S-PLUS 4.x complements to

Modern Applied Statistics with S-Plus

Second edition

by

W. N. Venables and B. D. Ripley

Springer (1997). ISBN 0-387-98214-0

3 May 1999

These complements have been produced to supplement the second edition of MASS. They will be updated from time to time. The definitive source is http://www.stats.ox.ac.uk/pub/MASS2/.

Introduction to 4.x Complements

These complements are made available on-line to supplement the book with a description of the new user interface in S-PLUS 4.x for Windows. There are other complements covering other aspects of S-PLUS, and these include comments on changes in version 4.x. In particular, the programming complements discuss the use of compiled C and FORTRAN code.

In this document 4.x refers to S-PLUS 4.0 releases 1, 2 and 3, S-PLUS 4.5 Professional Edition and any later versions in the same family such as S-PLUS 2000 Professional Edition. S-PLUS Standard Edition has much more limited features; in particular no access to the S language except through the GUI.

To forestall any possible confusion: there is no connection between version 4.x of S-PLUS and the version 4 of S which has been announced as the basis for future versions of S-PLUS for both Unix (S-PLUS 5.x) and Windows.

The chapter and section numbering in these complements corresponds to that in the book.

There are separate Complements documents for S programming and for statistical methods available from `http://www.stats.ox.ac.uk/pub/MASS2/`. In particular, the statistics complements cover the use of version 3 of the `nlme` library that is incorporated in S-PLUS 2000.

Contents

Chapter 1

Introduction

1.3 Using S-PLUS under Windows

Getting started

The ways to use a specific `_Data` directory have been changed. We have not tried S-PLUS 4.x under Windows 3.x or NT3.51.

Windows 95 *and* NT4.0

1. Create a new folder, say `SWS`, for this project, then (optional) create new folders `_Data` and `_Prefs` within that folder.
2. Copy any data files you need to use with S-PLUS to the folder `SWS`.
3. From the Start menu, select Settings, Taskbar, the Start Menu Programs page and click on the Advanced button.
4. Open the S-PLUS 4.x folder under Programs[1].
5. Create a duplicate copy of the S-PLUS 4.x icon for example, using Copy from the Edit menu. Change the name of this icon to reflect the project it will be used for.
6. Right-click on the new icon, and select Properties from the pop-up menu.
7. On the page labelled Shortcut, add at the end of the Target field `S_PROJ=` followed by the complete path to your folder. If that path contains spaces, enclose it is double quotes, as in

   ```
   S_PROJ="c:\my work\S-Plus project"
   ```

   If you have other files for the project, set the Start in field to their folder. If this is the same place as `_Data`, you can set `S_PROJ=.` on the target rather than repeating the path there.
8. Select the project's S-PLUS icon from the Start menu tree.

[1] Under NT4 the folder may not be there if S-PLUS was installed under another account. In that case you need to find the S-PLUS home directory and make a shortcut to `cmd\splus.exe` in Programs.

9. If you did not create them both at step 1, you will be asked if `_Data` and `_Prefs` directories should be created. Click on OK.
10. When the program has initialized, click on the Commands Window button with icon  on the upper toolbar. If you always want a Commands Window at startup, select this from the menus via Options | General Settings... | Startup. (This setting is saved for the project on exiting the program.)

More details of this are given in the release notes (file `README.TXT` in `SHOME`) and in Chapter 15 of the *Programmer's* Guide.

Recovering from crashes

We have experienced a small number of crashes after which S-PLUS 4.x would not restart. In all cases the problem was with the state of the information in the working directory, and the following procedure allowed us to recover.

