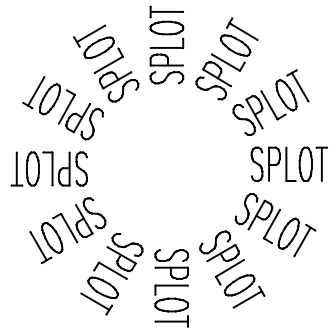


Splot Manual

T.W. Steiner

1994

THIS PROGRAM WAS WRITTEN BY THOMAS W. STEINER.
COPYRIGHT 1992 - 1994, ALL RIGHTS RESERVED.
THERE IS NO WARRANTY OF ANY KIND. USE AT YOUR OWN RISK.
THIS IS NOT FREEWARE. TO REGISTER YOUR COPY SEND $50 TO
T.W. Steiner, 312 - 1230 Haro St.
Vancouver, BC, V6E - 4j9, Canada
e-mail steiner@sfu.ca

C o n t e n t s

1 Introduction **5**

2 Using Splot **7**
2.1 Splot Basics - Getting Started . . . 7
2.1.1 Making a Drawing . . . 9
2.1.2 Making Plots . . . 10
2.1.3 Using The Mouse . . . 10
2.1.4 Actions Associated with the Mouse Buttons . . . 11
2.1.5 Hard Copy . . . 11
2.2 Miscellaneous notes . . . 12
2.3 Carrying On . . . 13
2.3.1 Text . . . 13
2.3.2 Symbols . . . 24
2.3.3 Gsave/Grestore . . . 24
2.3.4 Insets . . . 26
2.3.5 Tick Marks and Tick Labels . . . 30
2.3.6 clipping . . . 31
2.3.7 units . . . 31

3 Hardware Requirements and Installation **33**

4 Splot Library Functions **35**
4.1 Drawing Library Reference . . . 35
4.1.1 Drawing Functions . . . 35
4.1.2 'set' Functions . . . 42
4.2 Standard C Function Library Reference . . . 45

5 Sample Programs **49**
5.1 simple.spt . . . 49
5.2 simple2.spt . . . 50
5.3 simple3.spt . . . 50
5.4 inverax.spt . . . 51
5.5 errorbar.spt . . . 52
5.6 manyspec.spt . . . 53
5.7 invisax.spt . . . 56
5.8 landscap.spt . . . 61
5.9 intcalc.spt . . . 63
5.10 linedraw.spt . . . 66
5.11 multplot.spt . . . 73

6 Splot Definitions **77**

7 Setable Parameters **81**

8 Editor **83**
8.1 Basics 83
8.2 Block manipulations 85
8.3 Macros 85
8.4 Command Line 86
8.5 configuration file 88

9 Acknowledgements **89**

Chapter 1

Introduction

Scientific plotting programs fall into two main categories. The first is the non interactive type represented by most mainframe plotting programs which require that the plot be specified as a file of plotting instructions. This file is edited using a text editor and then the plot is generated by executing this file. The second major type of plotting programs are those popularized on PCs which allow a user to specify the drawing using a mouse and are inherently interactive. This type of program is easier to use and it is possible to rapidly make changes. It is however not ideal since the drawing is usually stored in a proprietary, non-text format which is difficult to interpret. This makes the task of reusing parts of a previous drawing in a new drawing or simply plotting new data in an old figure difficult or impossible. Furthermore, these programs invariably do not have the flexibility to handle the needs of preparing scientific plots. Splot has been developed with the goal of bridging these two disparate approaches in an attempt to gain the benefits of both.

Splot is geared towards generating two dimensional, publication quality scientific plots as painlessly as possible without sacrificing flexibility and power. Splot contains much of the functionality of Postscript™ and can thus be used as a general purpose drawing tool. Furthermore, it has functions specifically aimed at producing plots of x,y data pairs. Splot generates a drawing by interpreting a C program which specifies the drawing. Unlike traditional mainframe plotting programs this happens quickly since the interpreter and editor are integrated into splot. Specifying the plot in a powerful programming language has several advantages. Since C is currently the most popular programming language in existence many people will already be familiar with its use. This makes C a much better choice than developing a specialized language to be interpreted. Having the drawing specified by a programming language also gives algorithmic capabilities thus allowing repetitive components of a drawing to be generated using a standard loop construct. Data to be plotted can also be manipulated before drawing using the built in standard C math library. C also allows parts of a drawing that are to be used more than once to be specified as a subroutine. This sub component can then be drawn anywhere in the final drawing by first specifying a translation, scale and rotation and then calling the subroutine. Lastly, the description of the drawing is in plain ASCII text allowing it to be easily understood and transferred electronically. Splot contains a built in drawing and plotting function library whose members are called by the C interpreter to generate the drawing.

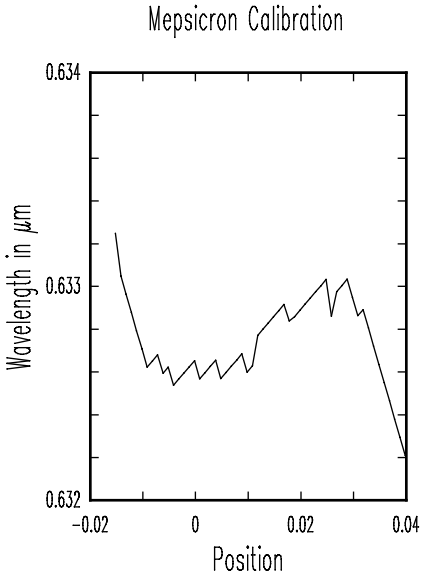
Chapter 2

Using Splot

2.1 Splot Basics - Getting Started

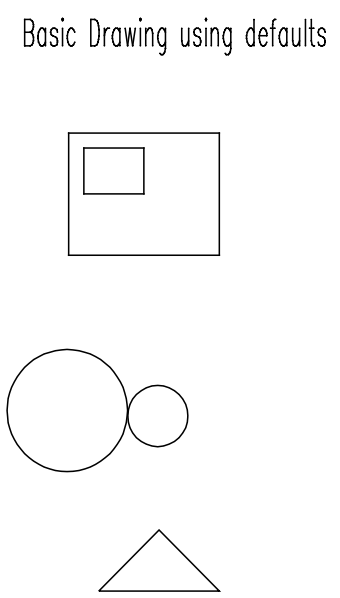
When splot is started the user is presented with two windows. The left window is the drawing area where the image will be drawn. On the right is a text window were the C program specifying the drawing can be displayed. The text is edited using the built in 'e' editor and the drawing is generated by 'executing' the file, i.e. by hitting CTRL G or selecting the menu item 'exec' at the top. Splot can be used for drawing, graphing data or both, and since it is basically C programming, has essentially all the C programming capabilities. To introduce you to the program and demonstrate the basic concept, three simple examples are given below. To begin, an example of a simple splot program to read in and plot a data file using splot default parameters is given. The resultant drawing is shown next to it.

```
/* demo of simplest possible data */
/* plotting using all the defaults */

#include <splot.h>
double *data;
main()
   {
   readdata("demo\data1.dat",data);
   plotdata(data);
   label(LOWER,"Position");
   label(LEFT,"Wavelength in !m!m");
   text(6.20,22.98,"Mepsicron Calibration");
   }
```



Here is a simple example of a drawing using some of the built-in functions and the corresponding drawing.

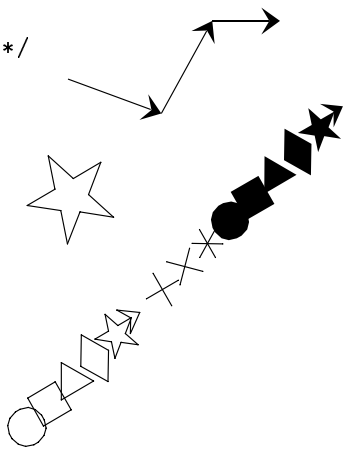
```
/* file basic.spt */
/* Demo of a simple drawing using */
/* some built-in functions */

#include <splot.h>
main()
   {
   /* draw two boxes */
   box(6.0,20.0,11.0,16.0);
   box(6.5,19.5,8.5,18.0);
   stroke();
   /* draw two circles */
   /* input coordinates with mouse */
   arc(5.95,10.91,2.0,0,360);
   stroke();
   arc(8.96,10.73,1.0,0,360);
   stroke();
   /* draw a triangle */
   moveto(7.0,5.0);
   lineto(9.0,7.0);
   lineto(11.0,5.0);
   lineto(7.0,5.0);
   stroke();
   /* add some text */
   text(4.45,23.22,"Basic Drawing using defaults");
   }
```



Lastly, here is an example using some C programming capabilities such as loops.

```
/* This demo file shows the available symbols */
/* which can be used for data points. Symbols names */
/* equating to the numerical constants are defined */
/* in splot.h */

#include "splot.h"
main()
   {
   int i;
   /* make the symbols large. Since they are just a special font */
   /* make the font width large. */
   set(FONTWIDTH,4);
   /* change the symbol orientation just for fun */
   set(FONTDIR,30);
   moveto(1,1);
   /* show all 17 symbols in a diagonal line */
   for (i = 0;i < 17;i++)
      {
      /* use a relative move here */
```

```
    rmoveto(1,1);
    symbol(i);
    }
/* try other form of symbol */
set(SYMMULT,2);
symbol(4,12,OSTAR);
set(SYMMULT,0.5);
/* add some extra arrows to illustrate the */
/* arrowto() function */
moveto(3.82,17.19);
arrowto(7.96,15.68);
arrowto(10.21,19.72);
rarrowto(3,0);
stroke();
}
```



There are two ways a drawing can be made.

1 . by typing in a C program text file using the built-in editor.

2 . by using the mouse and pull-down menus to insert commands into the text file (see section 2.1.3).

Depending on which is more convenient for the task at hand, one can either draw interactively with the mouse or enter text. The drawing is only generated upon execution of the text file.

The text editor that is built into Splot is a fully featured editor with some very powerful capabilities. Ordinary operation should, however, be very intuitive. For a description of the text editor component of Splot refer to Chapter 8. Only a brief introduction to give you the basics to get going is presented here. The text cursor represented by an underscore in the text window can be moved using the arrow key pad or the mouse. Clicking the left mouse button positions the text cursor at the mouse cursor location. Scrolling is accomplished with the page up/down keys or by clicking on the scroll fields along the right side of the text window. A file is written to disk with the name specified on the highlighted status line at the bottom of the text window with a CTRL W (write) command (or use the save menu item). To remove a file use the CTRL Q (quit) command (or use the quit menu item). Splot itself terminates when the last file is removed. To add a new file to the ring of files to be edited or executed type '`<esc> e file.nam`' where file.nam is the name of the desired file or select the open menu item.

2.1.1 Making a Drawing

In order to make a drawing it is useful to understand the underlying philosophy which is based on the postscript imaging model. A very brief introduction to the postscript imaging model will be presented here. To make a postscript like drawing it is first necessary to generate a current path. A path is built using functions such as `moveto(x1,y1);` and `lineto(x2,y2 );`. These functions by themselves do not cause anything to appear on the page. Once a current path is specified then it can be stroked or filled. A path is stroked to the page using the current line width, style, and colour with the `stroke();` function. This function also resets the current path to null. The current values of width etc. are set using the `set(attribute,value);` function were it is advisable to use the predefined values in splot.h for the attributes and values to improve readability of the code (The easiest thing is to select them from the pull down menu). Alternatively a closed current path can be filled with the current colour using the

`fill();` command. This also resets the current path to null. At this point a virtual page in memory has been marked but the actual physical page has not yet been affected. This allows subsequent drawing commands to draw over what is already there. The "paint" is in effect opaque. This means that it is possible to draw over something with the colour set to white and in effect white it out. This can be put to good use for fixing blemishes as shown in some of the example files. The virtual page is transferred to the physical page using the `showpage();` or function or at the end of file execution by default. However, only things within the current clip window are transferred to the physical page. The crucial points to remember are that in order to see anything on the physical page (either on the screen or a printer page) a path must first be stroked or filled and then finally transferred using `showpage();`. This imaging model is very powerful and allows essentially anything to be drawn given the a suite of path building functions. More details on these and other postscript issues can be found in a standard postscript reference.

2.1.2 Making Plots

Splot also contains some custom functions specialized for generating scientific plots. These, such as `plotdata( );`, depart from the postscript model in the sense that they are implicitly stroked. Implicitly stroked functions do not require a `stroke();` in order to mark the page. It is possible to override implicit stroking by starting with a `newpath( );` command.

There is a function called `readdata();` which reads in tabular data from a data file into an array. The data file must be in ASCII format with one line per data point with the x, y and optional error values each in separate columns. This data can then be drawn inside an axes box using `plotdata();` if the default tick positions, size and labelling are acceptable. Alternatively, custom axes boxes, tick marks and labels can be generated using the `axes_box();`, `tickmarks();`, and `ticklabel( );` commands followed by a `drawdata();` or `plotdata();` to draw the data points found in the data array. The difference between `drawdata();` and `plotdata();` is that `plotdata();` will pick default values for the axes box, tickmarks or ticklabels if not previously explicitly specified. The plotted data uses the current line colour, style and width to connect the data points if an ordinary plot was selected. Alternatively, the current symbol can be drawn at each data point. There is a function which automatically draws error bars as well as one which fits and draws the best straight line to a set of data points. The data points actually plotted are easily restricted to those within a range of x or y values. The data can also easily be shifted and scaled before plotting.

2.1.3 Using The Mouse

Clicking the left mouse button on a window changes the current focus if the window does not already have the input focus. On the drawing window the left mouse button is also used to draw the rubber zoom box by holding down the left mouse button. In the editor window double clicking brings up help for the item under the cursor. Help exists for function names, constants and global data values. Double clicking the left mouse button with the mouse on the drawing page inserts the current x,y coordinates into the text file at the current text cursor location. If the mouse was positioned at the beginning of a coordinate pair that coordinate pair is replaced with the new one. If the mouse cursor is in the text region then hitting the left mouse button will move the current text cursor to the mouse cursor location. The left mouse button is also used to select items in drop down menus.

The right mouse button is used for highlighting elements of a drawing. If the mouse cursor is on top of a displayed element on the drawing page and the right mouse button is hit then the corresponding element is highlighted and the text cursor jumps to the corresponding line in the text that generated that element. Conversely, if the mouse cursor is on the text window hitting the right mouse button will highlight any graphics elements generated by the line under the mouse cursor. This feature is very useful for finding the code associated with a particular element of a drawing or the reverse. For the pull down selection lists, the right mouse button opens a help window for those items for which help is available.

When the mouse cursor is on the drawing page the current coordinates are printed at the top. The units and origin used are those in effect at the end of the last program execution. The default units are

cm with the origin in the bottom left corner of the page. Pressing the middle mouse button (or both the left and right mouse buttons together on two button mice) with the mouse pointer on the drawing page causes the current coordinate reference point to be reset so that the status line subsequently reports the distance from this point if the relative option has been selected from the misc menu.

2.1.4 Actions Associated with the Mouse Buttons

There are several mouse menu buttons along the top of the plot window. The action of these buttons is as follows. Starting at the left top there are four buttons labelled 'draw', 'sopt', 'slib' and 'keyw'. These activate drop down menus which allow a function to be selected and inserted into the text. The 'draw' functions are a suite of drawing routines which cause lines, arcs, text etc. to be drawn to the page. The 'sopt' menu is a list of textual names for numerical constants which are used in conjunction with the `set();` command to change attributes of the current graphics state such as the current colour, line width, line style, axes type, etc. Choosing set from the list of drawing functions automatically opens a menu with only an appropriate subset of these and is the recommended approach. The 'slib' button allows selection of a standard C library function. Most of these are math functions such as `sin()` and will not be required for making ordinary plots. They are however useful for manipulating data before plotting or generating data mathematically all from within Splot. The last button of this group 'keyw' opens a list of C keywords for selection. The only one of these that is of interest for ordinary plots is 'main'. Every file that generates a plot or drawing must start with

```
#include <splot.h>
main
   {
   /* list of drawing commands go here between the braces on as many */
   /* lines as necessary */
   }
```

This is all automatically inserted in the text by selecting 'main'. The remaining keywords in this list are only of interest when constructing loops or branches. Note that the actual text inserted for the draw, slib and keyw menu choices can be customized by editing the corresponding macros in the splot.cfg startup file.8.5

The next set of menu buttons along the top are concerned with displaying the drawing. The default size of the drawing is one which fits on a standard 8 x 11 inch piece of paper if printed. However, as the screen resolution is not as good as that of printers it is possible to zoom in on part of a drawing in order to get a better view.

2.1.5 Hard Copy

In order to send a graph to the printer select the print option in the file menu of the editor window. Output can be directed to a file by opening the printer settings notebook and selecting 'print to file' Hard copy is device independent and relies on the OS/2 printer drivers. For example, the demo postscript figures were generated by loading a postscript printer driver and directing the output to a file. Unfortunately, the OS/2 postscript printer driver does not add a %%BoundingBox statement to the generated ps file. This may be needed for encapsulated postscript files which are to be included in documents. To circumvent this `%%BoundingBox 30 30 530 730` was added as the second line of all the figures. This can be done using any text editor including the Splot editor.

2.2 Miscellaneous notes

A drawing can be made in either of two ways. First the drawing can be specified by simply typing in a C program using the built in editor. Of course with this editor it is easy to import components of previous drawings. A drawing can however also be made using the mouse. First the desired drawing function is selected using the drop down menus on the top row. There are menu buttons for the C keywords, drawing functions, miscellaneous functions, and the defined constants. Selecting an item causes it to be inserted in the text at the current text cursor position. Many functions require coordinate points as parameters. These can be entered by moving the mouse to the desired location on the drawing page and double clicking the left mouse button. if the text cursor is currently located at the start of a coordinate pair, that coordinate pair is erased and replaced with a new coordinate when the left mouse button is doubled clicked. In this way it is possible to draw interactively with the mouse or by entering text which ever is more convenient for the task at hand. Parameter errors are not detected until the file is executed using CTRL G or the 'exec' menu button.

