Appendix B

PLT Command summary

CLear

`CLear`

Immediately clear the current plot device.

COlor

`COlor [#] ON|OFf [glist]`

The first (optional) number allows you to reset the default color index for the plot groups specified in `[glist]`. If this number is omitted, then the default color index is not changed. `COlor OFf` turns off the plotting of all plot groups specified; `COlor ON` turns the plotting back on. If `[glist]` is omitted then all plot groups are affected.

`COlor MOdel #`

Plot the model using color index `#`.

`Color # ON Grid`

Use the color index specified by `#` when plotting the Grid in the currently active window.

`COlor ?`

Generate a list of possible color indices and their default color representations.

Examples:

```
PLT> COlor OFf 1,2,3  ! Suppress plotting of groups 1, 2, and 3
PLT> COlor ON  2      ! Turn on plotting of group 2
PLT> COlor 5 ON 3     ! Use color 5 when plotting group 3
```

CONtour

`CONtour [#] ?|COlor list|LEvel list|LStyle list|LWidth list|OFf`

This command enables PLT to produce a contour plot. It is still experimental, and so the syntax could change. If PLT is displaying contours, then you can use the `SHow Contour` command to display the PLT commands that were used to generate that PLT.

CQuit

`CQuit`

Immediately clear the plot device and exit subroutine PLT.

CSize

`CSize #`

Set character size to `#` where `#` is greater than zero and less than five (one is default).

Example:

```
PLT> CSize 2.0       ! Use a character size twice the default
```

Device

`Device [$]`

Change the current plotting device. The current plot device is released; the next plot will be produced on the new device. Note: `Device` with no argument releases the current plotting device.

Examples:

```
PLT> Device /PS      ! Output Postscript commands to a file
PLT> Plot            ! Send a plot to the file
PLT> Device          ! Close the file
```

Error

`Error OFf [glist]`

Turn off the plotting of errors for all plot groups specified in `glist`.

`Error ON  [glist]`

Undo the effects of the `Error OFf` command for the specified plot groups.

`Error Sq  [glist]`

Set errors equal to `SQRT(value)` for the specified plot groups.

`Error Dia [glist]`

Draw diamond style errors on specified groups.

`Error X ON|OFf [glist]`

This command controls the plotting of the x-error bars. For example, if groups 1,2, and 3 all have associated errors, then `Xaxis 1`; `Error ON 2`; `Error OFf 3` would cause the x-errors to be plotted when group 2 is plotted but not when group 3 is plotted.

In all of the above, `[glist]` can include `X` to specify the x-coordinate. When fitting data, error bars will be used to weight the data only if the errors are visible. Thus

`Error OFf` followed by `Fit` will produce an unweighted fit. Also, note that only the y-errors are used to weight the data.

Examples:

```
PLT> Error OFf 1,2,3  ! Turn off errors for groups 1, 2, and 3
PLT> Error Dia 2      ! Plot diamond errors on plot group 2
PLT> Error ON         ! Plot errors for all plot groups
PLT> SKip Sing        ! Turn on the 'skip' mode
PLT> Error OFF X      ! Suppress plotting of errors on X-axis
```

EXit

`EXit`

Exit subroutine PLT.

Fit

`Fit [#] [Iterate #]`

Cause PLT to call the fitting routine to search for the best fitting parameters for the model specified with the `MOdel` command. The first optional number is the plot group to fit. If this number is omitted, then `Fit` will continue to fit the previous group that you were fitting if that plot group is still visible. For the first use of `Fit` or if the plot group has been `COlored OFf` then `Fit` will default to fitting the lowest numbered group which is visible. `LAbel PArm` command controls whether the model parameters are plotted on the right side of the plot. Fitting continues until $\Delta\chi^2$ is less than 0.05. As a default, you will be prompted to continue fitting every 10 iterations. If you wish to increase the number of iterations without being prompted, then use the `Fit Iterate [#]`. Thus, `Fit I 100`, would allow the program to try up to 100 iterations before you would be prompted again.

`Fit ON [#]`

Cause the current model to be plotted on group `#`. If `#` is omitted, the model will be plotted on the plot group that was most recently fitted.

`Fit OFf`

Cause the model to no longer be plotted.

`Fit Plot #`

Control where the model is evaluated when it plotted. If `#` is greater than zero, then the model is evaluated at `#` points between the current x-scale Minimum and Maximum. If `#` is less than 0, then the model is evaluated at `ABS(#)` points between the Data Min and Data Max — *i.e.*, no extrapolation is allowed. If `#` equals 0, then the model is evaluated at the values of the data points. (This is the default).

`Fit Stat Chi|Ml #`

Set the default fitting statistic to either χ^2 or maximum likelihood. For example, `Fit Stat M 2` would vary the fit parameter to minimize the likelihood function when compared with plot group 2.

FNy

`FNy #`

Return the value of the current `MOdel` function at location `#`.

Example:

```
PLT> MOdel LINR CONS ! Define a straight line
  1, LI: VAL( 1.00), SIG( 0.00 ), PLO( 0.00), PHI( 0.00)?
2.                   ! Set slope of line equal to 2
  2, CO: VAL( 1.00), SIG( 0.00 ), PLO( 0.00), PHI( 0.00)?
3.                   ! Set intercept of line equal to 3
PLT> FNY 3           ! Evaluate function at x=3.0
12.0                 ! The result
```

FOnt

`FOnt $`

Set the default to the font specified by the character string. (Default is Simple font).

`FOnt ?`

List possible fonts.