1. Locate the working directory, which contains a `_Data` and a `_Prefs` directory. We use `C:\Swork` in the examples; replace this by the appropriate directory.
2. Delete the files `__sum4.txt`, `__sum4i.txt` and `__sum4.tx_` in `_Data` if these exist, and try to restart S-PLUS. If this works, S-PLUS will create new versions of these files as needed.
3. Rename `_Prefs` to `_Prefs.bak`, make a new `_Prefs` directory and restart S-PLUS. If this is successful you will need to reset your preferences, and you may be able to copy crucial files from `_Prefs.bak` to `_Prefs`. We keep a backup copy of a working `_Prefs` for use when we wish to reset preferences to a known state.
4. If these fail the problem is probably a file such as `last.dump` in `_Data`. Experts can delete this file and the corresponding lines in `_Data\___nonfi`. A more cautious approach is to rename `_Data` to `_Data.old` and create a new `_Data` directory. Then restart S-PLUS, and use

```
attach("C:/Swork/_Data.old")
mydata <- mydata
```

to copy over crucial files such as `mydata`. This process can also be used to delete files[2] within `_Data.old` that have become corrupted (and their modification times will reveal which these might be). Another possibility is to start up another project, `attach` the corrupted working directory and use `remove` to delete the corrputed files.

5. It may be possible to restore either or both of `_Prefs` and `_Data` from their backups made during this process, once the problem has been pin-pointed.

[2] using `remove("name", w=2)`.

The S-PLUS 4.x interface

S-PLUS 4.x provides a new interface to the S-PLUS engine, and many new actions. The GUI is highly configurable, but in its default state looks similar to Figure 1.10. The top toolbar is constant, but the second toolbar and the menu items depend on the type of subwindow which has focus.

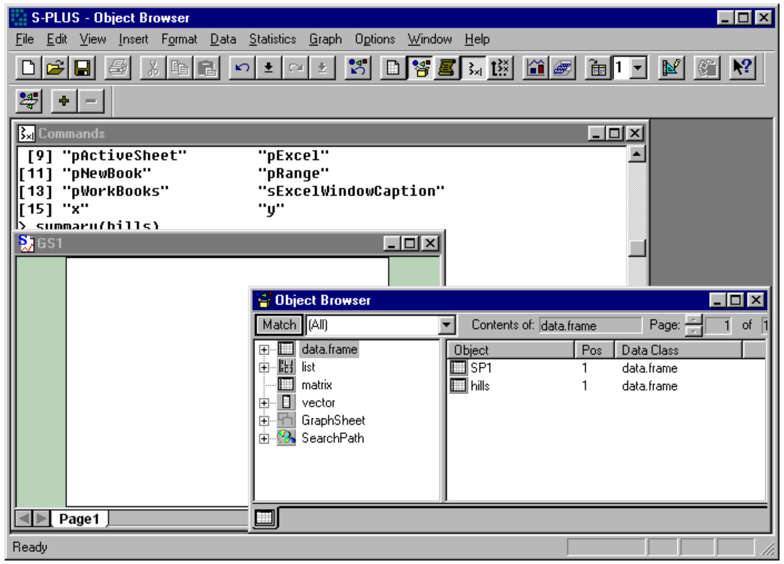


Figure 1.10: A snapshot of the interface of S-PLUS for Windows 4.0 showing three subwindows, from front to back an object browser, a graphsheet and a commands window.

The object browser and the commands window were selected by the two buttons on the top toolbar which are depressed. (By default there is an object browser but no commands windows.) To find out what the buttons mean, hover the mouse pointer over them and read the description in the bottom bar of the main S-PLUS window.

The commands window

To use this version of S-PLUS in the same way as earlier versions (or S-PLUS under Unix), type commands in the commands window. These are executed immediately. Previous commands can be recalled by using the up and down arrow keys, and edited before submission (by pressing the return key). The commands history button (immediately to the right of the command window button) brings up a dialog with a list of the last few commands, which can be selected and re-submitted. When the commands window has focus, the second toolbar has just one button (with icon representing a pair of axes and a linear plot). When depressed, this selects editable graphics. This is not recommended for routine use, as it may make the graphics very slow, and plots can be made editable later (see under 'object browsers' below). It is also possible to launch a graphsheet with editable graphics or not from the command line by

```
graphsheet(object.mode="object-oriented")
graphsheet(object.mode="fast")
```