2.3 Carrying On

2.3.1 Text

Text can be added to a figure using the `text();` command. The first two parameters are a coordinate pair giving the position at which the text should start. For subsequent lines of text the coordinates can be omitted in which case the line is positioned below the previous line. The third parameter is the actual text to be printed. There is also an optional fourth parameter which allows the text to be centered on the coordinates or end at the coordinates. Specialized text labels are also generated using the `label();` and `ticklabel( );` commands and the following discussion applies to them as well.

The appearance of the text is affected by several set-able parameters. The size of the letters is determined by `set(FONTWIDTH , size_in_cm);` while the ratio of height to width is set using `set(FONTASPECT , height_div_width);`. Alternatively, the size of the text can be changed from its current value by using `set(FONTMULT , multiplier);`. The spacing between letters can be modified using `set(FONTSPACE , space_in_cm);`. The text is ordinarily printed horizontally but this may be changed by using `set(FONTDIR , angle_in_deg);`. The width of the line used to draw the text is set using `set(LINEWIDTH , width);` and the colour is set using `set(LINECOLOUR , colour_number );`. The text is printed using Hershey stroked fonts. There are two available font families simplex and complex. The complex font is chosen using `set(FONT , COMPLEX);`. Greek and Italic text is selected by using special characters in the text string. Thus Greek may be selected by enclosing a string with '!' as in '!abc! while italics are selected using '#'. Super and sub scripts are handled in the same way by enclosing the super/sub script with '^' or '_' respectively. The position and size of the super/sub script relative to ordinary text may be modified using the `set(SCRIPTSCALE , ratio);` and `set(SCRIPTSHIFT , fraction );` commands respectively. Characters from the symbol font may be included in the text string by enclosing the symbol number with '$'.

The various text possibilities are explored in the two following demo files. The third demo file is a table of all the available text characters included in the font files.

```
/* This file illustrates some of the capabilities of the */
/* text() command to annotate figures */
/* Several fonts are available including full Greek. */
/* Text can be any size, orientation and colour. */

#include <splot.h>
main()
   {
   text(2.63,23,"The default position for the next line");
   /* change to complex font for next line */
   set(FONT,COMPLEX);
   text("right below the previous line");
   set(FONT,SIMPLEX);
   text("There is a complete set of Greek characters");
   text("!abcdefghijklmnopqrstuvxyz!");
   set(FONT,COMPLEX);
   text("!ABCDEFGHIJKLMNOPQRSTUVWXYZ!");
   set(FONT,SIMPLEX);
   set(LINECOLOUR,GREEN);
   /* optional change of script scale and position */
   set(SCRIPTSCALE,1);
   set(SCRIPTSHIFT,1);
   text("Here are ^super^ and _sub_ scripts");
   set(LINECOLOUR,RED);
   text("and #Italics# in different colours");
```

```
set(LINECOLOUR,BLACK);
text("Letters can be large, small, tall, short");
text("fat, thin, vertical and angled.");
set(FONTMULT,2);
text(2.94,9.56,"Large");
set(FONTMULT,0.30);
text(8.90,9.71,"small");
set(FONTWIDTH,1.0);
set(FONTASPECT,3);
text(13.10,9.71,"Tall");
set(FONTASPECT,0.5);
text(17.05,9.71,"Short");
set(FONTASPECT,2);
set(LINEWIDTH,0.01);
text(2.94,7.29,"Thin");
set(LINEWIDTH,0.2);
text(9.02,7.29,"Fat");
set(LINEWIDTH,0.05);
set(FONTDIR,90);
text(3.63,1.26,"Vertical");
set(FONTDIR,130);
text(11.28,1.44,"Angled");
}
```

The default position for the next line
right below the previous line
There is a complete set of Greek characters
$\alpha\beta\chi\delta\varepsilon\varphi\gamma\eta\iota\vartheta\kappa\lambda\mu\nu\circ\pi\vartheta\rho\sigma\tau\upsilon\psi\xi\psi\zeta$
$\mathrm{AB\Xi\Delta E\Theta\Gamma HI\Theta K\Lambda MNO\Pi\Theta P\Sigma T\Upsilon\Psi\Omega X\Psi Z}$
Here are $^{\text{super}}$ and $_{\text{sub}}$ scripts
and *Italics* in different colours
Letters can be large, small, tall, short
fat, thin, vertical and angled.

Large small Tall Short

Thin **Fat**

Vertical Angled

```
/* This file illustrates some of the capabilities of the */
/* text() command to annotate figures */

#include <splot.h>
main()
   {
   text(2,23,"The spacing between letters can be changed");
   set(FONTSPACE,0.5);
   text("to be further apart than usual");
   set(FONTSPACE,-0.2);
   text("or closer together.");
   set(FONTSPACE,0);
   text("The size and position of super/sub scripts can");
   text(2,17.38,"be varied as in");
   set(SCRIPTSHIFT,0.7);
   set(SCRIPTSCALE,0.5);
   text(8.94,17.38,"X_1_\b^2^,");
   set(SCRIPTSHIFT,1.0);
   set(SCRIPTSCALE,0.7);
   text(10.19,17.38,"X_1_\b^2^,");
   set(SCRIPTSHIFT,1.5);
   set(SCRIPTSCALE,1.0);
   text(11.4,17.38,"X_1_\b^2^.");
   text(2.10,15.81,"Note how the superscript was placed above");
   text("the subscript using the backspace char \\b.");
   text("Enter degree sign ∅ using '<ALT X> 127'");
   text("Use a \\ before special characters: \_\^\$\#\!\"");
   text("Symbols: $0 1 2 3 4 5 6 7 8 9 : ; < = > $");

   text(2,8,"The default text insertion point is LEFT");
   text(17,6,"but can be set to RIGHT",RIGHT);
   text(10,4,"Or CENTER",CENTER);
   }
```

The spacing between letters can be changed
to be further apart than usual
or closer together.
The size and position of super/sub scripts can
be varied as in X_1^2, X_1^2, X_1^2.
Note how the superscript was placed above
the subscript using the backspace char \b.
Enter degree sign ° using '<ALT X> 127'
Use a \ before special characters: _↑$#!"
Symbols: o □ △ ◊ ☆ › + x * ● ■ ▲ ♦ ★

The default text insertion point is LEFT

but can be set to RIGHT

Or CENTER

```
/* This file displays all the available characters */

#include <splot.h>
#define ROMAN 1
#define GREEK 0
#define ITALIC 0
int i,j;
char c[10];

main()
   {
   set(PAGEROT,ON);
#if ROMAN
   text(13.67,19.14,"Roman",CENTER);
   text(7.5,18,"Simplex",CENTER);
   text(19,18,"Complex",CENTER);
   gsave();
   scale(1.2,1.2);
   translate(1,-11);
   set(FONTWIDTH,0.5);
   set(FONTASPECT,1.5);
   table(0);
   translate(10,0);
   set(FONT,COMPLEX);
   table(0);
   set(FONT,SIMPLEX);
#endif
#if GREEK
   text(13.67,19.14,"Greek",CENTER);
   text(7.5,18,"Simplex",CENTER);
   text(19,18,"Complex",CENTER);
   gsave();
   scale(1.2,1.2);
   translate(1,-11);
   set(FONTWIDTH,0.5);
   set(FONTASPECT,1.5);
   table(1);
   translate(10,0);
   set(FONT,COMPLEX);
   table(1);
   set(FONT,SIMPLEX);
#endif
#if ITALIC
   text(7.5,18,"Italics Simplex/Complex",CENTER);
   text(19,18,"Symbols",CENTER);
   gsave();
   scale(1.2,1.2);
   translate(1,-11);
   set(FONTWIDTH,0.5);
   set(FONTASPECT,1.5);
   table(2);
   translate(10,0);
```

```
   table(3);
#endif
   grestore();
   }

table(int special)
   {
   /* label rows and columns */
   c[1] = '\0';
   for (i = 0; i < 10;i++)
      {
      c[0] = '0' + i;
      tputs(2 + i * 0.7,24.5,c,0);
      }
   c[0] = ' ';
   c[2] = '0';
   c[3] = '\0';
   for (j = 3;j < 10;j++)
      {
      c[1] = '0' + j;
      tputs(0,26 - j,c,0);
      }
   c[0] = '1';
   c[2] = '0';
   c[3] = '\0';
   for (j = 10;j < 13;j++)
      {
      c[1] = '0' + j - 10;
      tputs(0,26 - j,c,0);
      }
   /* prepend each char with '\' so that even special chars get printed */
   c[0] = '\\';
   c[2] = '\0';
   box(1.29,23.77,9.07,13.36);
   stroke();
   /* fill in table */
   for (j = 3;j < 13;j++)
      {
      for (i = 0; i < 10;i++)
         {
         c[1] = j * 10 + i;
         if (c[1] == 'b')
            tputs(2 + i * 0.7,26 - j,"b",special);
         else
            tputs(2 + i * 0.7,26 -j,c,special);
         }
      }
   }

tputs(double x,double y,char *c,int special)
   {
   char tmp[10];
```

```
switch(special)
   {
   case 0:
      text(x,y,c);
      break;
   case 1:
      /* greek */
      strcpy(tmp,"!");
      strcat(tmp,c);
      strcat(tmp,"!");
      text(x,y,tmp);
      break;
   case 2:
      /* italics */
      strcpy(tmp,"#");
      strcat(tmp,c);
      strcat(tmp,"#");
      text(x,y,tmp);
      break;
   case 3:
      /* symbols */
      strcpy(tmp,"$");
      strcat(tmp,c);
      strcat(tmp,"$");
      text(x,y,tmp);
      break;
   }
}
```

Roman

Simplex

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	’
40	(	)	*	+	,	−	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	↑	_	‘	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	°		

Complex

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	’
40	(	)	*	+	,	−	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	↑	_	‘	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	°		

Greek

Simplex

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	’
40	(	)	*	+	,	−	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	X	Δ	E
70	Φ	Γ	H	I	Θ	K	Λ	M	N	O
80	Π	Θ	P	Σ	T	Υ	Ψ	Ω	Ξ	Ψ
90	Z	[	\	]	↑	_	‘	α	β	χ
100	δ	ε	φ	γ	η	ι	ϑ	κ	λ	μ
110	ν	ο	π	ϑ	ρ	σ	τ	υ	ψ	ω
120	ξ	ψ	ζ	{	|	}	~	°		

Complex

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	’
40	(	)	*	+	,	−	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	Ξ	Δ	E
70	Φ	Γ	H	I	Θ	K	Λ	M	N	O
80	Π	Θ	P	Σ	T	Υ	Ψ	Ω	X	Ψ
90	Z	[	\	]	↑	_	‘	α	β	ξ
100	δ	ε	φ	γ	η	ι	ϑ	κ	λ	μ
110	ν	ο	π	ϑ	ρ	σ	τ	υ	ψ	ω
120	ξ	ψ	ζ	{	|	}	~	°		

Italics Simplex/Complex

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	−	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	↑	_	‘	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	°		

Symbols

	0	1	2	3	4	5	6	7	8	9
30										
40									o	□
50	△	◊	☆	›	+	×	∗	●	■	▲
60	⧫	★	›							
70										
80										
90										
100										
110										
120										

2.3.2 Symbols

Symbols can optionally be drawn at every data point by setting `set( PLOTTYPE , SYMBOLS);` or `set( PLOTTYPE , SYM_LINES);`. The actual symbol used is set using `set( CURSYMBOL , sym_number );` Symbols are just a special font and much of the above discussion on how to change a letter's appearance applies also to symbols. There is, however, no complex version of the symbols. Additionally, the size of the symbols can be adjusted relative to the current font size using `set( SYMMULT ,multiplier);`. Symbols can also be drawn independent of `drawdata( );` at an arbitrary location by using `symbol( x , y , sym_number);`. For an example of the symbol capabilities consult the last example of the previous chapter. Symbols may also be included in text strings drawn with `text();` by enclosing the symbol number with '$' in the text string.

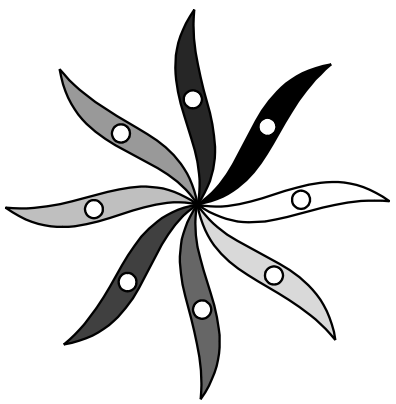
2.3.3 Gsave/Grestore

The pair of functions `gsave();` and `grestore();` are useful for a variety of purposes. `gsave();` stores the complete graphics state including the currently defined path while `grestore();` restores the state to the last saved state. Each `grestore();` must have a matching `gsave();` preceding it.

The simplest use for this pair is to restore the initial state after translations and scaling. This often comes up in composite figures that are constructed from other existing picture. The composite is built by setting translations and scale factors for each component and then drawing the component. If a `gsave();` was done at the beginning then a `grestore();` can be done after drawing the components to restore the initial scale and translation. This can be useful for adding annotations to the final composite figure.

It is also possible to use gsave/grestore to repeatedly draw a given path since the currently defined path is also saved. In the following example a path is defined, saved (using `gsave();`), filled, restored (using `grestore();`) and then finally stroked. Filling and stroking the same path causes the filled region to have a border drawn in the current line colour and style.

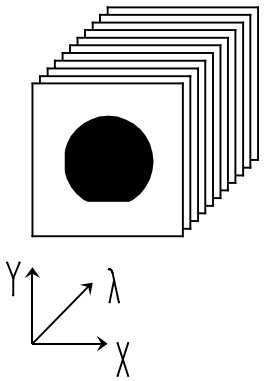
```
#include <splot.h>
/* This is a region filling example */
int i;
main()
   {
   set(LINEWIDTH,0.1);
   translate(10,15);
   for (i=0;i < 8;i++)
      {
      set(LINECOLOUR,i);
      leaf();
      rotate(45);
      }
   }

int leaf()
   {
   arc(3.1,3.38896,0.4,0,360);
   moveto(0,0);
   curveto(3.38924,0.598051,3.18987,3.85411,5.91456,6.11341);
   curveto(2.06013,5.44891,2.52532,1.5948,0,0);
   closepath();
   gsave();
   fill();
```



```
   grestore();
   set(LINECOLOUR,BLACK);
   stroke();
   }
```

The following example again illustrates the use of gsave/grestore to repeatedly draw the same path each time with a small translation to draw a stack of pages. It also illustrates the use of electronic white out (filling with colour white) to remove unwanted lines.

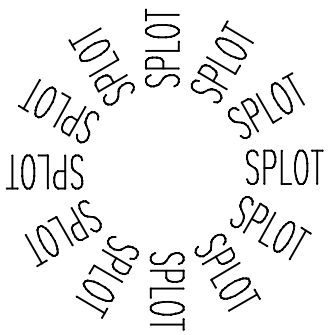
```
#include <splot.h>

int i;
double x,y,d,l;

main()
   {
   /* draw 3d stack of pages */
   x = 5;
   y = 15;
   l = 4;
   for (i = 0;i <= 10;i++;)
      {
      d = i * 0.2;
      box(x - d ,y - d, x - d + l, y - d + l);
      gsave();
      set(LINECOLOUR,WHITE);
      fill();
      grestore();
      stroke();
      }
   /* draw sample */
   arc(5.02,14.96,1.2,0,360);
   fill();
   white_box(3.95,13.51,6.134,13.904);
   white_box(3.862,13.689,3.752,15.443);
   /* draw axes */
   moveto(3.00,10.19);
   rarrowto(0.0,2);
   moveto(3.00,10.19);
   rarrowto(2,0);
   moveto(3.00,10.19);
   rarrowto(1.6,1.6);
   stroke();
   /* add labels */
   text(5.20,9.71,"X");
   text(2.31,11.82,"Y");
   text(5.01,11.64,"!l");
   }

int white_box(double x1,double y1,double x2,double y2)
```



```
   {
   gsave();
   newpath();
   box(x1,y1,x2,y2);
   set(LINECOLOUR,WHITE);
   fill();
   grestore();
   }
```

Using the above technique to repeatedly draw the same element will not work with functions such as `text();` which are implicitly stroked since there is no chance to do a `gsave();` before the stroke. However, implicit stroking can be temporarily turned off with the `newpath();` command. Without the `newpath();` in the following example only the first "SPLOT" would have been printed. The first `stroke();` after a `newpath();` turns on implicit stroking again with these functions.

```
#include <splot.h>
main()
   {
   int i;
   newpath();
   translate(10,10);
   text(0,0,"      SPLOT");
   for (i=0;i < 360;i += 30)
      {
      gsave();
      rotate(i);
      stroke();
      grestore();
      }
   }
```