Example:

```
PLT> Font Roman      ! Use the Roman font
```

FReeze

`FReeze [plist]`

Can only be used after a model has been defined. This command causes all parameters listed in `[plist]` to be frozen, which means the parameter value is not allowed to vary during a `Fit`.

Examples:

```
PLT> FReeze 2       ! Freeze the value of parameter 2
PLT> FReeze 3..5 9  ! Freeze the values of parameters 3 through 5 and 9
```

GAp

`GAp #`

Control the size of the gap between the edge of the plot and the data extrema, when using the default scale. The effect of the `GAp` command can only be seen when you reset the default scale with `R`, `R X`, or `R Y` (all with no arguments). The default gap is 0.025 which will leave a 2.5 percent margin around the edge. The size of the gap in world coordinates, depends on whether the data are being plotted on a logarithmic scale. Therefore, if you wish to use a default scale with a logarithmic scale, you should first issue the `LOg` and `GAp` commands and then use the `Rescale` command to reset the default scale.

Examples:

```
PLT> GAp 0.0         ! No gap
PLT> R X             ! Use default X scale
PLT> GAp 0.05        ! Reset gap, for use with Y scale
PLT> LOg Y           ! Log the Y scale
PLT> R Y             ! Use default Y scale
```

Grid

`Grid [clist]`

Control the plotting of a grid, where `[clist]` is one or more of `ON`, `OFf`, `X # #`, `Y # #`. For `Grid ON`, the major tic marks are expanded to place a grid over the entire plot. `Grid OFf` turns off the grid replacing the tic marks. PGPLOT automatically selects the location of the major and minor tic marks. The `Grid` command allows you to override this selection. For example, `Grid X 10,2` would divide the x-axis into 10 major divisions and then divide each major division into 2 minor divisions. Use `Grid X 0,0` to go back to the default grid. The number -1 can be used to suppress the plotting of tic marks. Thus, `Grid X -1 0` would only plot minor tic marks on the x-axis and `Grid Y -1 -1` would prevent any tic marks from being plotted on the y-axis.

IMPORTANT: PGPLOT only places major tic marks at locations where the least significant digit of the range increments by integer amounts. If you attempt to force PGPLOT to violate this condition, then unexpected things may happen. For example, suppose you had used `R Y 0 .15`, in which case the range is 0.15 and the least significant digit is 0.01. Then using `Grid Y 3,2` would place grid lines at intervals of 0.05. However, if you had tried `Grid Y 2,2`, then the grid lines would occur at intervals of 0.075, where the last digit (5) is not a significant digit. In this case, the plot will be incorrectly labeled. `Grid X 1.5,2` is legal and would correctly produce labels at intervals of 0.1. (Of course, only one label would actually be plotted in this case).

Hardcopy

`Hardcopy [$]`

Generate a copy of the plot using the current hardcopy specification. In general, this makes a file that can later be printed. The default PLT hardcopy device for the first use can be set using the PLT_HARDCOPY logical name (on VMS systems) or environment variable (on UNIX systems). After the first time, the `Hardcopy` command defaults to the same device as that specified in the previous `Hardcopy` command. `Hardcopy` produces a copy of what you would see if you reissued the `Plot` command. This might not be an exact copy of what is currently displayed on the graphics device.

`Hardcopy ?`

Display on the terminal the default hardcopy specification.

Example:

```
PLT> Hardcopy /PS   ! Create a Postscript file
```

HElp

`HElp [pcommand]`

Get help on the PLT command specified by `[pcommand]`.

Imodel

`Imodel # # [#]`

Integrate the current model over the range specified by the first two parameters. The third (optional) argument is the number of steps. (Default is 100).

LAbel

`LAbel X|Y|Top|OX|OY|OTop|File|G# [label]`

Place a label. The arguments `X`, `Y`, and `Top` would place a label on the x-axis, y-axis, or the top of the plot. It is possible to place an additional line of text just outside these locations by putting the letter '`O`' in front of these names. For example, `OT` would write a second label above the top label. Finally one can change the file name by using `LA File [name]`. If you should at any time wish to delete a label, then just omit the `[label]` from the command. The command `LA G# label` would associate a label with a particular plot group. PLT will use the group label on either the x or y-axis if there is no corresponding x or y label.

`LAbel OFf`

Suppress plotting of all text labels. (This speeds up the production of the plot on slow devices).

`LAbel ON`

Undo the effects of `LAbel OFf`.

`LAbel NX|NY ON|OFf [winlist]`

where `[winlist]` is a list of windows. NX affects the plotting of the numeric labels on the x-axis, NY on the y-axis. Thus `LAbel NX OFF 1`; `LAbel NY OFF 1`, will prevent the plotting of the numeric labels in both the X and Y directions for the first window. Note: there are 32 numbered labels, so using `LAbel NX OF`, followed by several `LAbel #` commands will allow you to create labels consisting of text strings *etc.*

`LAb POS Y #`

Allows the position of the y label(s) to be moved. The default position is `LA POS Y 2.0`.

`LAbel Rotate`

Rotate numerical labels plotted on the y-axis. The default PGPLOT mode is to plot the y-axis numerical labels in the vertical direction. `LAbel Rotate` will rotate these labels so they are written in a horizontal direction. If you issue the command a second time, then the labels will flip back to vertical in the next plot.

Examples:

```
PLT> LA F               ! Delete the file name from the plot
PLT> LA T Test! Plot    ! Top label is Test
PLT> LA T "Test! Plot"  ! Top label is Test! Plot
```

`LAbel Parm ON`

Display the parameter values, associated with the current model, on the right side of the plot. (This is the default).

`LAbel Parm OFf`

Suppress the display of the parameter values on the right side of the plot.