Script windows

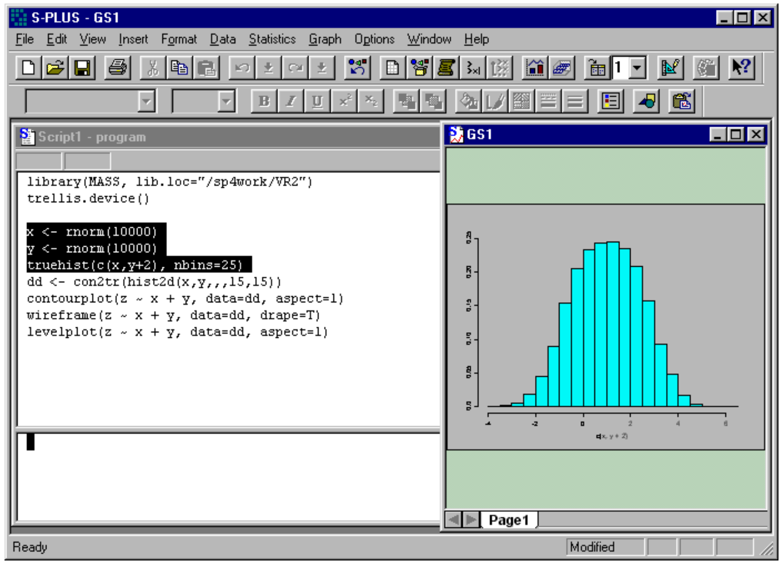


Figure 1.11: A script subwindow in S-PLUS for Windows 4.0.

There is another way to use this version of S-PLUS for S programming, which is to use a *script* window. This can be opened from the New file button or menu item, and presents a two-part subwindow as shown in Figure 1.11. S commands can be typed into the top window and edited there. Groups of commands can be selected (in the usual ways in Windows), and submitted by pressing the function key F10 or by the leftmost button on the second line (marked to represent a 'play' key). If text output is produced, this will appear in the bottom part of the subwindow. A script window can be launched from the command line to edit a function by the new function `Edit` (note the difference from `edit`).

It is the help features that mark a scripts window as different from a commands window. Select a function name by double-clicking on it. Then help on that function is available by pressing the function key F1, and the right-click menu has items Show Dialog... and Expand Inplace to pop-up a dialog box for the arguments of the function and to paste in the function body.

Scripts can be saved as text files with extension `.ssc`; use the Save file menu item or button when the script window has focus.

More than one script window can be open at once. To avoid cluttering the screen script windows can be hidden (and unhidden) from the Windows file menu. The Hide item hides the window which has focus, whereas the Unhide... provides a list of windows from which to select.

Object browsers

There can be one or more object browsers on screen. They provide a two-panel view which will be familiar from many Windows programs. Objects are grouped by class and can be expanded down to component level. The right-click menu is context-sensitive: for example for a linear model fit (of class `lm`) has Summary, Plot, Predict and Coefficients items. Double-clicking on a data frame or vector will open it for editing or viewing in a spreadsheet-like *data window*.

If a graphsheet is selected and expanded it will first show its pages (if there are more than one) and then the plotted objects. If the object is labelled CompositeObject then the right-click menu will include the item Convert to Objects which will convert that plot to editable form.

Object browsers are highly customizable, both in the amount of detail in the right pane and in the databases and classes of objects to be shown. Right-clicking on the background of the left and right panes or the Format menu will lead to dialog boxes to customize the format.

The ordering of items in the right pane can be puzzling: click on the heading of a column to sort on that column (as in the Explorer in Windows 95 and NT4).