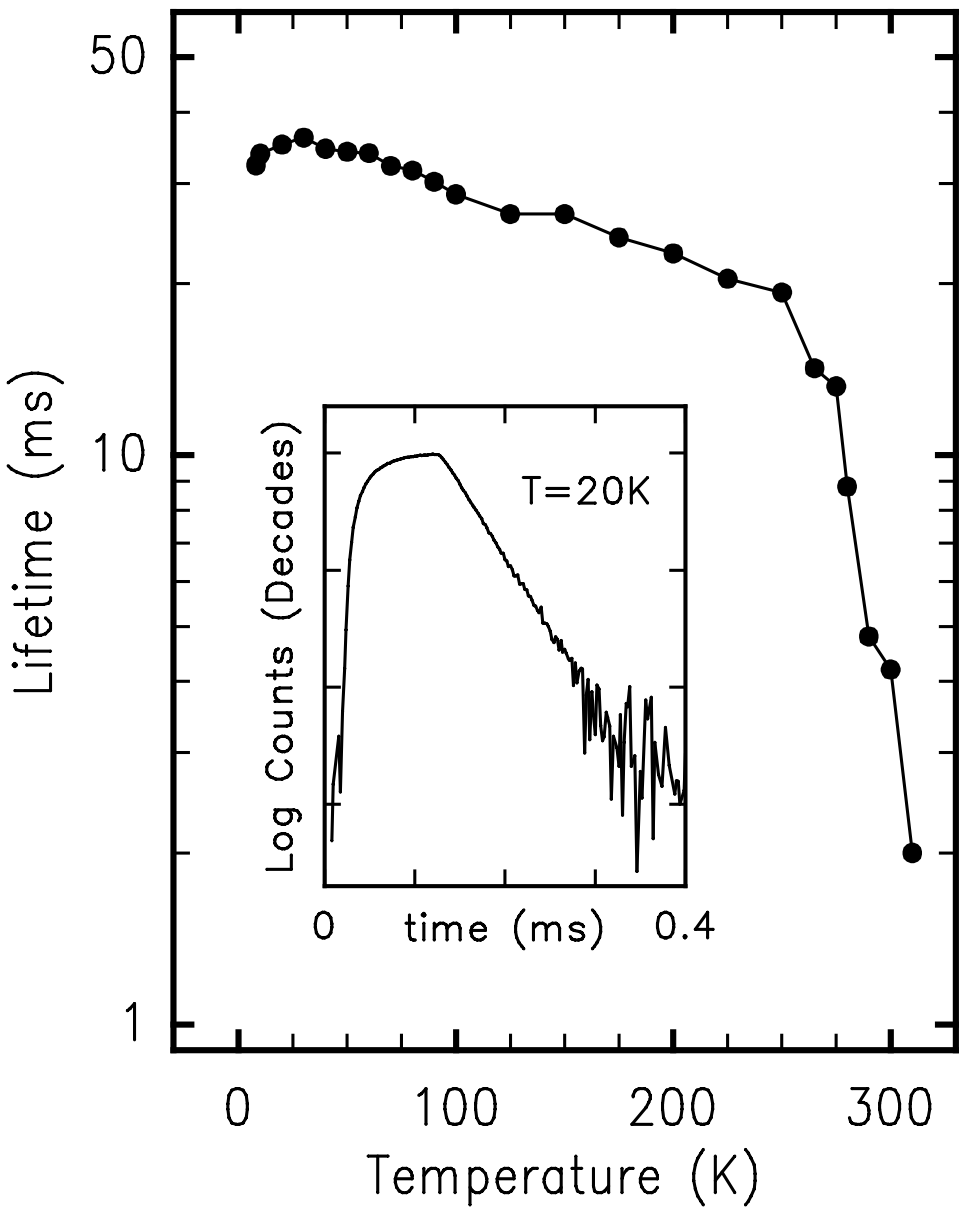
2.3.4 Insets

Inset figures can easily be made from existing figures by moving the origin to the desired insertion point using `translate(xpos,ypos );`, changing the scale with `scale(xfact,yfact);`, and copying the existing code defining the inset figure into the current file using the editor block move functions. It is also useful to bracket the inset figure with a `gsave();` / `grestore();` pair in order to restore the original coordinates after the inset.

```
/* This figure demonstrates how to use axes other */
/* than the standard linear axes. Also it illustrates */
/* how insets can be made. Insets alternatively can */
/* be made by encapsulating an entire drawing in a */
/* subroutine. */

#include <splot.h>

double *lifetime, *rawtime;
```

```
main()
   {
   int i;
   double majortic,minortic;

   /* define a tic length  */
   /* major and minor tics */

   majortic=0.3;
   minortic=0.25;

   /* set up some default values */

   set(FONTASPECT,1.2);
   set(FONTWIDTH,0.8);
   set(PLOTTYPE,SYM_LINES);
   set(CURSYMBOL,CIRCLE);
   set(AXESCLIP,ON);
   set(LINEWIDTH,0.1);
   set(AXESTYPE,LOGY);

   /* get the data and set the box  */

   readdata("demo\lifetime.dat",lifetime);
   /* draw axes box but not centered on the page */
   axes_box(13,17,-30,0.9,330,60,5.51,7.5);

   /* x-axis   */

   /* major tics*/

   set(TICKLENGTH,majortic);
   tickmarks(XAXES,0,100,200,300);

   /* minor ticks */

   set(TICKLENGTH,minortic);
   set(LINEWIDTH,0.05);
   tickmarks(XAXES,25,50,75,125,150,175,225,250,275);
   ticklabel(BOTTOM,0,"0",100,"100",200,"200",300,"300");

   /* y-axis */

   /* major tics*/

   set(TICKLENGTH,majortic);
   set(LINEWIDTH,0.1);
   tickmarks(YAXES,1,10,50);

   /* minor ticks */

   set(TICKLENGTH,minortic);
```

```
set(LINEWIDTH,0.05);
tickmarks(YAXES,2,3,4,5,6,7,8,9,20,30,40);
ticklabel(LEFT,1,10,50);

/* axis labels */

label(LOWER,"Temperature (K)");
label(LEFT,"Lifetime (ms)");

drawdata(lifetime,0,1);

/* now build the inset figure */
/* insert  setup stuff */
set(PLOTTYPE,LINES);
set(LINEWIDTH,0.065);
set(AXESTYPE,LOGY);
readdata("demo\rawtime.dat",rawtime);
axes_box(6,7.85,0.00,20,0.400,250000,8.02,10.19);
set(LINEWIDTH,0.05);
tickmarks(XAXES,0.1);
tickmarks(YAXES,1);
set(FONTASPECT,1.0);
set(FONTWIDTH,0.67);
ticklabel(BOTTOM,0,0.4);
drawdata(rawtime,0,1);
text(11.30,16.64,"T=20K");
label(LEFT,"Log Counts (Decades)");
set(LABELMARG,-0.8);
label(BOTTOM,"time (ms)");
}
```

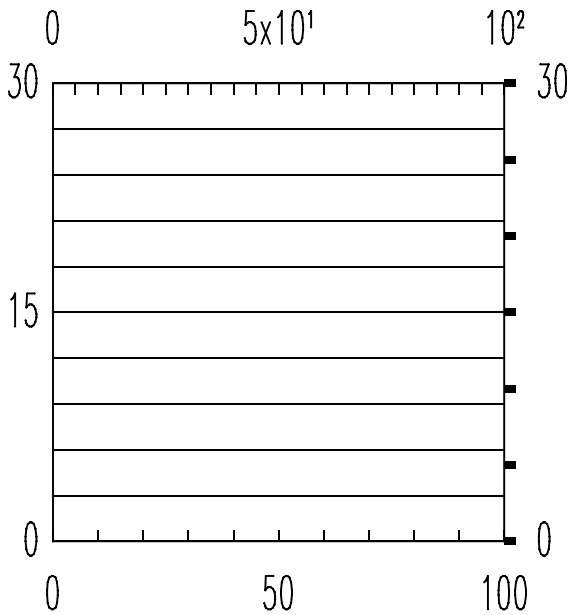

50
10
1
Lifetime (ms)
0
100
200
300
Temperature (K)
T=20K
Log Counts (Decades)
0
time (ms)
0.4

2.3.5 Tick Marks and Tick Labels

Tick marks are added to an existing axes box using the `tickmarks();` command. At its simplest no parameters are required and Splot tries to select a set of reasonable tick marks for both the x and y axes. Alternatively, a list of tick positions can be provided for each axis. Tick marks are lines and so the width of a tick mark is set using `set (LINEWIDTH, width);`. The length of a tick mark is set using `set(TICKLENGTH, length);`. Grid lines may be produced by choosing the length the same as the length of the axes box. If tickmarks on the outside of the axes box are desired they may be generated by setting a negative tick length.

Tick labels are produced with the `ticklabel();` command. At its simplest no parameters are required and Splot labels a subset of the existing tick marks on the bottom and left side. Alternatively, an axis may be specified along with a list of tick label positions. At its most powerful each tick position may be paired with a text string allowing any arbitrary string to appear at the desired location. In this way tick labels need not be in the same units as those used to draw the graph. Tick labels are just text and tick label appearance is affected by the text attributes discussed in 2.3.1.

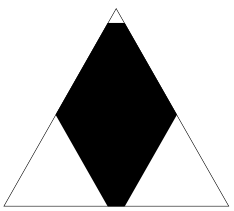
```
#include <splot.h>
main()
   {
   /* tick marks and tick label example */
   abox(12,12,5,6);
   ascale(XAXES,0,100);
   ascale(YAXES,0,30);

   /* default tick marks and labels */
   tickmarks(BOTTOM);
   ticklabel(BOTTOM);

   gsave();

   /* grid lines */
   set(TICKLENGTH,12);
   tickmarks(LEFT);
   ticklabel(LEFT);

   /* thick -ve ticks at specified positions  */
   set(LINEWIDTH,0.2);
   set(TICKLENGTH,-0.3);
   tickmarks(RIGHT,0,5,10,15,20,25,30);
   set(LINEWIDTH,0.05);
   set(TICKLMARG,0.5);
   ticklabel(RIGHT,0,30);

   grestore();

   /* ticks by spacing and custom labels */
   tickmarks(TOP,5);
   ticklabel(TOP,0,"0",50,"5x10^1^",100,"10^2^");

   }
```



2.3.6 clipping

A drawing can be clipped to an arbitrary path by first building a path with the path building functions such as `moveto();` and `lineto();` and then setting it up as the clip path with the `clip();` function. The clip path must be set before the drawing to be clipped is made. In the following example a triangular path is made and then saved using `gsave();`. This path is then set up as the current clip path using the `clip();` function. The path is then restored and stroked in order to show the clip path. The clip path is not ordinarily visible on the page. Finally the path is restored once more the y axis mirrored and the triangle filled. The result is the inverted triangle clipped against the original triangle.

```
#include <splot.h>
main()
   {
   translate(10,13);
   moveto(-4,-4);
   rlineto(10,0);
   alineto(10,120);
   closepath();
   gsave();
   clip();
   grestore();
   gsave();
   stroke();
   grestore();
   /* mirror about y axis */
   scale(1,-1);
   fill();
   }
```



Data drawn using `drawdata();` or `plotdata();` may be clipped to the most recently drawn axes box by setting `set(AXESCLIP,ON );` before calling `plotdata();` or `drawdata();`. This makes use of the clipping mechanism internally by temporarily adding the axes box to the clip path and then restoring the previous clip path after the data is drawn.

2.3.7 units

The default units for the drawing page are centimeters. The physical page available for drawing in is 20 cm wide by 26 cm high (approximately 8 x 11 inches). It is not possible to change the physical size of the drawing area but the units can easily be changed using the `scale(xfact, yfact);`scale command. Thus in order to change the units to inches for example put `scale(0.3937,0.3937);` right after `main()` in the file. The default orientation of the page is portrait with the x axes along the short side. The origin may be shifted using `translate();` and x and y interchanged using `rotate(90);`. It is thus possible to change to landscape mode using a combination of `translate();` and `rotate();` but it is much easier to just specify `set(PAGE_ROT, ON);` at the top of the file after `main()`.

It is sometimes convenient to change the coordinates of the page to match those used to draw data within the axes box. This can be useful for adding labels to features in the data. This could be done using `translate();` and `scale();` but is much more easily achieved with the function `cmatch(ON);` in the code or the coordinate match menu choice in the misc menu. The menu button is a toggle and alternately turns on and off the coordinate matching. If there are multiple axes boxes in a figure the last drawn one is used to match to. This may be changed by high lighting a different axes box using the right mouse button.

Chapter 3

Hardware Requirements and Installation

Splot will run as a PM application on any computer running OS/2. An extended DOS version with slightly less functionality is also available. Disk space requirements are modest. The executable, help, demo and font files take up less than one Mb of disk space. The LaTeXdocumentation splotdoc.tex and associated postscript example figures take up another 2-3 Mb.

To install Splot make a directory for it copy the distribution file to it and then use an un-zipping utility to un-zip the zipped distribution file. Be sure to use the -d flag so that the necessary sub directories are created if your un-zipping utility requires it. This should create three sub directories one containing the fonts, another containing the example programs and a third for the documentation. The executable and help files should be in the current directory. It is also necessary to set an environment variable so that Splot can find the configuration and help files. This can be done by running the supplied splotpth.cmd file or adding the following to your config.sys:

SET SPLOTPATH=c:\splot\

where 'splot' is the name of the directory containing the *.hlp and splot.cfg files. The trailing \ is required.

It is recommended that the actual drawing files be put in a sub directory of the main Splot directory.

To run the program type os2splot on a command line followed by an optional file name(s). Alternatively after setting up an association between *.spt files and os2splot.exe files can be dragged to the splot icon to start Splot.

Chapter 4

Splot Library Functions

What follows is a list of the built in library functions. This document does not include information on C in general or details of the editor usage. Editor documentation is found in chapter 8 and C information can be found in any number of readily available texts.

4.1 Drawing Library Reference

4.1.1 Drawing Functions

void **abox**(double xsi, double ysi, double xorig, double yorig);

> Adds an axes box to the current path. The box is drawn with a size of xsi by ysi and centered on the page. No internal coordinate system is set up. Use ascale() for this or use axes_box() to combine both functions. The last two parameters are optional and if present, specify the position of the axes origin relative to the page origin. For the last two parameters the special constants XCENTER and YCENTER can be used for the x or y position coordinate respectively and will cause the axes box to be centered on the page along that axis. Using both XCENTER and YCENTER is equivalent to the default behaviour with the last two parameters absent. abox() is implicitly stroked.

void **alineto** (double len, double ang);

> Adds a line of length len at an angle of ang with respect to the previous line to the current path. Generates an error if there is no previous line in the current path.

void **arc**(double xcen, double ycen, double rad, double alpha, double beta);

> Adds a circular arc of radius rad centered at (xcen, ycen) to the current path. The starting angle is alpha and the stopping angle is beta. The arc is drawn in the counter clockwise direction. A straight line section will be added from the previous current point if any to the starting point of the arc.

void **arcn**(double xcen, double ycen, double rad, double alpha, double beta);

> Adds a circular arc of radius rad centered at (xcen, ycen) to the current path. The starting angle is alpha and the stopping angle is beta. The arc is drawn in the clockwise direction. A straight line section will be added from the previous current point if any to the starting point of the arc. Exactly as arc() but draws the arc in the opposite direction.

void **arcto**(double x1, double y1, double x2, double y2, double rad);

Adds a circular arc of radius rad to the current path. The center and angles are chosen so that the arc is tangent to the line formed by (x1,y1) and the current point at its start and tangent to the line (x1,y1) - (x2,y2) at its end point. A straight line segment is added from the current point to the start of the arc. An error is generated is there is no current point.

void **arrowto**(double x,double y,...);

Adds a line segment to the current path from the current point to (x,y). The line is terminated by drawing an arrow head oriented in the direction of the line. More than one coordinate point can be specified in the command in which case a series of line segments terminated by arrows connecting the points will be added to the current path. If there is no current point then an error will be generated. The size of the arrow head may be changed with the set(FONTWIDTH,...); command.

void **ascale**(int axes, double xstart, double ystart, double xend, double yend);
or
void **ascale**(int axes, double *data, int col,...);

An internal coordinate system is set up for subsequent plotting of data within the existing axes box. The x axis start and stop values are xstart and xend respectively and ystart, yend for the y axis. for the y axis. If the axis choice is XAXES or YAXES rather than XYAXES then only two numbers follow the axes specifier rather than four. This allows the x and y axes scales to be set independently of each other. The alternate format allows for auto scaling to the specified data. There can be more than one data array in the list in which case the scales are chosen so that they will all fit. Each data array can optionally be followed by one or two integers specifying which columns to use for the x and y values. If axes is not XYAXES only one integer is allowed.

void **axes_box**(double xsi, double ysi, double xstart, double ystart double xend, double yend, double xorig, double yorig);

Adds an axes box to the current path. The box is drawn with a size of xsi by ysi and centered on the page. An internal coordinate system is set up for subsequent plotting of data within the box. The x axis start and stop values are xstart and xend respectively and ystart, yend for the y axis. for the y axis. The last two parameters are optional and if present specify the position of the axes origin relative to the page origin. For the last two parameters the special constants XCENTER and YCENTER can be used for the x or y position coordinate respectively and will cause the axes box to be centered on the page along that axis. Using both XCENTER and YCENTER is equivalent to the default behaviour with the last two parameters absent. More flexibility is provided by the pair of functions abox() and ascale() which provide the functionality of axes_box() in several steps. axes_box() is implicitly stroked.

void **box**(double x1, double y1, double x2, double y2);

Adds a box to the current path. The two end points of the box are (x1,y1) and (x2,y2);

void **clear**();

Clears the screen when the program is executed. This function should not be needed for ordinary plots.

void **clip**()

Converts the currently defined path into a clipping path. All subsequent drawing operations are then clipped against this path and only portions of the drawing on the inside of the clip path are displayed. What is inside and what is outside depends on whether even-odd or non-zero wind has been selected as the fill rule using a set(); If the current path is not closed then the current path is first closed. If there is no current path an error is generated. clip() is implicitly stroked and takes effect immediately.

void **closepath**();

Closes the current path. A straight line segment is added from the current point to the start of the current path as set by the moveto() of rmoveto() command at the beginning of the path definition. Generates an error if there is no current point.

void **cmatch**(int on);

If the passed parameter is TRUE then it does the necessary translations and scaling so that the coordinate system for the page matches that used within the axes_box. An error is generated if there is no current axes_box. The font scale is compensated automatically for the change in coordinate system so that characters will still be the same size as before. If the parameter is FALSE then the previous unmatched coordinates will be restored. If there is no parameter TRUE is assumed.

void **curveto**(double x1, double y1, double x2, double y2, double x3, double y3);

Adds a Bezier curve section to the current path starting at the current point. The curve starts tangent to (xcur,ycur) - (x1,x2) and ends tangential to (x2,y2) - (x3,y3) at (x3,y3); An error is generated if there is no current point.

void **drawdata**(double *data, int xcol, int ycol);