`LAbel # [clist] "string"`

Allow a label to be placed anywhere on the existing graph. Here `#` is a number in the range 1 to 32, `[clist]` is a list of subcommands that allow you to change various attributes of the label, and "string" is the (optional) text. For example,

`PLT> LAbel 1 Pos 2 2 "test"`

will create label 1 at position (2,2) containing the message `test`.

`LAbel # CEnter Top|Cap|Half|BAse|BOttom`

Control the vertical position of the text string. The default position is half, although the use of the `LIne` subcommand will override this.

`LAbel # COlor #`

Cause the label to be plotted with color index `#`. (Default is color index 1).

`LAbel # CSize #`

Cause the label to be plotted with character size `#`. (Default is character size 1.0).

`LAbel # Justify Left|Center|Right`

Control justification of text string. (Default is to center justify).

`LAbel # LIne # [#]`

Draw line at angle `#` (in degrees) from position to the label. If the second number is specified, then it will be used as the length of the line in viewport units. (Default line length is 0.08). If no number follows the `LIne` command, then the line is turned off.

`LAbel # LStyle #`

Control the line style of the line plotted, when the `LIne` subcommand is used. (Default is 1).

`LAbel # Marker #`

Include marker `#` in the text line. The default will produce a left justified string just to the right of the marker. If you then right justify the string, then the string will appear to the left of the marker. In either case, the (x, y) position corresponds to the position of the marker.

`LAbel # MSize #`

Use the specified marker size with plotting the specified numbered label.

`LAbel # Position # #`

Specify the (x, y) location of the string in world coordinates. When using the `LIne` subcommand, `Position # #` gives the (x, y) location of the position being 'pointed at' with the text string at the other end. If the `Marker` subcommand is used, then `# #` refers to the location of the marker. For other cases, `# #` is the location of the string. The `CEnter` and `Justify` subcommands can later override how text is plotted relative to a location. (The default location is (0,0)).

`LAbel # Position Cursor`

Display the cursor at the current default position. You can move the cursor to the desired position using the standard (PGPLOT) cursor keys. Once at the location where you want the label, press the space bar. The next time the plot is drawn, the label will appear at the new location.

`LAbel # Rotate #`

Plot the label at an angle of `#` (degrees) relative to the x-axis. (Default angle is 0).

`LAbel # VPos # #`

Specify the (x, y) location of the string in viewport coordinates. When using the `LIne` subcommand, `Position # #` gives the (x, y) location of the position being 'pointed at' with the text string at the other end. If the `Marker` subcommand is used, then `# #` refers to the location of the marker. For other cases, `# #` is the location of the string. The `CEnter` and `Justify` subcommands can later override how text is plotted relative to a location.

`LAbel # VPos Curs`

Display the cursor at the current default position. You can move the cursor to the desired position using the standard (PGPLOT) cursor keys. Once at the location where you want the label, press the space bar. The next time the plot is drawn, the label will appear at the new location.

Examples:

```
PLT> LAbel 1 "TEST"    ! Place the word TEST at location (0,0)
PLT> LAbel 1 P 10 10   ! TEST will now appear centered at (10,10)
PLT> LAbel 1 LIne 90   ! Draw a vertical line from the point (10,10) to
                       ! the word TEST.
PLT> LAbel 1 CO 3      ! Color line and word TEST green
PLT> LAbel 1 P 10 10 LIne 90 CO 3 "TEST"    ! Does all the above in a
PLT>                                        ! single command
```

LIne

`LIne ON [glist]`

Draw a line connecting all the points in each plot group specified in `[glist]`.

`LIne OFf [glist]`

Produce a scatter diagram by turning off the line for each plot group specified in `[glist]`.

`LIne Stepped [glist]`

Produce a stepped-line plot for each plot group specified in `[glist]`.

Note: If you set `Error OFf`, `MArker OFf`, and `LIne OFf`, then the line will still appear. The only way to prevent data from being plotted is to use the `COlor OFf` command.

Examples:

```
PLT> LIne ON         ! Use a line for all groups
PLT> LIne OFf 1,3    ! Turn off the line for groups 1 and 3
PLT> LIne Step 1,5   ! Plot groups 1 and 5 with a stepped line
```

LOCation

`LOCation # [# [# [#]]]`

The `LOCation` command allows you to control the location of the currently active window. The default location of all windows is 0. 0. 1. 1. which means that all windows overlap and use the entire plotting surface. This command in combination with the `WIndow` command allows great control over where the windows are plotted. Thus a small window could be plotted inside a larger one. However, no attempt is made to erase a plotting region, so overlapping windows could result in overlapping text.

Example: To create 4 windows in the 4 quadrants, use:

```
PLT> WIndow 1
PLT> LOCation 0 .5 .5 1.
PLT> WIndow 2
PLT> LOCation .5 .5 1. 1.
PLT> WIndow 3
PLT> LOCation 0. 0. .5 .5
PLT> WIndow 4
PLT> LOCation .5 0. 1. .5
```

LOg

`LOG X|Y|X,Y ON|OFf [winlist]`

Controls whether a log scale is used when plotting in the windows specified in winlist. Thus `LOG Y OFf 2 3 4`, would turn off the use of a log scale in the y direction in windows 2 3 and 4. Also, `LOG X,Y OFf 3` would turn off the log scale in both the x and y directions when plotting in window 3.

LStyle

`LStyle # [ON] [glist]`

Change the default line style for the plot groups specified in `[glist]`. The first number is the style type.

`LStyle # ON Grid`

Use the line style specified by # when plotting the grid in the currently active window.