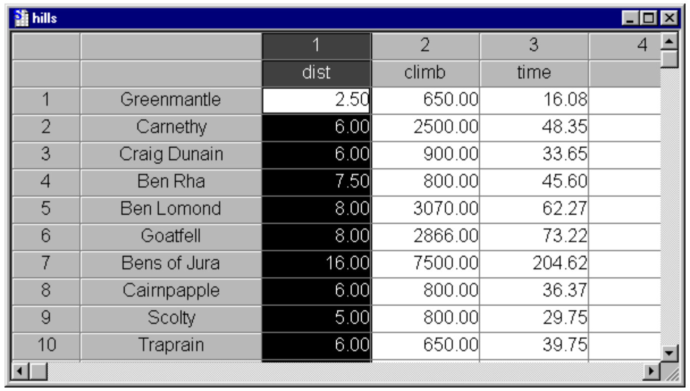
Data windows

A data window provides a spreadsheet-like view (Figure 1.12) of a data frame (or vector or matrix). The scrollbars scroll the table, but the headings remain visible. A region can be selected by dragging (and extended by shift-clicking); including the headings in the selection includes the whole row or column as appropriate.

Entries can be edited and rows and columns inserted or deleted in the usual spreadsheet styles. Toolbar buttons are provided for most of these operations, and for sorting by the selected column. Double-clicking in the top row of a column brings up a format dialog for that column: double-clicking in the top left cell brings up a format dialog for the window that allows the type font and size to be altered.

1.4 An introductory session

We can do something similar to the analysis of the `hills` and `michelson` datasets using the GUI. First attach library `MASS` via the commands window or a script window (this cannot be done from the GUI, although databases can be attached).



		1	2	3	4
		dist	climb	time	
1	Greenmantle	2.50	650.00	16.08	
2	Carnethy	6.00	2500.00	48.35	
3	Craig Dunain	6.00	900.00	33.65	
4	Ben Rha	7.50	800.00	45.60	
5	Ben Lomond	8.00	3070.00	62.27	
6	Goatfell	8.00	2866.00	73.22	
7	Bens of Jura	16.00	7500.00	204.62	
8	Cairnpapple	6.00	800.00	36.37	
9	Scolty	5.00	800.00	29.75	
10	Traprain	6.00	650.00	39.75	

Figure 1.12: A data window view of the `hills` dataset with the `dist` column selected.

Open an object browser, create a new page (from the right-click menu), filter on the location of the `MASS` library, select all classes and click on OK. Select `data.frame` in the left pane, and click on the `Object` header in the right pane to sort the items alphabetically. Finally select the `hills` data frame. Click on the 2D plot button on the top toolbar, and select the button for a scatterplot matrix. This gives a plot similar to Figure 1.4, but without the square aspect ratio. To change that, select the plot region (click outside the scatterplot matrix), right-click and select the Position/Size... item. In the dialog box enter equal dimensions for width and height of the plot display size.

To try out the brush-and-spin plot, select Brush and Spin from the Graph menu, select all the variables in the dialog box and click on OK.

To approximate Figure 1.6, double click on `hills` to open a data window, and select the columns `dist` and `time` in that order. Then open the 2D plots palette and select the Linear Fit button. Then to label points, open the annotations palette (second from right on the graphsheet toolbar), and select the Label Points button (third on the top row). Click near a point on the plot for a label, but shift-click for this label to be permanent so that further points can be labelled. To add the `ltsreg` line, we need to add another *plot*, which we can do by first selecting the existing plot and then shift-clicking on the appropriate button, this time for a robust fit. Finally we need to select the robust line and from the dialog box brought up by the lines shortcut on the right-click menu select a dashed line. The numerical labels on the y-axis are horizontal so overlap the axis label. There seems to be no way to rotate the numbers, so the axis label will need to be selected and moved.

Now return to the object browser, expand the list of data frames and select `michelson`. The right pane will then list the columns: select `Expt` and `Speed` in that order. (Hold down Ctrl whilst clicking on `Speed`.) Then open the 2D plots palette and click on the Box Plot button. The plot will be generated. Select the

x-axis label and edit the text (either 'in place' by double clicking or via the dialog box). Finally, open the annotations palette, select the Comment tool, and add the title.

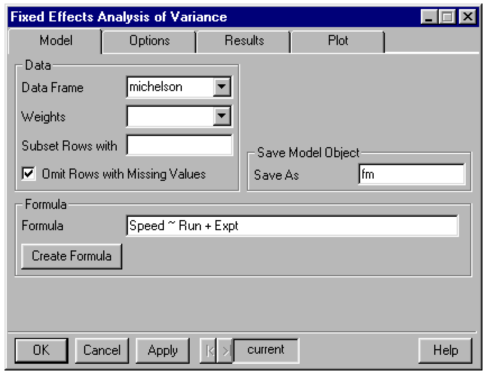


Figure 1.13: The dialog box for a fixed-effects analysis of variance.