Draws the data contained in the array data in the current axes_box. The two numbers xcol and ycol are optional and are the columns of the array data that are to be used for the x-axis and y-axis data respectively. If these values are omitted 0 , 1 are assumed. Each row of the array data represents one coordinate point to be plotted. The values will be plotted using the internal coordinate system established by the the call to axes_box. If there is no current axes box an error will be generated. drawdata() is implicitly stroked.

void **errorbars**(int axes,double *data, int xcol, int ycol, int errcol);

Draws error bars for the data points in the array "data". The first parameter is either XVALS or YVALS indicating along which axes the error bars are to be drawn. The numbers xcol and ycol are the columns of the array data that are to be used for the x-axis and y-axis data respectively. Each row of the array data represents one coordinate point to be plotted. The values will be plotted using the internal coordinate system established by the the call to axes_box. The last parameter "errcol" is the column of the array "data" which holds the size of the error for the corresponding data point in the same row. If there is no current axes box an error will be generated. errorbars() is implicitly stroked.

void **fill**();

Closes the current path if not already closed and fills the interior region with the current colour as specified by the last set( ) call. What is inside and what is outside the path depends on the currently chosen fill rule. The fill rule is either even-odd or non-zero wind (the default) and is specified using a set() call. An error is generated if there is no current path.

void **fitline**(double *data, int xcol, int ycol,double *yint,double *slope);

Fits the best straight line to the data in the array "data" using the column xcol of the array as the x values and the column ycol as the corresponding y values. The best line is drawn constrained to the current axes box. If there is no current axes box an error is generated. The last two parameters are the returned values giving the y intercept and slope of the fitted line. Note that they are pointers to doubles which must be declared at the top of the program. Given a declaration of the form: double slope,yint;

Call fitline using:

```
fitline(data,0,1,&yint,&slope);
```

Assuming that the first and second column of data are the x and y values respectively. Data points can be excluded from the fit by using the set(XRANGE, xmin, xmax); or set(YRANGE, ymin, ymax); commands. Only data points within the limits will be used for the fit. fitline() is implicitly stroked.

int or double **get**(int option,char *str);

Returns the value of the specified option in the current graphics state. The numerical value for single valued elements is returned by the function otherwise the return value is 0. A text representation of the value is optionally returned in str. If used, be sure to allocate a character array with sufficient space for str before calling this function as in `char str[80];` at the top of the program. The valid option values are defined in splot.h. See also set() for a description of the various options. The returned string can be printed using the command puts(str); in the program where "str" is the name of the array in the call to get(). Returned numerical values can be printed using print(value);. The second array parameter is optional and is only really needed for getting options that are more than just a single value such as line patterns.

void **grestore**();

Pops a graphics state off the state stack thereby restoring the graphics state that was in effect at the time the matching gsave() was executed. In particular, the path, the clip path, the line styles, line colours etc. are restored to their previous values.

void **gsave**();

Pushes the current graphics state onto the state stack. The current path, clip path, line style, colour etc. are saved so that they can be restored later using a grestore() command.

void **label**(int axis, char *label);

Add labels to the axes box previously defined. The first parameter is which axis to label either BOTTOM, TOP, LEFT or RIGHT. The following parameter is the label to be printed. The label will be printed centered between the appropriate edges of the current axes box. An error is generated if there is no current axes box. All labels should be after plotdata() if used otherwise spacing from the axes may not be correct. The spacing can also be changed using set(LABELMARG,...);. label(); is implicitly stroked.

void **lineto**(double x,double y,...);

Adds a line segment to the current path from the current point to (x,y). More than one coordinate point can be specified in the command in which case a series of line segments connecting the points will be added to the current path. If there is no current point then an error will be generated.

void **moveto**(double x, double y);

Sets the current point to (x,y). Also sets the path close point to (x,y) for subsequent use with closepath(). Many path building commands such as curveto() and lineto() require that a current point exist before calling them.

void **newpath**();

Resets the current path to NULL and also causes the current point to be undefined. Furthermore, it turns off the implicit stroking of elements that are normally implicitly stroked such as text();. Thus using newpath() these elements can be added to a path which must then be explicitly stroked. stroke() turns on implicit stroking again.

void **plotdata**(double *data,int xcol, int ycol);

Plots the data found in array data in a box. This command chooses the scale sizes tick marks etc. to display the data which is assumed to be in order of monotonically increasing or decreasing x value order. The optional parameters xcol and ycol correspond to the column numbers (starts at zero) of the tabular array which are to be used for the x and y axis values respectively. If these parameters are omitted then the values 0 and 1 are assumed. If the default choices of plotdata are not acceptable a plot can be generated using the step by step method using axes_box(), tickmarks(), ticklabel() and drawdata(). The data must first be read in to array data using readdata( ). The valid option values are defined in splot.h. See also set() for a description of the various options. plotdata() is implicitly stroked.

void **readdata**(char * filename, double * data);

Reads a file of name filename and puts the data into the array "data". The file should be in ASCII format with the x data in one column and the y data values next column(s). Any line containing non numeric characters or is blank will be considered a comment and ignored. WARNING! As an optimization data is only read from disk once if the configuration parameter always_load is off. Subsequent executions use the stored data already in memory (except after a reset(). This means that if you modify the array "data" after reading in values the next time the file is executed you will get strange results. Similarly, if you reuse the same array for different plots in the same drawing you will have trouble. The motto is never change the values in the array "data". If you want to change the values declare another array and copy the values. Also remember that readdata() implicitly allocates memory for the data array so it is correct to declare the data array as double *data; at the top. If however, you are going to fill in a new array with calculated values you need to declare the array as double newdata[ysize][xsize]; so that space will be allocated.

void **rarrowto**(double x, double y,...);

Adds a line segment to the current path from the current point to the current point plus x, y. The line is terminated by drawing an arrow head oriented in the direction of the line. This command is identical to arrowto except that the displacement is specified relative to the

current point. More than one coordinate point can be specified in the command in which case a series of line segments terminated by arrows connecting the points will be added to the current path. If there is no current point then an error will be generated. The size of the arrow head may be changed with the set(FONTWIDTH,...); command.

void **reset**();

Restores all set able parameters to their default values. Clears the current path and clip path.

void **rlineto**(double x,double y,...);

Adds a line segment to the current path from the current point to the current point plus x, y. This command is identical to lineto except that the displacement is specified relative to the current point. More than one coordinate point can be specified in the command in which case a series of line segments connecting the points will be added to the current path. If there is no current point then an error will be generated.

void **rmoveto**(double x, double y);

Sets the current point to the current point plus x,y. This command is the same as moveto except that a relative move is specified. It also sets the path close point for subsequent use with closepath(). Many path building commands such as curveto and lineto require that a current point exist before calling them.

void **rotate**(double ang);

Rotates the figure about the current origin by the the angle specified. The angle units are degrees and the +ve direction is counterclockwise. Changes in rotation are cumulative.

void **scale**(double xs, double ys);

Changes the scale of the figure by the factors specified for the x and y axes respectively. Changes in scale are cumulative.

void **set**(int option, ...);

All set able parameters can be set using the set command. The first parameter specifies which option to set. The defined constants corresponding to valid options are defined in the header file splot.h. (See Section 4.1.2, 'set' Functions.)

void **showpage**();

Transfers the marked page created in memory by the stroke and fill commands to the physical page. This is automatically done at the end of the file and thus this command is only needed if it is desirable to draw parts of the figure before the end of execution.

void **stroke**();

Marks the page in memory with the current path. The path coordinates are transformed using the current coordinate transformation matrix and the path is fleshed out using the current line width, style and colour in effect at the time of the stroke() command.

void **symbol**(double x, double y, int symbolconst);
or
void **symbol**(int symbolconst);

Plots the symbol chosen by symbolconst at the point x,y. symbolconst must be one of OCIRCLE, OSQUARE, OTRIANGLE, ODIAMOND OSTAR, OARROW, PLUS, CROSS, MULT, CIRCLE, SQUARE, TRIANGLE, DIAMOND, STAR,or ARROW. If the point x,y is omitted the symbol is drawn at the current point as set by a previous moveto();. symbol() is not implicitly stroked so follow with a stroke();. The size of a symbol can be changed using set(FONTWIDTH, val_in_cm); since symbols are just a special font. The size relative to the current FONTWIDTH can be set using set(SYMMULT,mult);

void **text**(double x, double y, char * str,int just);

Adds the text string str to the current path starting at location (x,y). The current font, size and orientation as set using the set() command are used. Super/sub scripts can be entered as ∧2∧ and _2_ respectively. Letters surrounded by '!' or '#' are printed in Greek or italics respectively. For example !m! generates the Greek lower case mu. The special characters "∧_!#$" can be printed by preceding them with \ as in \!. The combination \b back spaces by one character. Symbols may be included in the text string by enclosing them with $. If the starting x,y coordinates are omitted then the current string is positioned one line below the previous string added to the path using text(). The last parameter is the justification and must be one of LEFT, RIGHT or CENTER. This last parameter can be omitted in which case the default value of LEFT will be used. text() is implicitly stroked.

void **ticklabel**(int axis, double v, char *label,...);

Add tick labels to the axes box previously defined. The first parameter is which axis to label either BOTTOM, TOP, LEFT or RIGHT. The following parameters are paired values giving the tick position in data coordinates as established by a prior call to axes_box() or ascale() and the text string to be placed at that location. An error is generated if there is no current axes box. There are several default possibilities. If 'ticklabel();' is called without any parameters then if tick marks have been generated previously using 'tickmarks();' then they will be selectively labelled along the left and bottom axes. If only an axis parameter is given then the corresponding axis tick marks if any will be labelled. If the axis parameter is followed by a list of numbers only they will be assumed to be both the tick position and the desired label. If the numbers are paired with strings in the parameter list then the number will be used as the tick label position and the string will be used as the literal label. The spacing between the tick labels and the axes box can be changed using set(TICKLMARG,...); ticklabel() is implicitly stroked.

void **tickmarks**(int axis, double v,...);

Add tick marks to the axes box previously defined. The first parameter is which axis either BOTTOM , TOP, LEFT, RIGHT ,XAXES or YAXES. The following numbers are the positions were tickmarks are to to be placed. The length of the ticks is set using a set(TICKLENGTH,val) command. The location of the ticks is specified in data coordinates as established by the prior call to axes_box() or ascale(). An error is generated if there is no current axes_box. All the above parameters are optional. The default behaviour is as follows. If all parameters are omitted tick marks will be automatically generated for all axes. If only an axis specifier is given then tick marks will be generated for the corresponding axis or axes. In these cases 'tickmarks();' will try to find reasonable positions for a set of tick marks. If this default behaviour is unacceptable then the actual tick positions can be specified as a list

of values following the axis specifier. If there is only one numeric value it is interpreted as a tick spacing. tickmarks() is implicitly stroked.

void **translate**(double tx, double ty);

Translates the origin of the figure by the x and y distances specified. Translations are cumulative.

void **whereis**(double *x,double *y);

Returns the coordinates of the current point as set by the last moveto(), lineto() etc.

4.1.2 'set' Functions

What follows is a very brief description of all the values set able using set().

set(**AXESCLIP**,val);

If val is ON then the data values are clipped to the limits of the axes box. This is temporarily added to the user set clip limits if any. If val is OFF (the default) then the user specified clip limits from the last clip() call are used. Alternatively the range of data values plotted can be set using set(XRANGE,...); set(YRANGE,...); .

set(**AXESTYPE**,val);

AXESTYPE may be one of the following. LINEAR (the default), LOGX, LOGY, LOGLOG, INVX, INVY, INVINV, INVXLOGY or LOGXINVY. The position of data points and tick-marks are automatically adjusted to account for the axes type.

set(**CURSYMBOL**,sym);

Sets the symbol to use when plotting data with symbols. sym must be one of OCIRCLE, OSQUARE, OTRIANGLE , ODIAMOND OSTAR, OARROW, PLUS, CROSS, MULT, CIRCLE, SQUARE, TRIANGLE, DIAMOND, STAR, or ARROW. The default symbol is CIRCLE. The size of a symbol can be changed using set(FONTWIDTH,val_in_cm); since symbols are just a special font. The size relative to the current FONTWIDTH can be set using set(SYMMULT,mult);.

set(**FILLRULE**,type);

Determines the rule to be used when filling a path. Type must be one of NONZWIND (default) or EVENODD.

set(**FLATNESS**,num);

Sets the maximum allowable error in pixels when converting a curve to a set of straight line segments. Smaller values of num give smother curves but take longer to process. The default value of num is 1.

set(**FONT**,fonttype);

Sets the font type to use for subsequent text written using the text(); routine. font name type must be either SIMPLEX (default) or COMPLEX.

set(**FONTASPECT**,asp);

Sets the ratio of the glyph height to width used for text written using text();. The default value is 2.5.

set(**FONTDIR** ,angle);

Sets the rotation angle with respect to the x axis to use when writing text using text(); The default is 0.

set(**FONTMULT**,factor);

Multiplies the current font size by the given factor.

set(**FONTWIDTH**,wid);

Sets the average width of the characters written using text(). The default value is 0.7 cm. FONTWIDTH applies also to symbols.

set(**LABELMARG**,val);

Adds an additional amount to the margin between the plot and the plot labels generated using the "label();" command. val = 0 is the default.

set(**LINECAP**,type);

Determines how thick lines are to be terminated. The allowed types are BUTTCAP (default), ROUNDCAP and PROJCAP.

set(**LINECOLOUR**,col);

Where col is one of BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN or WHITE. This set the colour to use when the current path is stroked or filled. The default colour is BLACK with the background WHITE.

set(**LINEJOIN**,type);

Determines how thick lines are joined together. The allowed join types are MITERJOIN (default), BEVELJOIN and ROUNDJOIN.

set(**LINESTYLE**,pattern,...);

Sets the line style to use for the path when it is stroked. The constant LINESTYLE is followed by a list of floating point values that define the pattern. The values are interpreted as the length along the path that the line is visible followed by the length that it is invisible as an alternating sequence wrapping back to the beginning when the pattern reaches the end. For example a pattern of 1.0,1.0 implies on for 1 cm followed by off for 1 cm. A pattern of 1.0,0.5 is on for 1 cm followed by off for 0.5 cm. The predefined patterns are: SOLID 0 (default) DASHED 1.0,0.5 DOTTED 0.2,0.2 and DOTDASH 1.0,0.5,0.2,0.5. There must always be an even number of comma separated values in a pattern definition.

set(**LINEWIDTH**,width);

Sets the LINEWIDTH to width cm (default 0.05 cm). This line width is used when the current path is stroked.

set(**MITERLIMIT**,maxratio);

Sets the maximum length of spikes formed by miter joining two lines at an acute angle. If the ratio of the length of the spike to the width exceeds the value of maxratio then a BEVELJOIN is done instead. The default value is 10.0.

set(**PAGEROT**,flag);

Selects landscape orientation if flag is ON. Default is portrait.

set(**PATTOFF**,offset);

Sets the offset into the current LINESTYLE pattern. Can be used to adjust the starting point of a pattern for aesthetic reasons.

set(**PLOTTYPE**,type);

Sets the current plot type. type must be one of LINES (default), SYMBOLS or SYM_LINES. LINES connects data points with line segments while SYMBOLS causes the current symbol to be drawn at each data point. SYM_LINES does both. The size of a symbol can be changed using set(FONTWIDTH, val_in_cm); since symbols are just a special font. The size relative to the current FONTWIDTH can be set using set(SYMMULT,mult);.

set(**SCALEALL**,val);

If val is ON then the XSHIFT, YSHIFT, XMULT, YMULT values are applied to all coordinates (i.e. in lineto, moveto etc.). The default is OFF in which case only data plotted using plotdata or drawdata is affected by these values.

set(**SCRIPTSCALE**,val);

Sets the relative height of a super/sub script as compared to ordinary text. The default value is 0.5.

set(**SCRIPTSHIFT**,val);

Sets the distance that a super/sub script is shifted above/below ordinary text. The value is specified as a fraction of the ordinary text height. The default value is 0.7.

set(**SYMMULT**,val)

Sets the symbol size multiplier used when drawing symbols. The default value is 1.0. The actual symbol size is determined the current FONTWIDTH multiplied by the curent SYMMULt value.

set(**TICKLENGTH**,len);

Sets the length of axis tick marks to be used. The default value is 0.3 cm.

set(**TICKLMARG**,val);

This value corresponds to the margin used between the axes box and any tick labels generated using the "ticklabel();" command. val = 0 is the default.

set(**XMULT**,xmul);

Causes each x value to be multiplied by xmul before being plotted.

set(**XRANGE**,xmin,ymin);

data plotted using drawdata is constrained to have x values between xmin and xmax. There is also a corresponding YRANGE set option. The default is no constraints. The values should be specified in user coordinates i.e. those established by the current axes_box() or ascale().

set(**XSHIFT**,xshft);

Causes the value xshft to be added to all x values before plotting.

set(**YSHIFT**,yshft);

Causes the value yshft to be added to all y values before plotting.

set(**YMULT**,ymul);

Causes each y value to be multiplied by ymul before being plotted.

