard and scanned for something to stuff into the location pointed to by the second argument. If no appropriate object is found on the line of text, the contents of the location pointed to by the second argument remain unchanged. For the following code,

```
{
        static  char    file[64] = "data";
        static  double  temp = 98.6;

        get_snum("What file", file);
        get_dnum("What temperature", &temp);
        ...
}
```

the output is

```
What file (data)? <return>
What temperature (98.6)? <return>
```

The return values for each of these three functions are 1 if the user simply hits return, 0 if the user types some input and −1 on end-of-file (the user hit a ^D).

Getting and setting data points

In order to transfer information about a data point between your code and the overhead modules with type 4 user functions, these built-in routines are available

Name	What it Does
pt_get(*p*, *i*)	Fetches data point *i* into Point whose address is *p*
pt_set(*p*, *i*)	Sets data point *i* from Point whose address is *p*

The following code fragment sets the *x* and *y* values for 100 points. The error-bar value and line-control state only need to be set once since the same storage for the Point is reused each time.

```
{
        Point   pt;             /* Declare storage */

        set_s(&pt, 0);          /* Error bar = 0 */
        set_down_pen(&pt);      /* Set pen down */
        for (i=0; i<100; i++) {
                set_x(&pt, i);      /* x = i */
                set_y(&pt, i*i);    /* y = i * i */
                pt_set(&pt, i);     /* Set values */
        }
        set_npts(i);            /* Set number of points */
        new_points();           /* New ranges */
}
```

Getting and setting auxiliary data arrays

The user function overhead modules also provide you with a method of storing auxiliary data arrays that have the same number of elements as there are data points. This auxiliary data is also kept in a temporary file if there is not enough room in program memory.

The following functions handle the file control.

Name	What it Does
fpt_init(*ptr*, *size*)	*ptr* points to static storage of *size* bytes
fpt_get(*i*)	Fetch the *i*th element
fpt_set(*i*)	Set the *i*th element
fpt_fini()	Bring the temporary file up to date

The function fpt_init() is used to initialize the auxiliary storage routine. You supply it with a pointer to the data structure you are using and tell it how big that structure is. The fpt_get() and fpt_set() routines then use that storage you have allocated to transfer the data in and out.

For example,

```
{
        Point   pt;
        static  struct  aux {
                float   a_1;
                float   a_2;
                float   a_3;
        } aux;
        static  once;
        ...
        if (!once) {
                /* Initialize storage just once */
                fpt_init(&aux, sizeof(aux));
                once++;
        }
        for (i = 0; i  get_npts(); i++) {
                pt_get(&pt, i);
                aux.a_1 = get_x(&pt);
                aux.a_2 = get_x(&pt) * 2;
                aux.a_3 = get_x(&pt) * 3;
                fpt_set(i);
        }
        fpt_fini();
        ...
}
```

fpt_init() can be called more than once to reinitialize the storage, perhaps using a different data structure. Every time it is called, though, the previously stored numbers are wiped out. Call it just once and the values stored will be retained over successive invocations of the user function when it is reinvoked with the fn . syntax.

Quitting the function

The following function is used for handling serious errors. It uses the longjmp() C-library routine to jump to the overhead module code that returns control to the plot program. It also performs the set_error() command, described earlier, so that the plot program returns to command level.

Name	What it Does
quit_func()	Immediately return control to the plot program

The following function and variable names are reserved.

dtof	fpt_pos	pdata	pt_pos
back_gnd	old_point	pl	sa
cintflag	oncint	point	xa
fintflag	onfint	prnt_point	ya