`LStyle ?`

List possible line styles.

Examples:

```
PLT> LStyle 2 ON 3    ! Use line style 2 when plotting group 3
PLT> LStyle 1         ! Use solid line when plotting all groups
PLT> LStyle 5 ON 1,4  ! Use line style 5 when plotting groups 1 and 4
```

LWidth

`LWidth #`

Set the line width to the value #. Allowed values are 1 or greater. On some laser printers, the default line width is very narrow and so, using thicker lines will enhance the output quality.

Examples:

```
PLT> LWidth 3         ! Triple the default thickness of all lines
PLT> HArd /PS         ! and make a hardcopy
```

MArker

`MArker [#] ON [glist]`

Turn on the plotting of polymarkers. The first (optional) number is the marker type; if omitted the default marker type will not be changed. If [glist] is omitted, then markers will be used to plot all plot groups.

`MArker Size # ON [glist]`

Changes the size of the markers when plotting the plot groups specified in [glist]. The size can range from 0.0 to 5.0, where 1.0 is the default. Thus every plot group can have a different marker size.

`MArker OFf [glist]`

Turn off the plotting of markers for all plot groups specified. If [glist] is omitted, then markers will be removed from all groups.

`MArker ?`

Draw a table of all the PGPLOT markers on your current graphics device.

Examples:

```
PLT> MArk 19 ON 2     ! Use marker 19 when plotting group 2
PLT> MArk OFF 2       ! No longer plot group 2 with a marker
PLT> MArk ON 2        ! Use default marker when plotting group 2
PLT> MArk ON          ! Use markers when plotting all plot groups
PLT> MArk Size 5      ! Plot markers 5 times larger than default size
```

MOdel

`MOdel ?`

List all built-in model components.

`MOdel @filename`

Cause the model definition and parameters to be read from the file with name `filename`.

`MOdel $`

Use to define models. Any combination of model components can be added together. For example, `MOdel CONS LINR QUAD` will add a constant term, a linear term, and a quadratic term.

For each parameter required by the `MOdel` command, you will be prompted for four numbers — `VAL`, `SIG`, `PLO`, and `PHI` — as described below. For each parameter, you should enter an initial value for `VAL`; but you can usually default on the other three numbers.

`VAL`: This is the actual value of the parameter. Although CURFIT will often find the the best set of parameters to model the data, it never hurts to start it with parameters near the expected best fit.

`SIG`: Any value of `SIG`≥0 will not affect the outcome of `Fit`. After you fit the model, `SIG` will contain the one-sigma curvature errors. This number is used by the `Uncertainty` command to start a formal error determination. If the `Uncertainty` command fails to converge because the original error estimate is wrong, sometimes you can help the convergence by adjusting `SIG` to be a better estimate before using `Uncertainty`. If you set `SIG`=-1, then the parameter is frozen such that CURFIT is not allowed to change the parameter value while fitting. If you set `SIG`=-IPAR, the next number (`PLO`) will default to 1, such that the current parameter value is forced to equal the value of parameter IPAR. (Note: IPAR can not equal 1 or the current parameter number). If you place a number (N) after `SIG`, this will force the current parameter to be N times the specified parameter. (N defaults to 1.0.)

Example:

```
PLT> MOdel GAUS GAUS
  1, GC: VAL( 1.00), SIG( 0.00 ), PLO( 0.00), PHI( 0.00)?
,-4,2
  2, GW: VAL( 1.00), SIG( 0.00 ), PLO( 0.00), PHI( 0.00)?

  3, GN: VAL( 1.00), SIG( 0.00 ), PLO( 0.00), PHI( 0.00)?

  4, GC: VAL( 1.00), SIG( 0.00 ), PLO( 0.00), PHI( 0.00)?
3.
  5, GW: VAL( 1.00), SIG( 0.00 ), PLO( 0.00), PHI( 0.00)?

  6, GN: VAL( 1.00), SIG( 0.00 ), PLO( 0.00), PHI( 0.00)?
```

defines a model consisting of two gaussians, with the x values of the centers differing by a factor of 2. Although a value for parameter 1 was not entered, it will be set to the value of 6 (2 times value of parameter 4). This relationship will be maintained throughout a fit.

`PLO`, `PHI`: If `SIG`$\geq$0 and if `PLO`<`PHI`, the parameter value is constrained to lie within the range `PLO` to `PHI`. Note: If `PLO` and `PHI` are the same (say equal to zero), then the parameter will not be constrained in any way. In general, if you have difficulty fitting some data, the best thing to do is to freeze some parameters near to their expected values and then fit the reduced parameter set. When a good fit has been found with the reduced set, thaw some of the parameters and refit. If this method does not work, then you may be forced to use `PLO` and `PHI` to limit certain parameters to a meaningful range.

QDP/PLT currently supplies the model components defined below, which may be combined into multi-component models:

`MOdel CONS`

Select a model with a constant component:
`FNY=FNY+CO`.

`MOdel LINR`

Select a model with a linear component:
`FNY=FNY+LI*X`.

`MOdel QUAD`

Select a model with a quadratic component:
`FNY=FNY+QU*X**2`.

`MOdel CUBI`

Select a model with a cubic component:
`FNY=FNY+CU*X**3`.

`MOdel X4`

Select a model with an x^4 component:
`FNY=FNY+X4*X**4`.

`MOdel X5`

Select a model with an x^5 component:
`FNY=FNY+X5*X**5`.

`MOdel POWR`

Select a model with a power-law component:
`FNY=FNY+PN*X**IN`.