4.2 Standard C Function Library Reference

int **abs**(int i);

Returns the absolute value of i;

double **acos**(double x);

Returns the arc cosine of the value x. x must be between -1 and 1. Returns a value between 0 and π.

double **asin**(double x);

Returns the arc sine of the value x. x must be between -1 and 1. Returns a value between $-\pi/2$ and $\pi/2$.

double **atan**(double x);

Returns the arc tangent of the value x. Returns a value between $-\pi/2$ and $\pi/2$.

double **atan2**(double y,double x);

Returns the arc tangent of the value y/x. Returns a value between $-\pi$ and π.

double **atof**(char *str);

Converts a string to a double. The string must contain only digits and 'e', 'E', '.', '-' and '+' .

int **atoi**(char *str);

Converts a string to an integer. The string must contain only digits.

double **ceil**(double x);

Rounds up x to nearest integer value.

double **cos**(double x);

Returns the cosine of x. x is specified in degrees.

void **exit**(int status);

Terminates the execution of the program. If the status is 0 then it will be considered a normal exit otherwise an error induced exit.

double **exp**(double x);

Calculates the exponential function exp(x).

double **fabs**(double x);

Returns the absolute value of x. It is like abs() but works with floating point numbers rather than integers.

double **floor**(double x);

Rounds down x to the nearest integer.

double **fmod**(double x,double y);

Returns the remainder of x/y.

void **free**(char *ptr);

Frees the block of memory pointed to by ptr. The memory must have been previously allocated using malloc().

double **log**(double x);

Returns the natural log of x.

double **log10**(double x);

Returns the log base 10 of x.

char ***malloc**(int size);

Allocates a block of memory of size bytes and returns a pointer to the block. malloc returns NULL if there is insufficient free memory.

double **pow**(double x,double y);

Calculates x to the power y.

int **puts**(char *str);

This routine writes the string str to the output file and starts a new line.

int **printf**(char *format,...);

Prints the formatted data to the output file. The format string specifies the type and number of values to print. Some common examples include:

printf("i = %d",i); prints the integer value i.

printf("x = %g",x); prints the floating point value x.

printf("text = %s",str); prints the string str.

Multiple values can be printed as in printf("%d %d %g %s",i,j,x,str);

The format specifiers can also include field width information and justification etc. Consult a standard C text for more details.

void **print**(v,...);

Prints the value v which can be of any scalar type. i.e int, char, float, double or a pointer. This is not a function found in the standard C library.

double **sin**(double x);

Returns the sine of x. x must be specified in degrees.

int **sizeof**(t);

Returns the number of bytes required to store the value of type t.

double **sqrt**(double x);

Calculates the square root of x. x must be a positive number.

int **sprintf**(char str,char *format,...);

Prints the formatted data to the string str. The format string specifies the type and number of values to print. Some common examples include:

printf("i = %d",i); prints the integer value i.

printf("x = %g",x); prints the floating point value x.

printf("text = %s",str); prints the string str.

Multiple values can be printed as in

printf("%d %d %g %s",i,j,x,str);

The format specifiers can also include field width information and justification etc. Consult a standard C text for more details.

void **strcat**(char *dest, char *source);

Concatenates the string source to the string dest.

void **strcpy**(char *dest, char *source);

Copies the string source to the string dest.

int **strlen**(char *str);

Returns the length of the string str.

double **tan**(double x);

Calculates the value of the tangent of x. x should be specified in degrees.

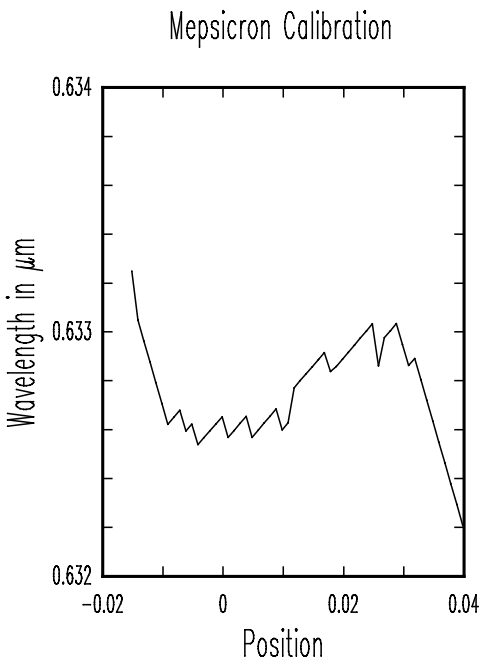
Chapter 5

Sample Programs

This chapter contains printouts of several example drawing or plotting files with the corresponding printed figure that the file generates. These for the most part are contributed files of actual figures used in publications.

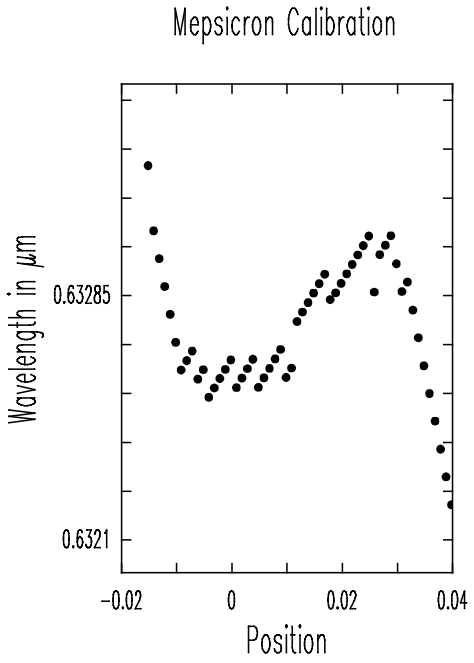
5.1 simple.spt

```
/* demo of simplest possible data */
/* plotting using all the defaults */

#include <splot.h>
double *data;
main()
   {
   readdata("demo\data1.dat",data);
   plotdata(data);
   label(LOWER,"Position");
   label(LEFT,"Wavelength in !m!m");
   text(6.20,22.98,"Mepsicron Calibration");
   }
```



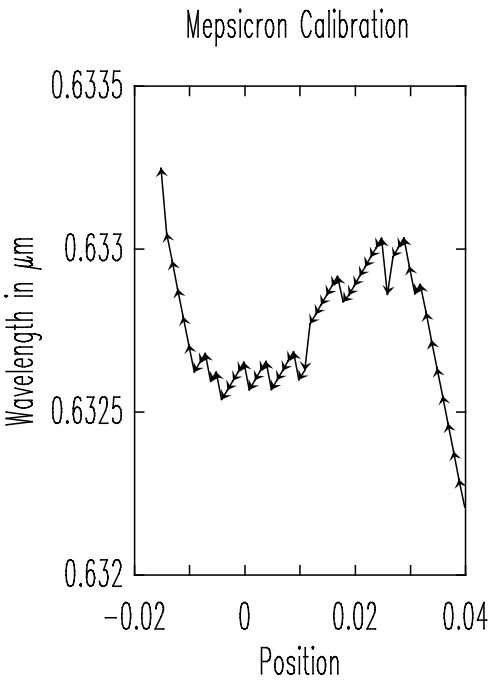
5.2 simple2.spt

```
/* demo of simple data plotting but*/
/* over riding some of the defaults */

#include <splot.h>
double *data;
main()
   {
   readdata("demo\data1.dat",data);
   set(PLOTTYPE,SYMBOLS);
   set(AXESCLIP,ON);
   axes_box(11,16,-0.02,0.632,0.04,0.6335);
   plotdata(data);
   label(LOWER,"Position");
   label(LEFT,"Wavelength in !m!m");
   text(6.20,22.98,"Mepsicron Calibration");
   }
```



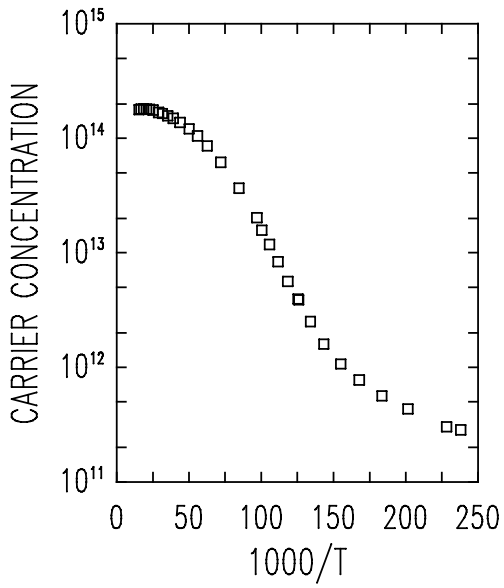
5.3 simple3.spt

```
/* demo of simple data plotting but*/
/* over riding some more of the defaults */

#include <splot.h>
double *data;
main()
   {
   readdata("demo\data1.dat",data);
   set(PLOTTYPE,SYM_LINES);
   set(CURSYMBOL,ARROW);
   set(AXESCLIP,ON);
   axes_box(11,16,-0.02,0.632,0.04,0.6335);
   tickmarks(YAXES,0.632,0.6325,0.633,0.6335);
   tickmarks(XAXES);
   ticklabel();
   label(LOWER,"Position");
   label(LEFT,"Wavelength in !m!m");
   text(6.20,22.98,"Mepsicron Calibration");
   set(FONTMULT,2);
   plotdata(data);
   }
```



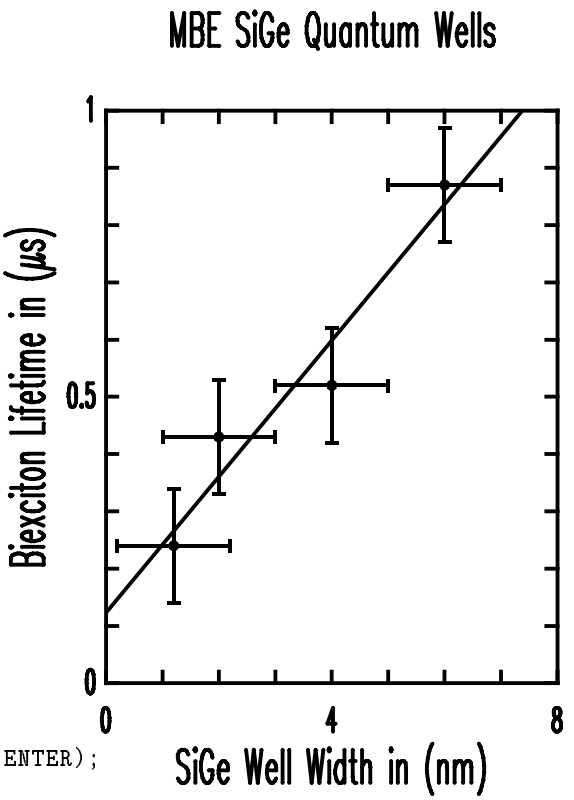
5.4 inverax.spt

```
/* This file illustrates the use of inverse and log axes */
\index{axes, log example}
\index{axes, inverse example}

#include <splot.h>
double *data;
main()
   {
   set(LINEWIDTH,0.06);
   set(CURSYMBOL,OSQUARE);
   set(AXESCLIP,ON);
   set(FONTASPECT,1.5);
   set(AXESTYPE,INVXLOGY);
   axes_box(12,15,1e20,0.001,4,10);
   set(FONTWIDTH,0.8);
   tickmarks();
   ticklabel(BOTTOM,1e+20,"0",20,"50",10,"100",6.6666,"150"
             ,5,"200",4,"250");
   ticklabel(LEFT,1e-3,"10^11^",1e-2,"10^12^",1e-1,"10^13^",1,"10^14^",10,"10^15^");
   set(XMULT,1);
   set(FONTWIDTH,0.9);
   label(BOTTOM,"1000/T");
   label(LEFT,"CARRIER CONCENTRATION");
   set(PLOTTYPE,SYMBOLS);
   readdata("demo\tb4_2_2.dat",data);
   plotdata(data);
   }
```

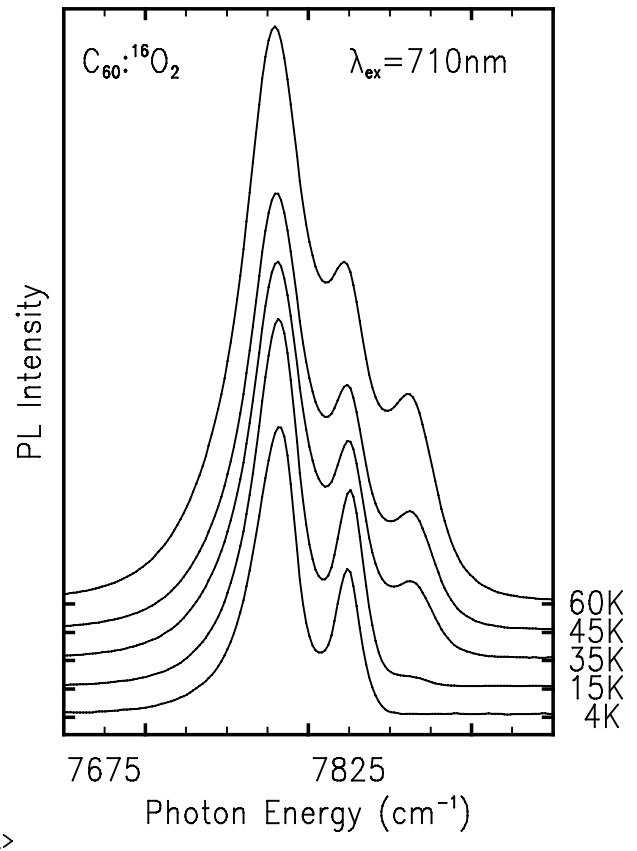


5.5 errorbar.spt

```
#include <splot.h>

/* This example file illustrates the use of errorbars and straight */
/* line fits to data. It also uses a small internal data array rather */
/* than reading data in from a file */

double data[4][4] = {
   1.2, 0.24, 0.1, 1,
   2.0, 0.43, 0.1, 1,
   4.0, 0.52, 0.1, 1,
   6.0, 0.87, 0.1, 1};

double yint,slope;


main()
   {
   set(LINEWIDTH,0.1);
   set(PLOTTYPE,SYMBOLS);
   axes_box(12,15,0,0,8,1.0);
   tickmarks();
   set(FONTWIDTH,0.5);
   ticklabel();
   set(FONTWIDTH,0.7);

   label(BOTTOM,"SiGe Well Width in (nm)");
   label(LEFT,"Biexciton Lifetime in (!m!s)");

   text(10.03,22.56,"MBE SiGe Quantum Wells",CENTER);

   /* draw the data points  */
   drawdata(data,0,1);

   /* put in the error bars for the x values */
   errorbars(XVALS,data,0,1,3);
   /* put in the error bars for the y values */
   errorbars(YVALS,data,0,1,2);

   /* fit the best line to the data and return the slope and intercept */
   fitline(data,0,1,&yint,&slope);
   print(slope);
   print(yint);
   }
```



5.6 manyspec.spt

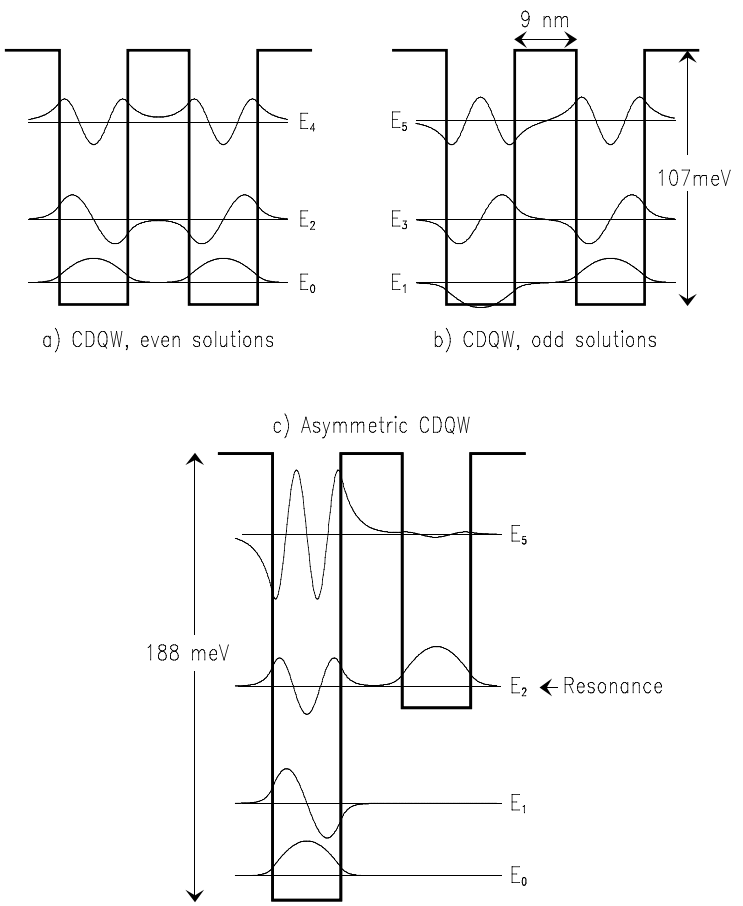



```
#include <splot.h>
/* Created by Scott Wilson 02/05/93 */
/* this splot file generates an xy
   plot of 5 data sets. Each curve
   has the same x scale and y scale
   but have been offset from each
   other in the y direction by an
   equal amount
*/
/* define the data arrays */
double *pldata1, *pldata2, *pldata3, *pldata4, *pldata5;
main()
  {

   /* define a tic length  */
   /* major and minor tics */

   double majortic,minortic;
   majortic=0.25;
   minortic=0.17;

   /* set up some default values */
```

```
set(FONTASPECT,1.2);
set(FONTWIDTH,0.8);
set(LINECOLOUR,BLACK);
set(AXESCLIP,ON);
set(LINEWIDTH,0.09);
set(PLOTTYPE,LINES);

/* load in the data */

readdata("demo\4kc.dat",pldata1);
readdata("demo\15kc.dat",pldata2);
readdata("demo\35kc.dat",pldata3);
readdata("demo\45kc.dat",pldata4);
readdata("demo\60kc.dat",pldata5);

/* define an axis box */

axes_box(13,19,7650,-3000,7950,125000);

/* set up some major tics*/

set(TICKLENGTH,majortic);

/* for the x-axis    */

tickmarks(XAXES,7700,7800,7900);

/* for the y-axis    */
/* note only denoting the zero point
for each curve */

tickmarks(YAXES,0,5000,10000,15000,20000);

/* set up some minor tics */

set(LINEWIDTH,0.05);
set(TICKLENGTH,minortic);

/* for the x axis */

tickmarks(XAXES,7675,7725,7750,7775,7825,7850,7875,7925);

/* place some ticklabels on the axes */

ticklabel(BOTTOM);
ticklabel(RIGHT,0," 4K",5000,"15K",10000,"35K",15000,"45K",20000,"60K");

 /* place some labels on the axes */

label(BOTTOM,"Photon Energy (cm^-1^)");
label(LEFT,"PL Intensity");
```

```
/* place some comments in the figure */

text(3.98,21.11,"C_60_:^16^O_2_");
text(11.10,21.11,"!l!_ex_=710nm");

/* put the data into the picture */

drawdata(pldata1);

set(YSHIFT,5000);
drawdata(pldata2);
set(YSHIFT,10000);
drawdata(pldata3);
set(YSHIFT,15000);
drawdata(pldata4);
set(YSHIFT,20000);
drawdata(pldata5);
}
```