Select the `michelson` data frame again, and select `Speed`, `Run` and `Expt` in that order. Then select Statistics, Analysis of Variance and Fixed Effects from the menus. A dialog box (Figure 1.13) will appear, with the appropriate formula already constructed. Give `fm` as the name by which to save the model object. Clicking on Apply gives output in the results window, including the analysis of variance table. Now delete `Run +` from the formula, change the name to `fm0` and click on OK.

Go back to the object browser, select the first page and the class `list`. Objects `fm` and `fm0` will have appeared. Right clicking on these will allow them to be summarized and plotted. (Plotting the object will give four pages of diagnostic plots.)

To compare the models, select Compare Models from the Statistics menu. Select models `fm0`, `fm` in that order[3] from the list of models presented in the dialog box, and select OK. The analysis of variance table appears in the report window.

[3] This will only work in release 2 or later of 4.0: release 1 ignores the order of selection.

Chapter 2

The S Language

2.4 Reading data

The function `scan`

It is not possible in some releases of 4.0 to paste data into a command window to be read by `scan`. However, there is a better alternative: copy the data as before, and use `scan("clipboard", ...)`. This also works in 3.3.

2.12 History and audit trails

Auditing was enabled in 4.0 release 1, apparently by mistake. It is disabled by default in 4.0 releases 2 and later, but can be enabled by setting

```
Splus.exe S_NOAUDIT=
```

via the properties of the S-PLUS icon for the project.

2.13 BATCH operation

It is sometimes desirable to run an S-PLUS job non-interactively. S-PLUS 4.x provides a command-line interface program `sqpe.exe` whose input and output can be redirected and which can be used from an MS-DOS window in Windows 95 or in other shells. Error messages are written to the output stream, but commands are not echoed unless `options(echo=T)` is set. The prompts will not appear in the output file unless the environment variable `ALWAYS_PROMPT` is set. Before using `sqpe` ensure that the environment variable `SHOME` is set: an example of its use (in an MS-DOS window) is

```
set ALWAYS_PROMPT=T
set SHOME=C:\Program Files\splus4
%SHOME%\cmd\sqpe < infile > outfile
```


Either the current directory should contain a `_Data` directory, or the environment variable `S_WORK` should point to a directory that does.

The main S-PLUS program has a BATCH switch with syntax

```
Splus [S_PROJ=dir] /BATCH infile [[outfile] errfile]
```

where the error messages are written to `outfile` if `errfile` is not supplied. Setting `S_PROJ` is needed to change the working directory `_Data` from its default. The input commands are not echoed (use `options(echo=T)`) and the prompts will not appear in the output file unless the environment variable `ALWAYS_PROMPT` is set. The files are specified relative to the working directory.

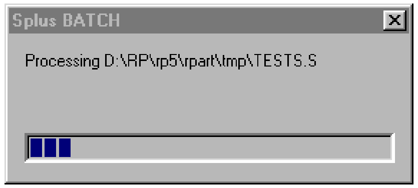


Figure 2.3: The BATCH progress box.

From 4.0 release 3 it is possible to ask for the progress dialog box (similar to Figure 2.3) to be minimized[1] by using the command line (all on one line)

```
Splus [S_PROJ=dir] /BATCH_PROMPTS progress:min /BATCH
        infile [[outfile] errfile]
```

This minimizes the progress dialog as a floating window rather than to the taskbar. In 4.5 you can also use `/BATCH_PROMPTS no` to turn off the dialogs completely, or `/BATCH_PROMPTS progress:no` to suppress the progress dialog box only.

Using `Splus /BATCH` needs all the resources of the GUI, and so takes a similar time to start and uses a similar amount of memory. Using `Sqpe.exe` provides much more efficient (faster, using less memory) access to the S-PLUS engine provided that access to graphsheets or `win.printer` is not required.