`MOdel SIN`

Select a model with a sinusoidal component:
`FNY=FNY+SN*SIN(2*PI*(X-PH)/PE)`.

MOdel GAUS

Select a model with a gaussian component:
FNY=FNY+GN*EXP(-Z*Z/2.),
where Z=(X-GC)/GW and with integral SQRT(2*PI)*GN*GW.

MOdel EXP

Select a model with an exponential component:
FNY=FNY+EN*EXP(-(X-EC)/EW).

MOdel AEXP

Select a model with a symmetric exponential component ($e^{-|x|}$ for all x):
FNY=FNY+EN*EXP(-ABS(X-EC)/EW).

MOdel BURS

Select a model with a burst component (linear rise followed by an exponential decay):
FNY=FNY+0 for X<ST;
FNY=FNY+BN*(X-ST)/(PT-ST) for ST<X<PT; and
FNY=FNY+BN*EXP(-(X-PT)/DT) for PT<X.

MOdel SBUR

Select a model with a smooth-burst component:
FNY=FNY+BN*(T**RR)*EXP(-(X-TS)/DT),
where T=EXP(1)*(X-TS)/(RR*DT), such that SBUR = BN at the peak.

MOdel PEAR

Select a model with a Pearson-function component:
FNY=FNY+K*(F1**M1)*(F2**M2),
where F1=[1.+(X-X0)/A1] and F2=[1.-(X-X0)*M1/(A1*M2)].

MOdel WIND

Select a model with a window-function component:
FNY=FNY+LE for T1<X<T2; and
FNY=FNY+0 otherwise.

MOdel KING

Select a model with a King-profile component:
FNY=FNY+S0*(1.+(X/RC)**2)**(-IN).

MOdel LORE

Select a model with a Lorentz-profile component:
FNY=FNY+LN/(1.+[2.*(X-LC)/LW]**2),
with integral PI*LN*LW/2.

MOdel SPLN #

Select a #-knot spline component[1]. The number of knots defaults to 2, which generates a straight line.

For unconstrained y values, the natural spline condition, which sets $y'' = 0$ at the boundaries is imposed. You may not extrapolate this function outside the interval fitted.

It is possible to impose a periodic boundary condition on the spline curve. To do this, constrain the y position of the last knot to be the same as the first. When this constraint is detected, the program automatically forces the first derivatives to match at the two boundaries. For this case, you are allowed to access the function outside the interval fitted. However, the function is assumed to be periodic, with the period given by the difference in x between the first and last knots.

For example, `MOdel SPLN 5` will generate a 5-knot spline (10 parameters). The spline can be added to other models; thus `MOdel SPLN 5 GA` would add a 5-knot spline to a gaussian. Hence, the spline would model the 'background' and the gaussian, a 'line'.

It is possible for the x position of two knots to lie between two adjacent data points. This results in a local χ^2 minimum as the lower knot adjusts to fit data below it, the upper knot adjusts to fit data above it. A strong wiggle occurs between the two knots but since there are no data points there, χ^2 is not affected. In this section, two knots very close to each other will be called a collision. If collision occurs during a fit, then convergence will be very slow.

One method to greatly reduce the number of collisions is to first fit the y locations before attempting to fit the x locations. By default, the knots are evenly spaced in the x direction and are not allowed to vary. For the first fit you should leave the x positions frozen, although you can move the knots (using `Newpar`) to concentrate them where the function is changing rapidly. Once a reasonable set of y positions is determined you can then thaw the x positions and re-fit. You should never thaw the end points: They determine the range over which the spline is to be evaluated.

With the above recipe, collisions can still occur. The straight-forward method to separate the knots is: Use `Newpar` to re-position the two knots, freeze the x locations, and then re-fit. After this the knots will sometimes stay separated when you thaw their positions and re-fit. The trick is to force the knots far enough apart so that they will not be attracted to the local minimum, but not so far apart as to grossly distort the fit.

Sometimes two knots collide when you are trying to fit the data with too few knots. This case can be easily tested for by increasing the number of knots and re-fitting.

`MOdel AKIM #`

Select a #-knot Akima component[2]. An Akima component is very similar to `SPLN` in that both use a cubic function to interpolate between the knots. Akima's method does not introduce false extrema and inflection points as does the cubic spline and therefore, is far superior for data that show abrupt transitions.

Like `SPLN` two different boundary conditions are allowed. If the last y value is unconstrained, then the code uses 'virtual' knots outside the boundaries to determine

[1] The spline is computed by solving equation 3.3.7 in the 1988 edition of *Numerical Recipes*, by Press, Flannery, Teukolsky, and Vetterling.

[2] Details of Akima's method can be found in "A New Method of Interpolations and Smooth Curve Fitting Based on Local Procedures" by Hiroshi Akima in J. of the Ass. for Computing Machinery, (1970) 17, pp. 589-602. An implementation is described in PPC Journal, (1985) 12, No. 10, pp. 11-14.

the function at the boundaries. The locations of the virtual knots mirror the location of the knots just inside the boundaries. If the y position of the last knot is constrained to match the y position of the first knot then a periodic boundary condition is imposed.

`MOdel DEMO`

Call the Fortran user-defined component. Chapter 6 describes how to create how to write a Fortran function that can be linked in to PLT to replace the `DEMO` component.

`MOdel $codfile`

Call the user-defined COD (COmponent Definition) function found in `codfile.COD`. Briefly, a COD function is a program written in a Forth-like computer language. To understand COD, read the documentation or help file for COD. A COD file can be added to any combination of built-in components. For example, the model specified by `MOdel CONS LINR $TEST` would calculate the sum of a constant term, a linear term, and the value of the COD function contained in the file `TEST.COD`.

