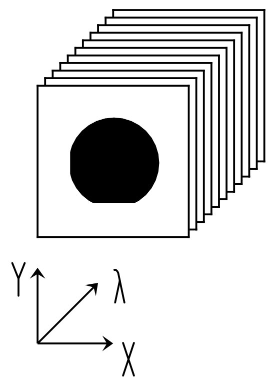
```
   grestore();
   set(LINECOLOUR,BLACK);
   stroke();
   }
```

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.

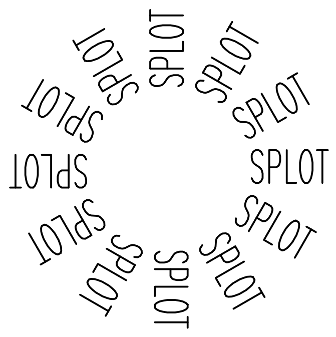
```
#include <splot.h>

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();
      }
   }
```

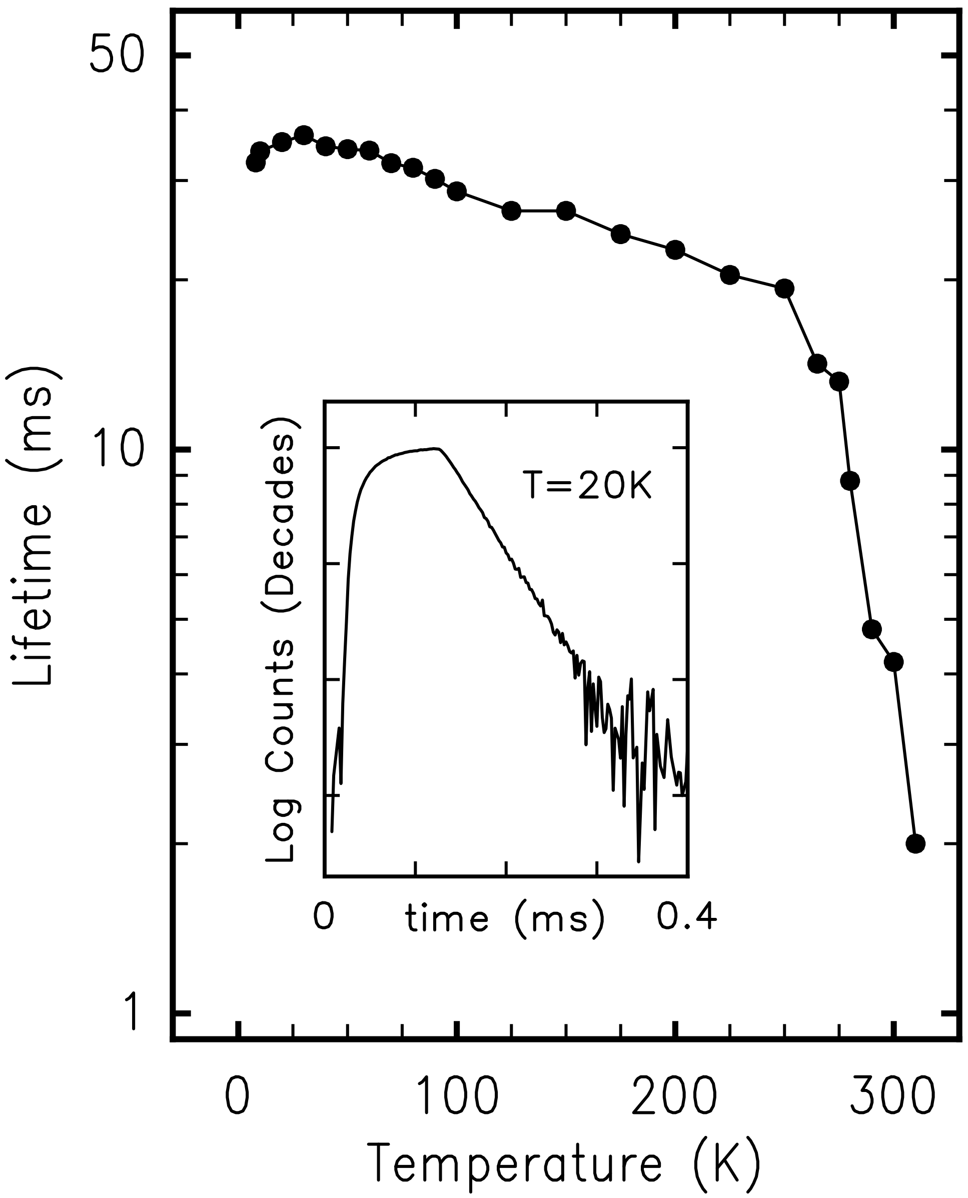


2.3.4 Insets

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)");
}
```

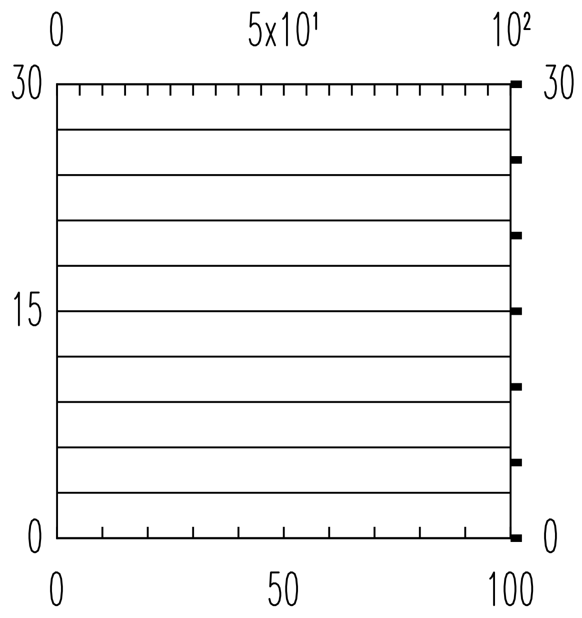

50
10
1
Lifetime (ms)
0
100
200
300
Temperature (K)
Log Counts (Decades)
T=20K
0
time (ms)
0.4

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.

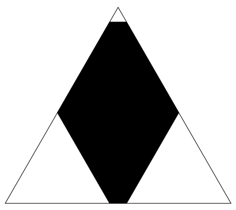
```
#include <splot.h>
main()
   {
   /* tick marks and tick label example */
   abox(12,12,5,6);
   ascale(XAXES,0,100);
   ascale(YAXES,0,30);

   /* 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 documentation in LaTeXform and the resultant postscript document containing all the figures in the demo directory take up another 2 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 appropriate 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. There are two included configuration files splot.cfg and spexpert.cfg. The default is splot.cfg and contains function macros which will fill in prompts for the necessary parameters. For an expert Splot user this is annoying since these prompts must always be erased. Experts should copy spexpert.cfg to splot.cfg to remove the parameter prompts or customize the parameter prompts in splot.cfg.

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

To run the program type splot (or spltdemo for the demo version) on a command line followed by an optional file name(s). Alternatively after setting up an association between *.spt files and splot.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 tickmarks 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 -π 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.