[1] it has no minimize control

Chapter 3

Graphical Output

3.3 Enhancing plots

Adding information

Mathematics in labels

Under S-PLUS 4.x there is a series of escape codes which can be used to enhance graph labels on graphsheets. These are

`‘string‘`	*string* in italics
`#string#`	**string** in bold
`x[2]`	x^2, superscript
`x]2[`	x_2, subscript
`\`n	change to font n (the normal font being 0)
`|`n`|`	change to colour n
`~xyz`	character with ASCII code `xyz`

These escape codes can be escaped by preceding them by `@`. The special characters depend on the font selected: use the Character Map accessory to see what is where in each font. In particular, the Greek letters (including variant forms) are in positions `A--Z` and `a--z` of the Symbol font (normally font 1).

Note that these escapes are normally disabled for graphics created by commands from a commands or script window, but can be used by editing text in such graphics.

3.6 Object-oriented editable graphics

[This section is (mainly) reproduced from the book for convenience. There are some added graphics.]

S-PLUS 4.0 for Windows introduced a completely separate style of graphics based on menus and toolbars with simple command-line equivalents. The style of the interface is designed for intuitive exploration by experienced Windows users; most of the options are set from dialog boxes brought up by selecting and double-clicking or right-clicking elements of the plot. There are separate dialog

boxes for different plot elements: at least the background, plotted objects, axes and any annotations.

This graphical system works in a new graphical device called a *Graph Sheet*. A graphsheet device can be launched from the command window by the function `graphsheet()`, but it will normally be opened from a plot palette button or from the new document icon on the toolbar. Graphsheets can be used as graphics devices with both base and Trellis command-line graphics, and provide limited editing facilities, for example to edit the text of labels and the colour and width of lines, if the ‘Object-oriented Graphs’ button  has been selected or if the graphsheet was started with argument `object.mode=T`. It is possible to convert graphs to the editable form at a later date using the object browser (see page 5). Note that command-line graphics will not normally use[1] a graphsheet already opened from the GUI, but will open a new one. Normally the GUI graphics will start a new graphsheet for each plot, whereas the command-line graphics will re-use the existing one.

To produce a hardcopy of a graphsheet to a printer the normal Windows printing facilities can be used: it is also possible to export the graph(s) to a file via the Export Graph... item on the File menu. Finally, graphsheets can be saved (as S-PLUS graph files with extension `.sgr`) and re-imported for editing or additions. Their structure can be browsed in the graphsheets view of the object browser (another part of the new interface).

2D plots

The 2D plot palette (Figure 3.23) on the toolbar contains many buttons for different plot types plus six which control the placement of axes and three the production of conditioning plots. The basic types of plots are scatter and line plots (of `types` `"l"`, `"s"` and `"S"`), smoothed plots, and various fitted lines. There are also barplots, histograms, and pie charts.

Most of these plots work on two columns of data[2]. As they are not specified as arguments to a function, they need to be specified in some other way, and there are many possibilities. The most convenient way will often be to double-click a data frame in the object browser, select the desired columns (in the order x, y, z if needed, . . .) and either drag-and-drop onto the appropriate button in the 2D plot palette, or just click on the button. The variables may be specified or changed using the either of the dialog boxes brought up by double-clicking the plotted points or the background.

All plots can be conditioned, in the Trellis style but with a little less flexibility. The default is to use four panels in a 2×2 layout. The conditioning variable(s) can be set in the Multipanel tab of the plot background dialog box or by drag-and-drop. For the latter, select the desired variable(s) in an object browser of an open view of the data frame, then click and hold over the columns (not their headers) and drag to the graphsheet window. A rectangular drop target will appear

[1] An existing graphsheet can be taken over by using its name as the `Name` argument to `graphsheet`.
[2] If only one column is selected this is used for y and the index is used for x.