At the present time only one COD function can be defined in a model, although this function can be referenced more than once. If you wish to combine two COD functions, you will need to write a third function that combines the first two.

COD should be used for all simple components that cannot be expressed by adding together the built-in components. Since a COD function is interpreted, it will run slower than the user-defined component. However, since COD is highly efficient and supports many mathematical functions, it is expected that the interpreter will be good enough for most purposes. For large numbers of points ($> 10^4$) or models that involve reading a disk file, the user is advised to write a Fortran function using the user component.

Example:

```
: GAUS ! The file must contain a : followed by a dummy name
X      ! Push current value of X onto the stack
X      ! Push current value of X onto the stack
*      ! Multiply the top two numbers on the stack to get X*X
P1     ! Push the value of parameter 1 onto the stack
*      ! Multiply to get P1*X*X
NEG    ! Negate the number on the top of the stack (-P1*X*X)
EXP    ! Calculate EXP of -P1*X*X
P2     ! Push the value of parameter 2 onto the stack
*      ! Multiply to get P2*EXP(-P1*X*X)
;      ! The function must end with a ; character
```

This simple COD function (`GAUS.COD`) contains two parameters and calculates the value of `P2*EXP(-P1*X*X)`. It could be written much more concisely as

```
: GAUS X X * P1 * NEG EXP P2 * ;
```

Newpar

`Newpar`

Display the values associated with all of the parameters and allow the user to change them. If you wish to display the parameter values without changing them, then use the `WModel` command.

`Newpar #`

Display the values associated with the parameter specified by the first argument and allow the user to change them.

`Newpar # #`

Change the value of the specified parameter to the value you entered in the second (and following) arguments. You will not be shown the original values.

Examples:

```
PLT> Newp 2      ! Prompt for new values of parameter 2
  2, GW: VAL( 1.00), SIG( 0.00 ), PLO( 0.00), PHI( 0.00)?
3.               ! Value of parameter 2 is now set to 3
PLT> Newp 3 10   ! Value of parameter 3 is now set to 10
PLT> Newp 6,,-1  ! Freeze value of parameter 6
```

Plot

`Plot`

Cause the plot to be redrawn on the graphics device.

`Plot All`

Cause all data points, including those flagged as no-data, to be plotted.

`Plot Good`

Undo the effects of the `Plot All` command and prevent plotting of points flagged as no-data (default).

`Plot Vertical`

Plot up to 20 plot groups in separate panels, in a vertical stack.

`Plot Overlay`

Plot all groups in a single panel (default).

`Plot Zero ON`

Cause the plot groups that have color index zero to be plotted with the background color. This is sometimes useful for erasing plots.

`Plot Zero OFf`

Do not plot groups with color index zero (default). This is much faster than plotting with the background color.

PRompt

`PRompt $`

Redefine the "`PLT>`" prompt.

Rescale

`Rescale X [#] [#]`

Reset XMIN and XMAX in the current window to the values specified. If both XMIN and XMAX are omitted, then PLT will reset the range to the default.

`Rescale Y [#] [#]`

Reset YMIN and YMAX in the current window to the values specified. If both YMIN and YMAX are omitted, then PLT will reset the range to the default.

`Rescale [#] [#] [#] [#]`

Reset XMIN, XMAX, YMIN, and YMAX in the current window to the values specified. If all four numbers are omitted, then PLT will reset both the x- and y-ranges to the defaults.

`Rescale ?`

Display the current XMIN, XMAX, YMIN, and YMAX values for each window.

Note: For Vertical plots each plot group can be specified separately; thus `R Y1` will rescale the y-range in window 1 and `R Y4` will rescale the y-range in window 4.

SCr

`SCr # # # #`

Immediately change the color representation for the specified color index. The first number is the color index and the following three numbers give the red, green, and blue color intensities and must lie in the range 0.0 to 1.0. This command only works on color devices for which the color representation can be changed.

Examples:

```
PLT> SCR 0 1. 1. 1.  ! Set the background color to be white
PLT> SCR 1 0 0 0     ! Plot color index 1 in black
PLT> SCR 2 0 1 0     ! Plot color index 2 in green
```

SHow

`SHow Contour`

Display on your terminal the PLT commands to generate the current contour plot (if any).

`SHow Group`

Display on your terminal information about each plot group.

`SHow Internal`

Display on your terminal the values of various PLT internal variables.

SKip

Most PLT commands operate on 'groups' of data points. The default is for each vector of the input data to be in a separate group. For the default mode, commands such as `COlor` and `MArker` affect the appearance of an entire vector. Using the `SKip` command, it is possible to independently control the appearance of sub-sets of data within a single vector. Thus, when using `SKip`, a single y vector can be divided into several plot groups that can be independently controlled with `COlor`, `MArker`, `R Y1`, *etc.* Currently, `SKip` should only be used when the input data consists of two vectors as the other vectors will not be plotted. Note, `SKip` cannot affect whether a data point has an error associated with it.

`SKip OFf`

Each vector of the input data is plotted as a separate group (default).

`SKip Single`

A new plotting group begins every time x-coordinate equals `NO` (the no data flag) ONE or more consecutive times. `SKip Single` is useful when you wish to plot different groups with different markers.

`SKip Double`

A new plotting group begins every time the x-coordinate equals `NO` (the no data flag) TWO or more consecutive times. `SKip Double` is useful when you wish to plot different groups using lines that may contain breaks.