5.7 invisax.spt




```
/* This example shows how invisible axes can be used to establish */
/* an alternate coordinate system and still use the drawdata function */
/* to draw the curves */

/*draws quantum wells and wave functions */
#include <splot.h>
double *data1;
double *data2;
double *data3;
double *data4;
double *data5;
```

```
main()
   {
   set(LINECOLOUR,INVIS);
   set(FONTWIDTH,0.6);
   set(FONTASPECT,1.4);
   set(LINEWIDTH,0.02);
   /* read in data generated by Mathematica */
   readdata("demo\cdqw_wfc.dat",data1);
   readdata("demo\cdqw_wfc.da2",data2);
   readdata("demo\cdqw_wfc.da3",data3);
   readdata("demo\cdqw_wfc.da4",data4);
   readdata("demo\cdqw_wfc.da5",data5);
   /* drawing turned out too big so reduce the overall size */
   scale(0.7,0.65);
   gsave();
   axes_box(8,15,-185,0.0,185,188.36);
   set(LINECOLOUR,BLACK);
   translate(-1.25,22.5);
   /* even solns to cdqw */
   /*   norm = 1.0/0.00009134 * 60.0; */
   set(YMULT,656886.);
   set(YSHIFT,9.13863);
   drawdata(data1);
   /*   norm = 1/.0002838 * 60.0;  */
   set(YMULT,211416.);
   set(YSHIFT,35.8858);
   drawdata(data1,0,3);
   /*   norm = 1/0.008109 * 60.0;*/
   set(YMULT,7400.);
   set(YSHIFT,77.0532);
   drawdata(data3);
   /* call subroutine to draw quantum well. Also sets coordinate scale to match*/
   s_qw_box();
   /* energy eigenvalues */
   moveto(-190,9.13863);
   lineto(190,9.13863);
   moveto(-190,35.8858);
   lineto(190,35.8858);
   moveto(-190,77.0532);
   lineto(190,77.0532);
   stroke();
   text(205,9.13863,"E_0_");
   text(205,35.8858,"E_2_");
   text(205,77.0532,"E_4_");
   text(0,-15,"a) CDQW, even solutions",CENTER);
   grestore();
   /* odd solns to cdqw */
   set(LINECOLOUR,INVIS);
   gsave();
   axes_box(8,15,-185,0.0,185,188.36);
   set(LINECOLOUR,BLACK);
   translate(11,22.5);
```

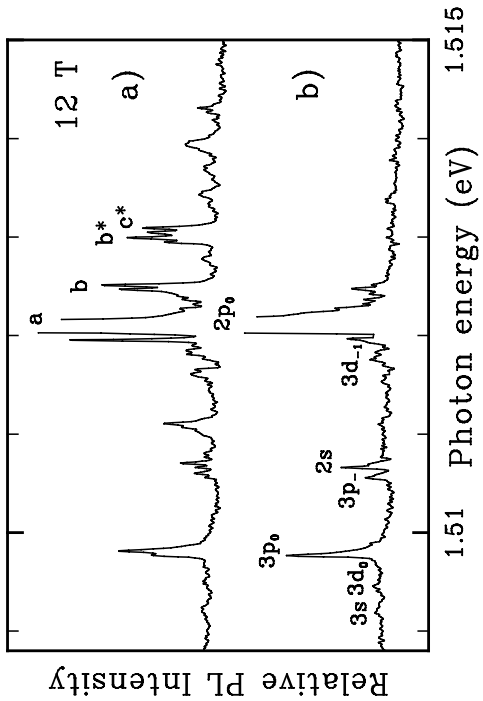
```
set(YMULT,656886.);
set(YSHIFT,9.14186);
drawdata(data1,0,2);
set(YMULT,211416.);
set(YSHIFT,35.9177);
drawdata(data1,0,4);
set(YMULT,7400.);
set(YSHIFT,77.4394);
drawdata(data3,0,2);
s_qw_box();
/* energy eigenvalues */
moveto(-190,9.14186);
lineto(190,9.14186);
moveto(-190,35.9177);
lineto(190,35.9177);
moveto(-190,77.4394);
lineto(190,77.4394);
stroke();
text(-227,9.14186,"E_1_");
text(-227,35.9177,"E_3_");
text(-227,77.4394,"E_5_");
text(217.,53.,"107meV",CENTER);
text(0,120,"9 nm",CENTER);
text(0,-15,"b) CDQW, odd solutions",CENTER);
grestore();
/*asymmetric coupled wells */
set(LINECOLOUR,INVIS);
gsave();
axes_box(8,15,-185,0.0,185,188.36);
set(LINECOLOUR,BLACK);
translate(5.5,2.5);
set(YMULT,1000.);
set(YSHIFT,90.23);
set(XSHIFT,-95.);
drawdata(data4,0,2);
a_qw_box();
moveto(-195,90.23);
lineto(190,90.23);
stroke();
text(215,90.23,"E_2_",CENTER);
text(280,90.23,"Resonance");
text(-270.,104.,"188 meV",CENTER);
grestore();
set(LINECOLOUR,INVIS);
gsave();
axes_box(8,15,-185,0.0,185,188.36);
set(LINECOLOUR,BLACK);
translate(5.5,2.5);
set(XSHIFT,-95.);
set(YMULT,-80.0);
set(YSHIFT,10.29);
drawdata(data5);
```

```
   set(YMULT,-80.0);
   set(YSHIFT,40.81);
   drawdata(data5,0,2);
   set(YMULT,-275.0);
   set(YSHIFT,154.16);
   drawdata(data5,0,3);
   set(YSHIFT,116.97);
   set(YMULT,500.0);
   a_qw_box();
   moveto(-190,10.29);
   lineto(190,10.29);
   moveto(-190,40.81);
   lineto(190,40.81);
   moveto(-190,154.16);
   lineto(190,154.16);
   stroke();
   text(215,10.29,"E_0_",CENTER);
   text(215,40.81,"E_1_",CENTER);
   text(215,154.16,"E_5_",CENTER);
   text(0,200,"c) Asymmetric CDQW",CENTER);
   grestore();
   set(LINECOLOUR,BLACK);
   set(FONTWIDTH,1.);
   set(FONTASPECT,2.5);
   moveto(20.91,36.9);
   arrowto(20.0,36.9);
   moveto(20.91,36.9);
   arrowto(22.0,36.9);
   moveto(25.5,33.02);
   arrowto(25.5,36.53);
   moveto(25.5,31.79);
   arrowto(25.5,27.95);
   moveto(9.9,16.90);
   arrowto(9.9,23.02);
   moveto(9.9,15.70);
   arrowto(9.9,7.93);
   moveto(21.4,15.14);
   arrowto(20.8,15.14);
   stroke();
   }

int s_qw_box()
   {
   set(LINEWIDTH,0.06);
   cmatch();
   /* draw quantum well box */
   moveto(-225.,107.27);
   lineto(-145.,107.27);
   lineto(-145.,0.);
   lineto(-45.,0.);
   lineto(-45.,107.27);
   lineto(45.,107.27);
```

```
   lineto(45.,0.);
   lineto(145.,0.);
   lineto(145.,107.27);
   lineto(225.,107.27);
   stroke();
   set(LINEWIDTH,0.02);
   }

int a_qw_box()
   {
   set(LINEWIDTH,0.06);
   cmatch();
   /* draw quantum well box */
   moveto(-225.,188.36);
   lineto(-145.,188.36);
   lineto(-145.,0.);
   lineto(-45.,0.);
   lineto(-45.,188.36);
   lineto(45.,188.36);
   /* 188.36-107.27  */
   lineto(45.,81.09);
   lineto(145.,81.09);
   lineto(145.,188.36);
   lineto(225,188.36);
   stroke();
   set(LINEWIDTH,0.02);
   }
```

5.8 landscap.spt

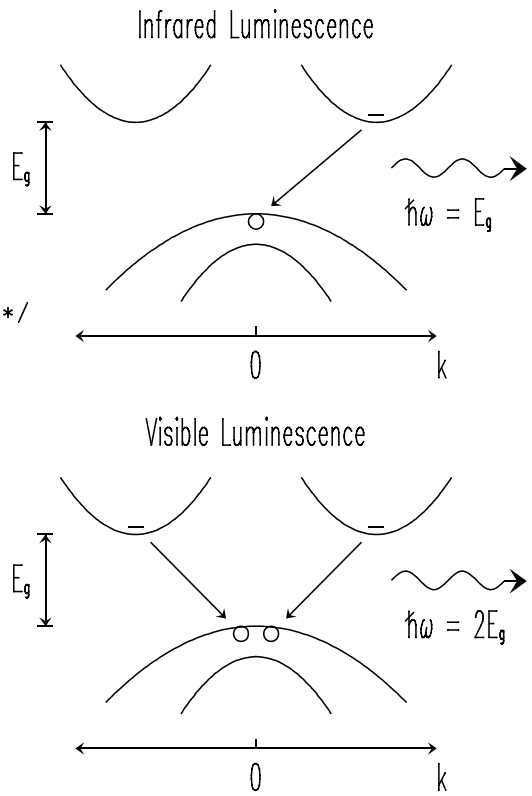
```
#include <splot.h>
double *data1,*data2;
main()
   {

   /* select landscape orientation */
   set(PAGE_ROT,ON);
   set(LINEWIDTH,0.09);
   axes_box(20,14,1.5088,0,1.515,1.03e8,3,3);

   set(TICKLENGTH,0.5);
   tickmarks(XAXES,1.510,1.515);

   set(LINEWIDTH,0.06);
   set(FONTWIDTH,0.8);
   /* use short squat letters for labels */
   set(FONTASPECT,1.1);
   set(FONT,COMPLEX);
   ticklabel(LOWER,1.510,1.515);
   set(FONTWIDTH,1.0);
   label(LEFT,"Relative PL Intensity");
   text(9.00,1.90,"Photon energy (eV)");
   text(20.00,15.00,"12 T");
   text(21.00,13.00,"a)");
   text(21.00,7.00,"b)");
   set(FONTWIDTH,0.70);
   text(13.47,9.84,"2p_0_");
   text(5.70,8.35,"3p_0_");
   text(8.80,6.46,"2s");
   text(4.80,5.40,"3d_0_");
   text(3.82,5.30,"3s");
   text(7.65,5.70,"3p_-_");
   text(11.65,5.60,"3d_-1_");
   set(FONTWIDTH,0.75);
   text(13.65,16.10,"a");
   text(14.70,14.60,"b");
   text(16.25,13.75,"b*");
   text(16.80,13.10,"c*");

   set(LINEWIDTH,0.05);
   set(TICKLENGTH,0.35);
   tickmarks(XAXES,1.509,1.511,1.512,1.513,1.514,1.516);

   /* constrain plot to axes box */
   set(AXESCLIP,ON);

   /* change x axis units to eV */
   set(XMULT,0.1239842e-3);

   /* plot first spectrum */
```



```
set(XSHIFT,0);
set(YSHIFT,47e6);
set(YMULT,300);
readdata("demo\fe21m23a.dat",data1);
set(YRANGE,0,99e6);
drawdata(data1,0,1);

/* plot second spectrum shifted up */
set(YSHIFT, 7e6);
set(YMULT,150);
readdata("demo\fe22v31a.dat",data2);
set(YRANGE,0,45e6);
drawdata(data2,0,1);
}
```

5.9 intcalc.spt

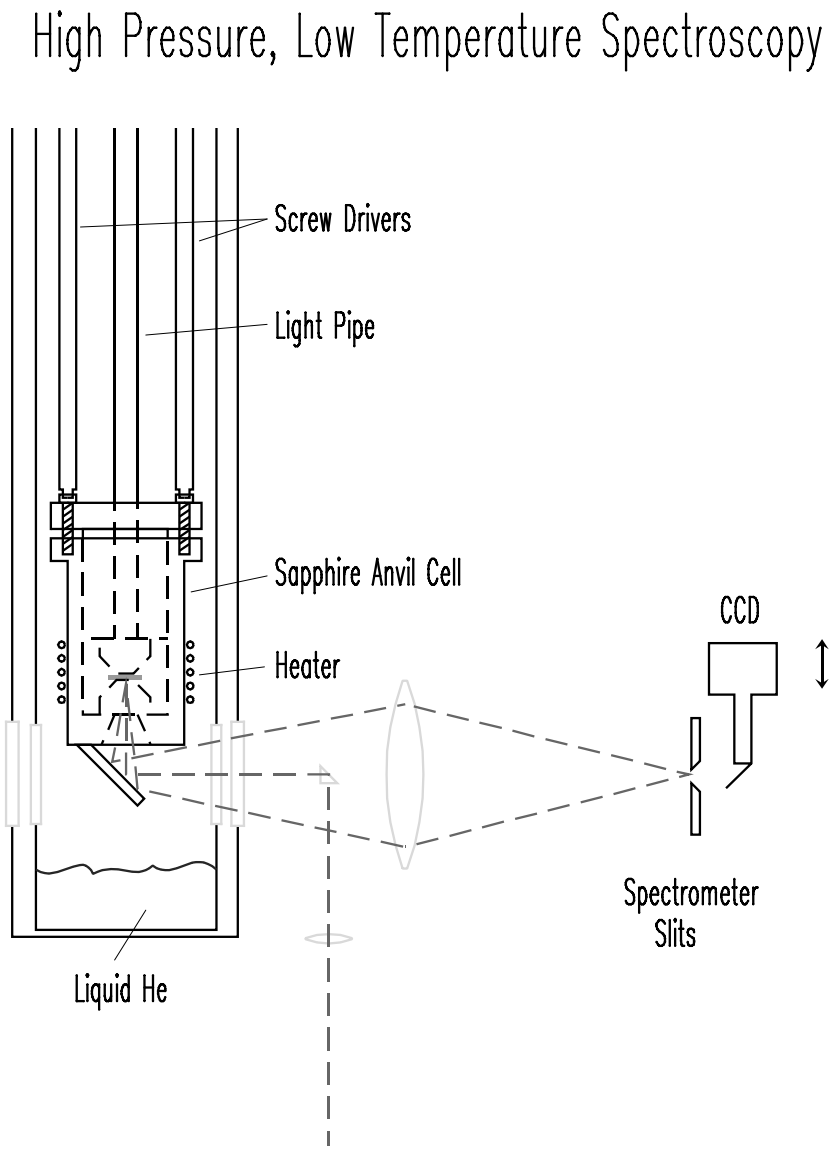
```
#include <splot.h>

/* This example file illustrates some of the internal math capabilities */
/* In this case  x,y data points for several parabolas are calculated */
/* and then drawn. Also the photon uses the sine function to draw a wave. */
/* For the photon the data is not stored in an array, rather the calculated */
/* values are added to offsets given as parameters to the photon subroutine */

/* make space for the to be generated data */
double pdat[101][2];

main()
   {
   double x0,y0;

   x0 = 10;
   y0 = 5.5;

   /* draw parabolas at x0,y0  to represent bands*/
   siband(x0,y0);

   /* draw circles to represent holes in the valence band */
   arc(x0 - 0.5,y0 - 0.25,0.25,0,360);
   stroke();
   arc(x0 + 0.5,y0 - 0.25,0.25,0,360);
   stroke();

   /* draw electrons in the conduction band */
   moveto(x0 + 4.25,y0 + 3.25);
   rlineto(-0.5,0);
   moveto(x0 - 4.25,y0 + 3.25);
   rlineto(0.5,0);
   stroke();

   /* draw arrows to indicate optical transitions */
   moveto(x0 + 3.5,y0 + 2.75);
   arrowto(x0 + 1,y0 + 0.25);
   moveto(x0 - 3.5,y0 + 2.75);
   arrowto(x0 - 1,y0 + 0.25);

   /* label the photon */
   text(x0 + 5, y0,"h!w! = 2E_g_");
   /* but bar through h */
   moveto(x0 + 4.95,y0 + 0.16);
   rlineto(0.3,0.3);
   stroke();

   /* draw bands again but higher up */
   x0 = 10;
   y0 = 19;
```



```
   siband(x0,y0);

   /* draw single hole */
   arc(x0,y0 - 0.25,0.25,0,360);
   stroke();

   /* draw single electron */
   moveto(x0 + 4.25,y0 + 3.25);
   rlineto(-0.5,0);
   stroke();
   moveto(x0 + 3.5,y0 + 2.75);
   arrowto(x0 + 0.5,y0 + 0.25);

   /* label photon */
   text(x0 + 5, y0,"h!w! = E_g_");
   moveto(x0 + 4.95,y0 + 0.16);
   rlineto(0.3,0.3);
   stroke();

   /* add titles */
   text(10.0,25.15,"Infrared Luminescence",CENTER);
   text(10.0,11.8,"Visible Luminescence",CENTER);
   }

int siband(double x0,y0)
   {

   /* draws strained SiGe band structure */
   /* as a set of parabolas */
   parab(x0,y0,0.1,-0.1);
   parab(x0,y0 - 1.0,0.05,-0.3);
   parab(x0 - 4.0,y0 + 3.0,0.05,0.3);
   parab(x0 + 4.0,y0 + 3.0,0.05,0.3);

   /* draw axes */
   moveto(x0,y0 - 4.0);
   rlineto(0,0.3);
   moveto(x0,y0 - 4.0);
   rarrowto(6,0);
   moveto(x0,y0 - 4.0);
   rarrowto(-6,0);
   stroke();
   moveto(x0 - 7, y0);
   rarrowto(0,3);
   rarrowto(0,-3);
   moveto(x0 - 7.25,y0);
   rlineto(0.5,0);
   moveto(x0 - 7.25,y0 + 3);
   rlineto(0.5,0);
   stroke();

   /* draw a photon */
```

```
   photon(x0 + 4.5,y0 + 1.5,0.3,0.3);

   /* a k vector labels */
   text(x0,y0 - 5,"0",CENTER);
   text(x0 + 6,y0 - 5,"k");
   text(x0 - 7.5,y0 + 1.5,"E_g_",RIGHT);
   }

int parab(double x0,double y0,double dx,double a)
   {
   int i;
   double x;
   /* draws a parabola by generating */
   /* a data array first */
   for (i = 0; i <= 50;i++)
      {
      x = i * dx;
      pdat[50 + i][1] = a * x * x + y0;
      pdat[50 - i][1] = a * x * x + y0;
      pdat[50 + i][0] = x0 + x;
      pdat[50 - i][0] = x0 - x;
      }
   moveto(pdat[0][0],pdat[0][1]);
   for (i = 0;i <= 100;i++)
      {
      lineto(pdat[i][0],pdat[i][1]);
      }
   stroke();
   }

int photon(double x0,double y0,double xscale,double yscale)
   {
   double x,y;
   int i;

   /* draw a photon using the sine function */
   moveto(x0,y0);
   for (i = 0;i < 50;i++)
      {
      x = i * 0.255;
      y = sin(x);
      lineto(x * xscale + x0,y * yscale + y0);
      }
   set(FONTMULT,2);
   rarrowto(2.5 * xscale,0);
   set(FONTMULT,0.5);
   stroke();
   }
```