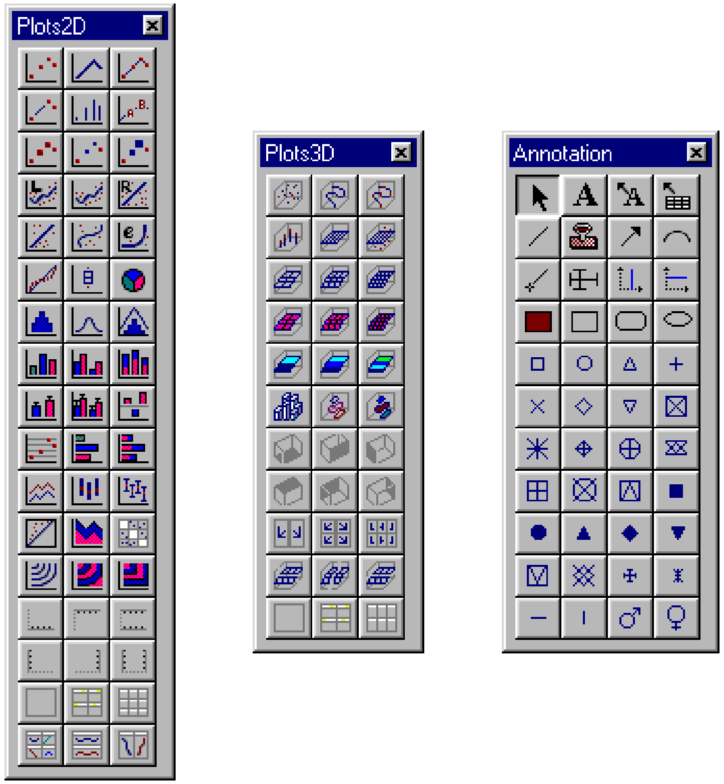


Figure 3.23: Three palettes from S-PLUS 4.0.

in the graph window above the plot area: 'drop' the columns there. (Dropping the variables in the plot area will replace the plotted variables.) Alternatively, additional columns can be selected and the Conditioning button selected on the toolbar before generating the plot.

Buttons on the 2D plot palette can select no conditioning or a 3×3 layout as well as the default 2×2 layout. Considerably greater control over the conditioning is available from the Multipanel tab. This allows the number and layout of panels to be changed, the partitioning of continuous variables to be set (with overlap as in shingles if desired), and strips to be plotted or not. (If they are, they can be selected and their properties edited.)

3D plots

The distinction between 2D and 3D plots is by appearance not data; contour, level and filled contour plots are on the 2D plot palette even though they require *three* columns to be selected. Like Trellis plots, the contour and surface plots require a z coordinate evaluated at a rectangular grid of x and y coordinates, but unlike Trellis they will interpolate irregular data to an automatically chosen grid (controlled from the Gridding tab of the plot dialog box).

The contour and levels plots are analogues of `contourplot` and `levelplot`; the filled contour plots are contour plots with the regions between contours filled in different colours. The details of the contours, levels and colours can be altered from the dialog box brought up by double-clicking or right-clicking inside the plot region.

The analogue of the Trellis function `cloud` is a 3D scatterplot from the 3D plot palette. A cloud of points can be represented as points, connected by lines (with or without highlighting the points) or as a 'dropped line scatter' plot where each point is represented by a line segment from (x, y, z) to $(0, 0, z)$. There are also 3D bar charts (sometimes known as Manhattan diagrams) for data on a rectangular grid.

Surfaces can be represented in many ways:

(a) As a wireframe surface, plotted at the grid spacing or at every other grid point.

(b) As a wireframe interpolated to a finer grid (default half the spacing) by splines.

(c) Filled versions of (a) and (b), in which the surface is shown in a solid colour (selected from the Fills tab of the dialog box selected from the surface).

(d) A draped surface, with levels represented by 8, 16 or 32 colours. (Many representations intermediate between (c) and (d) can be selected from the Lines and Fills tabs.)

(e) Contours or filled contours, in which contour levels are plotted at the appropriate height as horizontal sections of the surface.