Example: Consider the QDP file:

```
1  2
2  1
NO NO
3  4
4  3
NO NO
NO NO
5  6
6  5
```

With the default `SKip Off`, the above will be plotted as two groups each containing nine points. Using `SKip Single`, would cause the above data to be divided into three groups. The first group would consist of the data in the first three lines, the second would come from lines four to seven, and the third group from lines eight and nine. Using `Skip Double` would cause the above data to be plotted as two groups with lines one to seven being in the first group, and the last two lines making the second group.

STatistics

`STatistics [fgroup]`

where `[fgroup]` is the default group for fitting. This command causes a short table displaying some basic statistical properties about that group of data to be printed on the terminal. The first line tells you which group is fitted and over what range. Next the unweighted average, variance, and 3rd moment are displayed. For the unweighted data, the column labeled SUMW contains the total number of points used in the calculation, for weighted data, SUMW is the sum of the weights. YMIN and YMAX are the minimum and maximum data values in the range. If the plot group has errors associated with it, then weighted values of the average, variance, and 3rd moment will be displayed. The next row contains WCHI and WRED which are the χ^2 and reduced χ^2. The W is appended to remind you that the actual errors on the data were used. The line labeled Sum of Y*XDEL contains the sum of the y values times the Δx values, where Δx is given by the x-error bars. This is a rectangle rule integral of the data. The last line gives the (unweighted) linear correlation coefficient of the y vs. x data.

For maximum accuracy, this routine makes two passes through the data, once to calculate the average, and the second time to calculate moments based on the difference between the data and the average.

If you do not understand the difference between the unweighted and weighted values then you should use unweighted quantities.

Example:

```
PLT> STat
Group  2, from   430.0    , to   540.0

        YBAR        YVAR        Y3M         SUMW       YMIN         YMAX
UNWTD  0.5915      0.1341     -8.3771E-03   23.00     2.2100E-02   1.096
  WTD  7.8728E-02  2.0009E-02  1.2435E-02  4.9056E+05
 WCHI=  9.389E+03,  WRED=  426.768

     Sum of Y*XDEL=   0.883411
Correlation coeff.=  -0.853596
```

THaw

THaw [plist]

Can only be used after a model has been defined. This command causes all parameters listed in [plist] to be thawed, which means the parameter value will be allowed to vary when fitting.

Examples:

```
PLT> THaw 2         ! Cause parameter 2 to be thawed
PLT> THaw 3..5 9    ! Cause parameters 3 through 5 and 9 to be thawed
```

Time

Time ON

Cause the date and time to be plotted in the lower right corner (default).

Time OFf

Remove the date and time from future plots.

Uncertain

`Uncertain [# [# [#]]]`

Vary the specified parameter(s) in order to estimate their uncertainties. Each specified parameter, in turn, is stepped and χ^2 is minimized. Stepping stops when the requested value of $\Delta\chi^2$ is obtained. `Uncertain` can take up to 3 numbers as arguments. If one number contains a decimal point, then that number is interpreted as the requested value of $\Delta\chi^2$ (which for the first time defaults to 2.7 and for later occasions defaults to the previous value). The remaining two numbers in the `Uncertain` command specify the lower and upper parameter numbers for which you want to estimate the error. If only one number is given (without a decimal point), then the error is generated only for a single parameter. If errors are currently turned off (or do not exist), then the routine works out a correction factor that converts the W-VAR to χ^2.

UPper

`UPper # ON [glist]`

If a number in one of the groups specified in glist is less than `#` sigma from zero, then plot that number as a `#`-sigma upper limit.

Example:

```
PLT> UPper 2.7 ON 3   ! When plotting group 3, all numbers within 2.7
PLT>                  ! sigma of zero will be plotted as an upper limit.
```

VErsion

`VErsion`

Return the date of last modification to the current version of PLT.

Viewport

`Viewport  #,[#,[#,#]]`

Control location of the viewport in normalized device coordinates, where (0.0,0.0) is the bottom left corner and (1.0,1.0) the top right corner. The default viewport is 0.1 0.1 0.9 0.9, with the first two numbers giving the location of the bottom left corner and the next two numbers, the upper right corner. If you use `Viewport` with and only specify two numbers then PLT centers the viewport about the center of the plot, thus `View x,y` is the same as `View x,y,1.0-x,1.0-y`. If you do not wish to center the viewport, then you can specify all four numbers, where the last two numbers refer to the top right corner.

Examples:

```
PLT> View .4 .4        ! Viewport extends from (.4,.4) to (.6,.6)
PLT> View .8 .8 .9 .9  ! Use small viewport in top right corner
PLT> View .1 .1        ! Go back to the default viewport
```

WData

`WData [$]`

Write all data between the current x-scale minimum and maximum to a QDP file. If you want all the data to be written to the file then you should use the `R X` command to reset current scale to include the minimum and maximum data values. A blank file name will cause the data to be written to your current terminal screen.

The `WData` command will not write any PLT commands to the file. However, it will include a reference to an indirect file. For example, `WData TEST` will create a file called `TEST.QDP` that includes the line `@TEST`. The PLT command `WHead` can be used to create a `TEST.PCO` file that contains all the PLT commands needed to re-create the current plot.

`WData [$] #`

Write the data with only `#` digits of accuracy (numbers will be rounded). If `#` is negative, the error on a number is written out to (-`#`) number of digits and the number itself is written to the same accuracy.

Examples:

```
PLT> WData          ! Write the data to the terminal screen
PLT> WData TEST     ! Write the data to TEST.QDP
PLT> WData TEST 3   ! Write the data (3 significant digits) to TEST.QDP
PLT> WData TEST -2  ! E.g., 123.758 +/- 2.698 will be written 123.8 2.7
PLT> WData,,-2      ! As above, but written to the terminal screen.
```

WEnviron

`WEnviron [$]`

This command does the same thing as if you entered a `WHead` command followed by a `WData` command. This command should be used if you want to save both the current data and the PLT commands needed to re-create the current plot.