5.10 linedraw.spt




```
/* This file illustrates how to generate a line drawing of apparatus */
/* It makes heavy use of gsave() and grestore() and axes transformations*/
/* to draw elements repeatedly in different locations and/or orientations */
/* This is just one way to do this. */

#include <splot.h>

main()
   {
   /* it turned out too big in the end so scale it back some */
   scale(0.75,0.75);
```

```
/* draw Dewar */
tail();
gsave();
/* draw a slightly bigger tail around the first */
scale(1.25,1.05);
translate(-1.05,-0.5);
tail();
grestore();

/* draw lenses */
newpath();
set(LINECOLOUR,BROWN);
gsave();
translate(12.79,10.85);
rotate(90);
scale(2,2);
lens();
grestore();
gsave();
translate(11.09,6.21);
scale(0.5,0.5);
lens();
grestore();

/* draw the prism */
gsave();
translate(10.84,10.6);
prism();
grestore();

/* draw helium level */
set(LINECOLOUR,BLUE);
moveto(2.44,8.08);
curveto(3.13,7.66,3.82,8.68,4.13,7.96);
curveto(4.89,8.38,5.32,7.72,5.89,8.20);
curveto(6.58,7.72,7.08,8.74,7.77,8.08);
stroke();

/* draw 45 degree fold mirror */
set(LINECOLOUR,BLACK);
moveto(4.07,11.72);
lineto(5.64,10.15,5.44,9.95,3.64,11.72);
closepath();

/* draw diamnond anvil cell */
moveto(3.38,11.72);
lineto(6.82,11.72,6.82,17.07,7.33,17.07,7.33,17.73,2.88,17.73,
       2.88,17.07,3.38,17.07);
closepath();
moveto(3.82,17.72);
lineto(3.82,18,6.33,18,6.33,17.72);
stroke();
```

```
/* draw plate and screws */
moveto(2.88,18.0);
lineto(7.33,18.0,7.33,18.76,2.88,18.76);
closepath();
stroke();
gsave();
translate(3.38,18.76);
screw(1.5);
translate(3.45,0);
screw(1.5);
grestore();

/* draw heater */
gsave();
translate(7.0,13.03);
heater();
translate(-3.8,0);
heater();
grestore();

/* draw slits */
gsave();
translate(21.8,11.0);
slit();
grestore();
gsave();
scale(1,-1);
translate(21.8,-10.6);
slit();
grestore();

/* draw periscope */
moveto(22.82,10.45);
lineto(23.57,11.18);
rlineto(0,2, 0.75,0, 0,1.5, -2,0, 0,-1.5, 0.75,0, 0,-2, 0.5,0);
stroke();

/* draw arrows */
moveto(25.66,14.07);
arrowto(25.66,14.79);
moveto(25.66,14.07);
arrowto(25.66,13.35);
stroke();

/* draw screw drivers and light pipe */
moveto(4.76,18.76);
lineto(4.76,30.0);
moveto(5.45,18.76);
lineto(5.45,30.0);
stroke();
gsave();
```

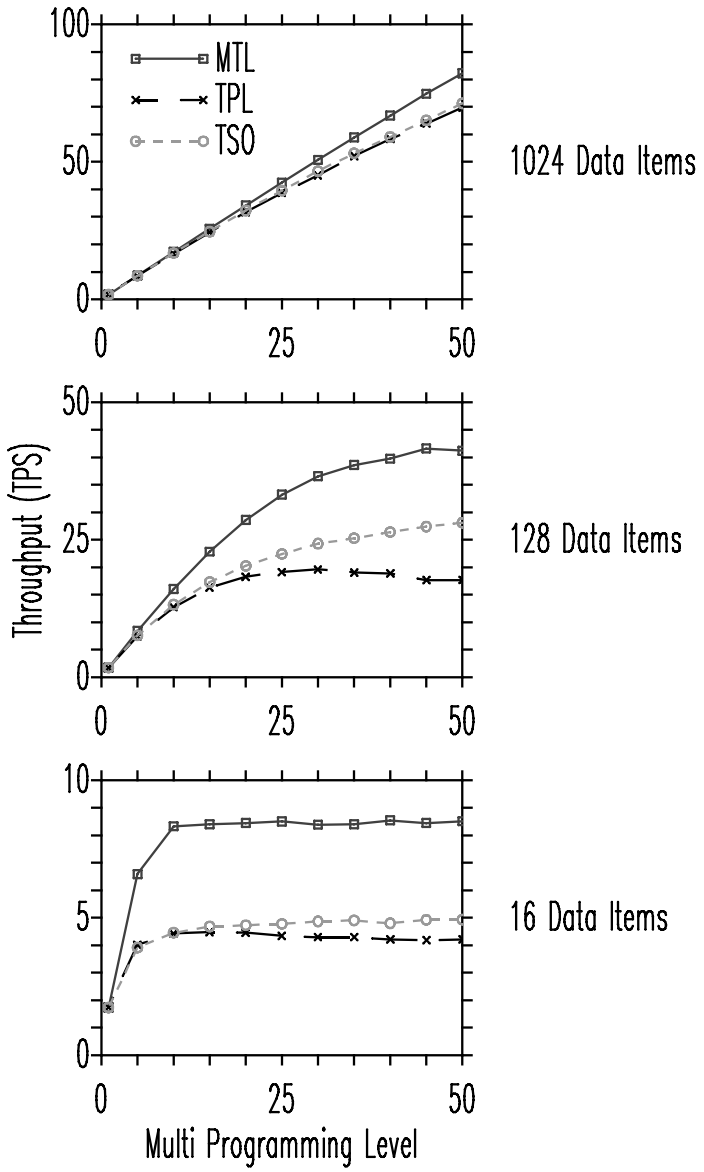
```
translate(3.38,18.9);
driver();
translate(3.45,0);
driver();
grestore();

/* white out some of the lines at */
/* the top that turned out too long */
white_box(1.17,29.67,8.61,31.20);
newpath();

/* draw copper gasket */
gsave();
set(LINECOLOUR,GREEN);
set(LINEWIDTH,0.10);
moveto(4.57,13.7);
lineto(5.57,13.7);
stroke();
grestore();

/* draw dashed lines in diamond cell */
set(LINESTYLE,0.5,0.25);
moveto(3.82,17.72);
lineto(3.82,12.60,6.33,12.60,6.33,17.72);
moveto(3.82,14.8);
lineto(6.33,14.8);
moveto(4.76,14.8);
lineto(4.76,18.76);
moveto(5.45,14.8);
lineto(5.45,18.76);
moveto(4.76,12.6);
lineto(4.38,11.72);
moveto(5.45,12.6);
lineto(5.83,11.72);
stroke();

/* draw the sapphire anvils */
gsave();
translate(4.32,12.6);
anvil();
translate(1.5,2.2);
rotate(180);
anvil();
grestore();
stroke();

/* draw light paths */
set(LINECOLOUR,MAGENTA);
moveto(5.1,13.57);
lineto(5.1,10.85,11.09,10.85,11.09,0.06);
moveto(13.35,12.9);
lineto(4.7,11.22,5.1,13.51,5.45,10.43,13.35,8.74);
```

```
   moveto(13.35,12.9);
   lineto(21.73,10.85,13.35,8.74);
   stroke();

   /* add comments */
   set(LINECOLOUR,BLACK);
   set(LINESTYLE,0);
   set(FONTWIDTH,1.0);
   text(2.33,32.29,"High Pressure, Low Temperature Spectroscopy");
   set(FONTWIDTH,0.6);
   text(3.57,4.58,"Liquid He");
   text(22.65,15.60,"CCD");
   text(19.81,7.39,"Spectrometer");
              text("   Slits");
   text(9.5,16.7,"Sapphire Anvil Cell");
   text(9.5,14,"Heater");
   text(9.5,23.88,"Light Pipe");
   text(9.5,27,"Screw Drivers");

   /* draw lines to labelled parts */
   set(LINEWIDTH,0.01);
   moveto(4.76,5.46);
   lineto(5.68,6.91);
   moveto(9.19,13.99);
   lineto(7.27,13.75);
   moveto(9.27,16.64);
   lineto(7.02,16.16);
   moveto(9.27,23.96);
   lineto(5.68,23.64);
   moveto(9.27,27.02);
   lineto(7.27,26.38);
   moveto(9.27,27.02);
   lineto(3.76,26.78);
   stroke();
   }

int anvil()
   {
   /* draw a sapphire anvil */
   newpath();
   moveto(0,0);
   rlineto(0,0.5,0.5,0.5,0.5,0,0.5,-0.5,0,-0.5);
   stroke();
   }

int screw(double len)
   {
   int i,j;
   /* draw a screw using a loop for */
   /* the threads */
   newpath();
   moveto(0,0);
```

```
   rlineto(0.25,0,0,0.25,-0.5,0,0,-0.25);
   closepath();
   moveto(0.15,0);
   rlineto(0,-len,-0.3,0,0,len);
   moveto(-0.15,0);
   j = (int) (len / 0.2);
   for (i=0;i < j;i++)
      {
      rmoveto(0.3,0);
      rlineto(-0.30,-0.20);
      }
   stroke();
   }

int heater()
   {
   int i;
   /* draw a heater using a loop */
   /* for the windings */
   double y = 0;
   newpath();
   moveto(0,0);
   for (i=0;i < 5;i++)
      {
      moveto(0.1,y);
      arcn(0,y,0.1,360,0);
      y = y + 0.4;
      }
   stroke();
   }

int slit()
   {
   /* draw a spectrometer slit */
   newpath();
   moveto(0,0);
   rlineto(0,1.5,0.25,0,0,-1.25);
   closepath();
   stroke();
   }

int driver()
   {
   /* draw long screw drivers */
   newpath();
   moveto(0,0);
   rlineto(0.15,0,0,0.25,0.1,0,0,10.8);
   moveto(0,0);
   rlineto(-0.15,0,0,0.25,-0.1,0,0,10.8);
   stroke();
   }
```

```
int white_box(double x1,double y1,double x2,double y2)
   {
   /* draw a white filled box for */
   /* white out purposes. The parametrs */
   /* passed in are the box corners */
   newpath();
   box(x1,y1,x2,y2);
   gsave();
   set(LINECOLOUR,WHITE);
   fill();
   grestore();
   }

int lens()
   {
   /* draw a spherical lens using arcs */
   newpath();
   arcn(0,-4,4,110,70);
   arcn(0,3.45,4,290,250);
   closepath();
   stroke();
   }

int prism()
   {
   /* draw a prism */
   newpath();
   moveto(0,0.5);
   rlineto(0,-0.5,0.5,0);
   closepath();
   stroke();
   }

int tail()
   {
   /* draw the dewar tail */
   set(LINECOLOUR,BLACK);
   moveto(2.44,30);
   lineto(2.44,6.33,7.77,6.33,7.77,30);
   stroke();
   /* draw windows */
   set(LINECOLOUR,BROWN);
   white_box(2.29,12.30,2.59,9.41);
   stroke();
   white_box(7.62,12.30,7.92,9.41);
   stroke();
   }
```

5.11 multplot.spt




```
#include <splot.h>

int i,j;
double calc[11][4];
double *data;

double calc2[11][4];
double *data2;

double calc3[11][4];
```

```
double *data3;

main()
   {
   /* read in data files */
   readdata("demo\mult1.dat",data);
   readdata("demo\mult2.dat",data2);
   readdata("demo\mult3.dat",data3);

   /* average batch values */
   for (i = 0; i < 11;i++)
      {
      for (j = 0;j < 10;j++)
         {
         calc[i][0] += data[i * 10 + j][0];
         calc[i][1] += data[i * 10 + j][1];
         calc[i][2] += data[i * 10 + j][2];
         calc[i][3] += data[i * 10 + j][3];
         calc2[i][0] += data2[i * 10 + j][0];
         calc2[i][1] += data2[i * 10 + j][1];
         calc2[i][2] += data2[i * 10 + j][2];
         calc2[i][3] += data2[i * 10 + j][3];
         calc3[i][0] += data3[i * 10 + j][0];
         calc3[i][1] += data3[i * 10 + j][1];
         calc3[i][2] += data3[i * 10 + j][2];
         calc3[i][3] += data3[i * 10 + j][3];
         }
      calc[i][0] /= 10;
      calc[i][1] /= 10;
      calc[i][2] /= 10;
      calc[i][3] /= 10;
      calc2[i][0] /= 10;
      calc2[i][1] /= 10;
      calc2[i][2] /= 10;
      calc2[i][3] /= 10;
      calc3[i][0] /= 10;
      calc3[i][1] /= 10;
      calc3[i][2] /= 10;
      calc3[i][3] /= 10;
      }

   abox(8,6,5,19.25);
   ascale(YAXES,0,100);
   ascale(XAXES,calc,0);

   /* set(YMULT,10);*/
   set(FONTMULT,0.7);
   set(TICKLENGTH,-0.2);

   tickmarks();
   ticklabel();
```

```
text(14.03,22.25,"1024 Data Items");

set(PLOTTYPE,SYM_LINES);
set(AXESCLIP,ON);
set(CURSYMBOL,OSQUARE);

set(LINECOLOUR,RED);
set(LINESTYLE,SOLID);
drawdata(calc,0,1);

set(CURSYMBOL,CROSS);
set(LINECOLOUR,BLACK);
set(LINESTYLE,DASHED);
drawdata(calc,0,2);

set(CURSYMBOL,OCIRCLE);
set(LINECOLOUR,GREEN);
set(LINESTYLE,DOTTED);
drawdata(calc,0,3);

/* the second data set */
set(LINECOLOUR,BLACK);
set(LINESTYLE,SOLID);

abox(8,6,5,11);
ascale(YAXES,calc2,1);
ascale(XAXES,calc2,0);

tickmarks();
ticklabel();

label(LEFT,"Throughput (TPS)");
text(14.03,14,"128 Data Items");

set(CURSYMBOL,OSQUARE);

set(LINECOLOUR,RED);
set(LINESTYLE,SOLID);
drawdata(calc2,0,1);

set(CURSYMBOL,CROSS);
set(LINECOLOUR,BLACK);
set(LINESTYLE,DASHED);
drawdata(calc2,0,2);

set(CURSYMBOL,OCIRCLE);
set(LINECOLOUR,GREEN);
set(LINESTYLE,DOTTED);
drawdata(calc2,0,3);

/* the third data set */
set(LINECOLOUR,BLACK);
```

```
set(LINESTYLE,SOLID);

abox(8,6,5,2.75);
ascale(YAXES,0,10);
ascale(XAXES,calc3,0);

tickmarks();
ticklabel();

text(14.03,5.75,"16 Data Items");

set(CURSYMBOL,OSQUARE);

set(LINECOLOUR,RED);
set(LINESTYLE,SOLID);
moveto(5.76,24.5);
symbol(OSQUARE);
rlineto(1.5,0);
symbol(OSQUARE);
drawdata(calc3,0,1);

set(CURSYMBOL,CROSS);
set(LINECOLOUR,BLACK);
set(LINESTYLE,DASHED);
moveto(5.76,23.6);
symbol(CROSS);
rlineto(1.5,0);
symbol(CROSS);
drawdata(calc3,0,2);

set(CURSYMBOL,OCIRCLE);
set(LINECOLOUR,GREEN);
set(LINESTYLE,DOTTED);
moveto(5.76,22.7);
symbol(OCIRCLE);
rlineto(1.5,0);
symbol(OCIRCLE);
drawdata(calc3,0,3);

set(LINECOLOUR,BLACK);
set(LINESTYLE,SOLID);

label(BOTTOM,"Multi Programming Level");
text(7.53,24.5,"MTL");
text(7.53,23.6,"TPL");
text(7.53,22.7,"TSO");
}
```

Chapter 6

Splot Definitions

Contents of Splot.h Constant Definition File

```
/* splot.h */

#ifndef SPLOT_H
#define SPLOT_H

/* general defines */
#define OFF 0
#define ON 1
#define TRUE 1
#define FALSE 0

/* define line styles */
#define SOLID 0
#define DASHED 1.0,0.5
#define DOTTED 0.2 0.2
#define DOTDASH 1.0,0.5,0.25,0.5

/* page size in cm */
#define XLENGTH 20.00
#define YLENGTH 26.00

/* define graphics draw op */
#define REP 0
#define AND 0x08
#define OR 0x10
#define XOR 0x18

/* define colours */
#define INVIS -1
#define BLACK 0
#define BLUE 1
#define GREEN 2
#define CYAN 3
#define RED 4
#define MAGENTA 5
```

```
#define BROWN 6
#define WHITE 7

/* define the axes labels */
#define LOWER 0
#define BOTTOM 0
#define UPPER 2
#define TOP 2
#define LEFT 1
#define RIGHT 3
#define XAXES 4
#define YAXES 5

#define CENTER 4
#define XCENTER 1.7e308
#define YCENTER 1.7e308

/* define axes types */
#define LINEAR 0x11
#define LOGX 0x12
#define LOGY 0x21
#define LOGLOG 0x22
#define INVX 0x14
#define INVY 0x41
#define INVINV 0x44
#define INVXLOGY 0x24
#define LOGXINVY 0x24

/* define line cap styles */
#define BUTTCAP 0
#define ROUNDCAP 1
#define PROJCAP 2

/* define line join styles */
#define MITERJOIN 3
#define MITREJOIN 3
#define ROUNDJOIN 4
#define BEVELJOIN 5

/* define fill rules */
#define NONZWIND 0
#define EVENODD 1

/* define devices */
#define DISPLAY 0
#define LASERJET 2
#define PAINTJET 3
#define PCXFILE 4
#define HP500 5
#define HP500C 6
#define OS2PMPRINT 7
```

```
/* define which column for error bars */
#define XVALS 0
#define YVALS 1

/* define default font info */
#define CELL_WIDTH 0.7 /* default font width */
#define CELL_ASPECT 2.0 /* default font height/width ratio */
#define CELL_SPACING 2.0

#define NFONTS 7 /* number of available fonts */
#define NORMALTEXT 0 /* for use with array of font pointers */
#define ROMAN_S 1
#define ROMAN_C 2
#define GREEK_S 3
#define GREEK_C 4
#define ITALIC_C 5
#define SYMBOL_S 6
#define SIMPLEX 0
#define COMPLEX 1

/* define symbol names */
#define OCIRCLE 0
#define OSQUARE 1
#define OTRIANGLE 2
#define ODIAMOND 3
#define OSTAR 4
#define OARROW 5
#define PLUS 6
#define CROSS 7
#define MULT 8
#define CIRCLE 9
#define SQUARE 10
#define TRIANGLE 11
#define DIAMOND 12
#define STAR 13
#define ARROW 14

/* define plot types */
#define LINES 0
#define SYMBOLS 1
#define SYM_LINES 2

/* define set options */
#define AXESCLIP 0
#define AXESTYPE 1
#define CURSYMBOL 2
#define FILLRULE 3
#define FLATNESS 4
#define FONT 5
#define FONTASPECT 6
#define FONTDIR 7
#define FONTMULT 8
```

```
#define FONTSPACE 9
#define FONTWIDTH 10
#define LABELMARG 11
#define LINECAP 12
#define LINECOLOUR 13
#define LINEJOIN 14
#define LINESTYLE 15
#define LINEWIDTH 16
#define MITERLIMIT 17
#define PAGEROT 18
#define PATTOFF 19
#define PLOTTYPE 20
#define SCALEALL 21
#define SCRIPTSCALE 22
#define SCRIPTSHIFT 23
#define SYMMULT 24
#define TICKLMARG 25
#define TICKLENGTH 26
#define XMULT 27
#define XRANGE 28
#define XSHIFT 29
#define YMULT 30
#define YRANGE 31
#define YSHIFT 32
#define LINECOLOR 13
#define MITRELIMIT 17
#define PAGE_ROT 18

/* set aliases for old commands */
#define tabdata readdata
#define hputs text

#endif

/* splot.h */
```

Chapter 7

Setable Parameters

Table of Default Values for Setable Parameters

The following values are in effect upon program startup, after execution of the "reset();" command or after clicking on the "reset" menu button. Other possible pick list choices defined in splot.h are given in brackets.

AXESCLIP = OFF	(ON)
AXESTYPE = LINEAR	(LOGX, LOGY, LOGLOG, INVX, INVY, INVINV, INVXLOGY, LOGXINVY)
CURSYMBOL = CIRCLE	(OCIRCLE, OSQUARE, OTRIANGLE, ODIAMOND, OSTAR, OARROW, PLUS, CROSS, MULT, CIRCLE, SQUARE, TRIANGLE, DIAMOND, STAR, ARROW)
FILLRULE = NONZWIND	(EVENODD)
FLATNESS = 1	
FONT = SIMPLEX	(COMPLEX)
FONTASPECT = 2.0	
FONTDIR = 0.0	
FONTMULT = 1.0	
FONTWIDTH = 0.7	
SYMMULT = 1.0	
LABELMARG = 0.0	
LINECAP = BUTTCAP	(ROUNDCAP, PROJCAP)
LINECOLOUR = BLACK	(INVIS, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, WHITE)

LINEJOIN = MITERJOIN (BEVELJOIN, ROUNDJOIN)

LINESTYLE = SOLID (DASHED, DOTTED, DOTDASH, or enter numerical pattern)

LINEWIDTH = 0.05

MITERLIMIT = 10.0

PAGEROT = OFF (ON)

PLOTTYPE = LINES (SYMBOLS, SYM_LINES)

SCALEALL = OFF (ON)

SCRIPTSCALE = 0.5

SCRIPTSHIFT = 0.7

TICKLMARG = 0.0

TICKLENGTH = 0.3

XMULT = 1.0

XRANGE = -1.7e308, +1.7e308

XSHIFT = 0.0

YMULT = 1.0

YRANGE = -1.7e308, +1.7e308

YSHIFT = 0.0

Chapter 8

Editor

8.1 Basics

The editor integrated into Splot is a fully featured editor that is also available as a separate product. The editor is modeled after the E PC editor. This editor is however a functional superset of E with many useful additional features. The OS/2 epm editor is another editor with the same ancestors. Users of E or epm will quickly feel at home with this editor since operation is quite similar.

Highlights of this editor include unlimited undo of previous changes and corresponding step for step redo. Column editing features to allow movement of columns of text (allows easy change of indentation). Bracket matching of all sorts of brackets and C code comment delimiters. There is a view mode which displays either; only the lines starting in the first column to quickly locate the start of functions or only the changed lines. Furthermore, a whole ring of files can be loaded up at once allowing painless copying of text between files. One particularly nice feature is the ability to load up a set of files in a ring and rapidly cycle through them using then ctrl N and ctrl P key combinations. Text can be copied between these files by using the block and line mark commands. There are all also some features to facilitate carrying out complex repetitive editing tasks. In particular sequences of key strokes can be saved and assigned to a function key for subsequent reuse. Furthermore, any command including function key macros can be repeated a number of times automatically by entering a command multiplier. The editor is completely key configurable. A set of keystrokes can be mapped to an editor function simply by specifying the sequence in the auxiliary key map file. The combination of key mapping and key macros allow the editor to imitate the user interface of other editors or allow the construction of a custom user interface to the taste of the user.

The editor is extremely easy to use and largely self explanatory. Text is input by straight typing anywhere on the screen. Cursor movement is controlled by the arrow keys and auxiliary keypad keys as per their standard definitions as well as accelerated movement with CTRL combinations. The cursor position control keys are as follows:

Cursor Motion Keys

left arrow	left one space
right arrow	right one space
up arrow	up one line
down arrow	down one line
end	end of line
home	beginning of line

PgDn	scroll down one screen
PgUp	scroll up one screen
Center Key	fast motion toggle
CTRL left	left one word
CTRL right	right one word
CTRL up	scroll up five lines
CTRL down	scroll down five lines
CTRL end	end of file
CTRL PgDn	end of file
CTRL home	beginning of file
CTRL PgUp	beginning of file

Editor Control Keys

CTRL A	set place mark
CTRL B	bracket match {, (, [, <, or /*
CTRL C	change case
CTRL D	delete character
DEL	delete character
CTRL E	delete to end of line
CTRL F	find next occur of search string
CTRL G	go. Execute code
CTRL H	destructive backspace
CTRL I	tab
CTRL J	join next line to current line
CTRL K	cut current line at cursor
CTRL L	return to previous set place mark
CTRL M	insert new line
CTRL N	edit next file in ring
CTRL O	
CTRL P	edit previous file in ring
CTRL Q	quit without saving file
CTRL R	redo previous undo
CTRL S	swap current and next character
CTRL T	toggle insert/replace
CTRL 0	toggle insert/replace
CTRL U	undo changes to last line
CTRL V	toggle view function mode
CTRL W	write file to disk
CTRL X	delete line
CTRL BACK	delete line
CTRL Y	
CTRL Z	zap word
CTRL DEL	zap word
ALT A	alternate binary representation
ALT E	execute last macro
ALT I	insert coordinates
ALT J	join next line to current line
ALT K	enter next char literally

ALT P	position curs at prev location
ALT Q	restore a line from undo record
ALT R	re flow paragraph
ALT S	split line at cursor
ALT T	teach new macro
ALT V	view only changed lines
ALT W	view lines containing search string
ALT X	enter control character
ALT Z	zero command multiplier
ALT 0-9	enter command multiplier

The ALT P command is useful if the cursor was accidentally moved away from the region of interest with a CTRL Home or search for example. In these situations ALT P will restore the cursor to its previous location. This action is sort of similar to using place marks except that unlike place marks these are automatically set by any command that causes the cursor to jump by more than one line. Only one previous location is remembered though.

8.2 Block manipulations

Block mark and line mark are somewhat different. Use line mark for moving, copying or deleting one or more whole lines and block mark for moving, copying, deleting, filling or overlaying columns of text. Note also that a block marked region can also be moved to the left with ctrl ∧ and to the right with ctrl _ but the behaviour is different than if the ALT arrow keys are used. ALT left and ALT right (or ctrl _ and ctrl ∧) are meaningless for line marked regions.

Block Manipulation Keys

ALT B	mark column block start, end
ALT C	copy marked block
ALT D	delete marked block
ALT F	fill B marked block with char
ALT L	mark LINE(s) start, end
ALT M	move marked block
ALT O	overlay B marked block
ALT U	unmark block
ALT Y	yank back deleted block
CTRL_	shift B marked block right
CTRL∧	shift B marked block left

8.3 Macros

There is also a macro capability for repetitive, complex editing tasks. The editor is taught a macro by hitting the ALT T combination. After this all subsequent keystrokes are stored as the macro definition until ALT T is hit again. There is a 128 keystroke limit on the length of a macro. A macro can consist of any keystrokes except ALT T and ALT E. At the end of the macro definition the macro can be bound to a function key by hitting the desired function key. Available function keys are F1 - F12 as well as shift

F1 - F12. A bound macro is subsequently executed by hitting that function key. The predefined macro for that key will of course be lost. The last learned macro whether bound or unbound can be executed by hitting ALT E. If the macro involves a search and another item is not found the rest of the macro is not executed. If macros have been bound to function keys they will be written to the configuration file splot.cfg which will be created in the current directory so that the macros will be preserved in between editing sessions.

A repeat factor for a keystroke can also be entered by prefixing the command with a multiplier which is input using the ALT number keys. The multiplier can be reset to zero during input using ALT Z. This feature also facilitates repetitive editing as an operation can be repeated several times. This is particularly powerful in conjunction with macros.

8.4 Command Line

Lastly, there are a small number of commands which require typed input on the command line. The command line is reached by hitting ESC. This places the cursor at the beginning of the command line which is on the last row and ordinarily displays some status information. After ESC however text can be input here for the following functions:

Command Line Operations (requires
text input so terminate line with a
return)

esc	goto or leave command line
	Change current line
number	goto line number n
+number	down n lines
-number	up n lines

Change current column offset

@number	start at column n
@+number	scroll left n columns
@-number	scroll right n columns

Search and Replace.
find string str (/ actually any punct)
/str[/-bclr] return
replace str1 with str2
c/str1/str2[/-bcl] return

The optional search suffixes are

/-	search up (down is default)
/b	search marked block only
/c	ignore case
/l	loop through all files
/r	interpret as regular expression

For regular exp the following symbols
have special meaning
∧ start of line
$ end of line

. any character
 quote next character
* match zero or more times
+ match one or more times
[aei0-9] match a,e,i and 0 through 9
[∧aei0-9] match anything but a,e,i and 0 through 9
( ) sub expression
a(ab)*b matches ab aabb aababb aababababb etc.

Add other files to ring
e name1,name2,...[-r]
The optional edit suffixes are
-r read only

display directory and choose one to edit
e [*.*]
activate this help file
?
insert named file at current location
m name
rename current file to name
r name
show and modify editor configuration
configure
exit without saving changes
quit
previous command line entered
up arrow
next command line entered
down arrow
send command to operating system
os command
also any unrecognized string

8.5 configuration file

The startup file splot.cfg preserves the values selected from the configuration menu item between sessions. You can have multiple splot.cfg files in different directories since the current directory is searched before using the environment variable SPLOTPATH to find the default splot.cfg. The splot.cfg file is plain ASCII and can thus be edited with a text editor as an alternative means of changing the settings or the macro definitions. In particular, to change the text string inserted in the text by chosing a draw menu item the splot.cfg defined function macros should be edited.

The macros are defined in plain text and thus in order to accommodate control and cursor keys as well each macro keystroke is defined by a pair of letters using the following scheme. Starting in the first column of a definition must appear the key word or function key identifying the macro. This is followed by the macro body. Function keys may not appear in macro bodies. Keystroke and are encoded as follows with implied sequences in ASCII order.

Ordinary characters	prepend with _	
Control Keys	prepend with ∧	∧M Return, ∧[esc
Alt Keys	prepend with %	
Cursor keys	prepend with #	#. Del, #0 Ins - #9 PgUp
Ctrl Cursor Keys	prepend with #	#N Del, #P Ins - #Y PgUp
Alt Cursor Keys	prepend with #	#n Del, #p Ins - #y PgUp
Function Keys	prepend with $	$1 - §¡
Shift Function Keys	prepend with $	$= - $H

Chapter 9

Acknowledgements

I wish to express my thanks to the people that assisted me during the development of this program. First of all Alan Steele, who wrote the first version of the Hershey font drawing routine and with whom the philosophy of the program was extensively discussed. I would like to acknowledge Clare Morrison for helping with the preparation of this document and for extensive testing. I thank the people who put up with many alpha and beta versions that were far from perfect. In particular, I wish to thank Alan Steele, Scott Wilson, Clare Morrison, Yves Lacroix, Lori Lenchyshyn, Valentin Karasyuk, Simon Watkins and Darlene Brake for helpful feedback.

I n d e x

abox 35
alineto 35
arc 35
arrowto 36
ascale 36
attributes, changing 11
axes box 10, 36
    invisible example 56
    log example 26

Bezier curves 37
box 36

C 5,7,8,35
    library functions 11
        abs 45
        acos 45
        asin 45
        atan2 45
        atan 45
        atof 45
        atoi 46
        ceil 46
        cos 46
        exit 46
        exp 46
        fabs 46
        floor 46
        fmod 46
        free 46
        log10 46
        log 46
        malloc 46
        pow 47
        printf 47
        print 47
        puts 47
        sin 47
        sizeof 47
        sprintf 47
        sqrt 47
        strcat 47
        strcpy 48
        strlen 48
        tan 48
    C, keywords 11
clear 36
clip 10, 31, 37
    example 31
closepath 37
cmatch 31, 37
constants, definitions 77
coordinates 10, 12
    changing 12
    inserting 10
    matching 31
    relative 11
cursor positioning 83
curveto 37

data 37, 38, 39
    always load 39
    file 10
    plotting 10
    shifting example 53
drawdata 10, 24, 37, 39
    clipping 31
drawing functions 35
drawing, example 66

editor 7, 9, 83
    block mark 85
    command line 86
    control keys 84
    macros 11, 85
error bars 10, 37
    example 52
example files 49
exiting program 9

fill 9, 24, 37
finding elements 10
fit line 38
    example 52

get attributes 38
greek 13

gsave/grestore 24, 38

hard copy 11
highlighting elements 10

implicit stroking 10, 39
input focus 10
inset 26
installation 33
interactive drawing 9, 10, 12
interpreter 5
italic 13
label 13, 38
landscape 31
lineto 9, 39

main 11
making a drawing 9
math 5, 11
  example 63, 73
menus 10, 11, 12
moveto 9, 39

newpath 10, 26, 39

online help 10

path 9, 37, 38, 40
plotdata 7, 10, 39
  clipping 31
portrait 31
postscript 5, 9, 11
  encapsulated 11
puts 38

rarrowto 39
readdata 7, 10, 39
reset 40
rlineto 40
rmoveto 40
rotate 31, 40

saving file 9
scale 26, 36, 40
set attributes, default values 81
set attributes 40
set 9, 11, 38
  functions 42
  AXESCLIP 31, 42
  AXESTYPE 42
  CURSYMBOL 24, 42
  FILLRULE 42
  FLATNESS 42
  FONTASPECT 13, 43
  FONTDIR 13, 43
  FONTMULT 13, 43
  FONTWIDTH 13, 42
  FONTSPACE 13, 43
  FONT 13, 42
  LABELMARG 38, 43
  LINECAP 43
  LINECOLOUR 13, 43
  LINEJOIN 43
  LINESTYLE 43, 43
  LINEWIDTH 13, 43
  MITERLIMIT 44
  PAGEROT 31, 44
  PATTOFF 44
  PLOTTYPE 24, 44
  SCALEALL 44
  SCRIPTSCALE 13, 44
  SCRIPTSHIFT 13, 44
  SYMMULT 24, 38, 44
  TICKLENGTH 41, 44
  TICKLMARG 41, 44
  XMULT 45
  XRANGE 38, 42, 451
  XSHIFT 45
showpage 10, 40
starting splot 33
stroke 9, 24, 26, 40
subroutine 5
subscript 13
superscript 13
symbol 37
symbols 8, 10, 24
system requirements 33

text, character table 17
text 13, 26, 39, 41
tick labels 30, 39
tick marks 30, 39
  length 30
  width 30
  example 30
tickmarks 10, 41
ticklabel 10, 13, 41
translate 26, 31, 42

units 10, 31
using mouse 9, 10

whereis 42
white out 10
  example 25

zoom 10, 11