Examples:

```
PLT> WEnv          ! Write commands and data to the terminal screen
PLT> WEnv TEST     ! Create TEST.PCO and TEST.QDP files
PLT> WEnv TEST 3   ! Write the data (3 significant digits) to TEST.QDP
PLT> WEnv TEST -2  ! E.g., 123.758 +/- 2.698 will be written 123.8 2.7
PLT> WEnv,,-2      ! As above, but written to the terminal screen.
```

WHead

`WHead [$]`

This command only writes the list of PLT commands needed to create the current figure. Since this command will NOT write any data, it will run faster than the `WEnviron` command. Typically one would first use `WEnviron` to write both the PLT commands and the data to files. If any changes are made to the appearance of the plot (such as adding labels, *etc.*) then the `WHead` command can be used to update the PLT command file without over-writing the QDP file containing the data.

Examples:

```
PLT> WHead          ! Write commands to the terminal screen
PLT> WHead TEST     ! Write commands to TEST.PCO
PLT> WHead TEST 3   ! Same as previous (the 3 is ignored)
```

WIndow

`WIndow #`

This command sets the currently active window to be the number specified by `#`. After the window command has been issued, commands such as `Rescale X`, `LA Y` will affect the currently active window. For maximum compatibility, a `Plot Vertical` command creates N windows numbered by the number of the plot group that they contain. Thus if group 1 is used on the x-axis, then the upper (first) window plotted will contain plot group 2 and will be plotted in window 2.

WModel

`WModel [$]`

Write the current model into the named file. The model written out can later be read with the `MOdel @filename` command. If you do not enter a file name, the model will be written to your terminal screen. Since all significant digits are written, writing a file provides a good way to save your current `MOdel` parameters. If you have previously `Fit` the data then this command will write two additional lines at the end of the model file as comments. These lines are the WVAR and NBIN obtained in the most recent fit.

Xaxis

`Xaxis #`

Causes the plot group `#` to be used as the x variable. Thus `Xaxis 3` will cause plot groups 1,2 (and any >3) to be plotted as a function of group 3. A `LAbel Y3` command will associate a label with the third plot group and will then appear as the x-axis label.

`Xaxis Linear # #`

Cause the x variable to be a linear function. Thus, the command `Xaxis Linear 10. 1` would cause the first point to be plotted at x=10, the second point at x=11, the third at x=12, *etc.*

Yaxis

`Yaxis [ON] [glist]`

where `[glist]` is a list of plot groups. This command is more intuitive to naive users than the `COlor ON` command. This command causes the plot groups specified in `[glist]` to turned on (plotted) in the currently active window.

`Yaxis Lin # #`

Specify the y-axis scale to be used for contour plots.

Example: Assume you have a 10 by 10 array of data, then the commands then

```
PLT> Xax L 10 1  ! Would cause the X values to range from 10 to 19
PLT> Yax L 5 1   ! Would cause the Y values to range from 5 to 14
PLT> CON Lev 1,2,3 ! Draws a contour plot with these scales
```

$

`$ [command]`

Spawn to the operating system, where `[command]` is an operating system command. If no command is specified, then a system shell is created that will allow you to enter several commands until you logout (under VMS) or exit (under UNIX or DOS).

Examples:

```
PLT> $         ! Spawn to system (assume VMS)
Spawning...  ! Wait for something to happen
$ (enter VMS commands)
$ LO
PLT>         ! You have now returned to PLT
PLT> $ DIR   ! This will display your current directory
Spawning...  ! Wait for something to happen
(directory appears here)
PLT>         ! and you are left in PLT.
```

@

`@ $`

Execute commands from an indirect command file. Command files can be nested to a depth of 10. The default file type is `.PCO` (for Plt COmmands).

Example:

```
PLT> @NICE   ! Execute the PLT commands in the file NICE.PCO
```

Appendix C

QDP Command summary

QDP commands must be inserted at the beginning of a QDP file, as these commands tell QDP how to read in the data. Any command not recognized by QDP is passed to PLT. QDP separates command lines from data lines based on the first non-blank character in the line. If this character is `+` , `-` , `.` , or a digit, then the entire line will be read as data.

READ Serr

`READ Serr [vlist]`

Tell QDP/PLT which vectors have symmetric errors. The command `READ Serr 1 3 5` will cause vectors 1, 3, and 5 to be read with symmetric errors, and vectors 2 and 4 to be read without. Only one `READ Serr` command should appear in a QDP file.

Example:

```
READ Serr 1 3 5
 1. .1     2.    3. .3    4.   5. .5
```

This would be read as 5 vectors: 1.0 ± 0.1, 2.0 (no error), 3.0 ± 0.3, 4.0 (no error), and 5.0 ± 0.5. Without the `READ Serr` command, it would be read as 8 vectors.

READ Terr

`READ Terr [vlist]`

Tell QDP/PLT which vectors have two-sided errors. It takes three columns to specify a vector with two-sided errors. The first column is the central value, the second column (which must be positive) specifies the upper bound, and the third column (which must be negative or zero) specifies the lower bound.

Example:

```
READ Serr 1
READ Terr 2
 1. .1    2. +.1 -.2
```

This would be read as 1.0 ± 0.1 and $2.0^{+0.1}_{-0.2}$. Note: In fitting, non-positive errors are ignored; thus the first error of two-sided errors should be positive.