To change the representation of a surface, select the surface by clicking on it, then the appropriate button in the 3D plot palette (Figure 3.23).

All of these 3D plots can be rotated. First select the plot area by clicking within the plot region delimited by the axes, but not on the surface. Four circles and a triangle will appear. Any of these can be dragged to rotate the plot: the circles give horizontal rotation (about the vertical axis) and the triangle rotates the vertical axis. It may be a good idea to change to a simple view of the surface (such as a coarse wireframe grid) if rotation proves to be slow.

The software also allows multiple (2, 4 or 6) views of the plot from different (equispaced) angles, and these can be rotated simultaneously by rotating one of the panels. Multiple views are selected by buttons on the 3D plot palette. As these are a form of conditioning, a single view is selected again by clicking on the 'no conditioning' button in the palette.

Plots can also be conditioned on the x, y or z variables and shown as a series of 'exploded' views.

All the plots from the 3D palette can be conditioned on additional variables, selecting 2×2 or 2×3 layouts from a palette button, with fine-tuning from the background dialog box.

The shape of the enclosing cuboid (the `aspect` parameter in 3D Trellis) can be set from the 3D Workbox tab of the plot's dialog box.

Editing plots

Many of the properties of a graph can be altered from dialog boxes. To select a part of the graph (such as an axis or fitted line or label) (left-)click on it. Clicking on any of the data points in a 2D plot will select both the points and the fitted curve. Then either double-clicking or right-clicking will bring up or a tabbed dialog box or a shortcut menu to the tabs from which the properties of that part can be selected.

The plot region or the whole graph can be selected. The dialog box has at least four tabs, Plot Summary (including the data frame used), Position/Size (which includes aspect ratio, with a welcome option for 'proportional units'), Fill/Border (colours, patterns, . . .) and Multipanel (for conditioning). The 3D plots add the 3D Workbox tab. When the whole graph is selected it can be resized by dragging the handles, or moved by dragging a point outside the plot region, and similarly for the plot region.

Once an axis label or title is selected, clicking on the text brings up an 'in-place' edit box for replacement text. Double-clicking on the surrounding box enables properties such as font and colour to be altered: these can also be altered from the toolbar when the text is selected.

Legends and titles can be added from the Insert menu; showing a legend can also be toggled from a toolbar button.

There is an *annotation* palette (Figure 3.23) which has tools to label points (as in `identify`) and to add text, a date stamp or various symbols to a graph. This palette is selected from its toolbar button or from the Toolbars item on the Views menu. Its effect is to provide a simple drawing package with which to enhance graphs.

Multiple graphs

A graph sheet can display more than one graph. To add a graph to an existing graph sheet display, ensure that no elements are selected and create the new graph, holding down Shift whilst the plot button is clicked. (As we saw in the Introduction, doing this whilst a graph is selected adds to that graph.) It may be necessary to use the Arrange Graphs item on the Format menu to produce a usable layout (as the default might be to overlay the graphs). The graphs can be re-ordered by selecting them in the order required (use shift-click) and then using Arrange Graphs.

Graphsheets can make multiple pages of graphs. The circumstances under which they do so is set by the Auto pages item in the Options|Graph Options

... dialog box, which can be overridden for each graphsheet from the Options tab of its right-click background menu. The default is to create separate pages if sent several frames (plots starting on a new page) during the execution of a single expression, for example when using `plot(lm.object, ask=T)`. This can be changed to a new graph page for every plot, or to always use the same graph page. To set this up from the command line we can use, for example

```
graphsheet(Name="GStest")
guiModify("GraphSheet", Name="GStest",
          AutoPageMode="Every Graph")
```

In `4.5` we can use `guiGetGSName()` for the name of the current graphsheet.

Setting GUI properties from the command-line

As the last example shows, it is possible to change almost all of the settings in the GUI by calls from the S language. The simplest (and in many cases the only) way to find the corresponding S command is to invoke the operation from the GUI and then open a history window (using toolbar button to the left of that for the command-line window labelled by a scroll). The appropriate command(s) will be recorded in the history window and can be used from an S script with minimal changes (for example giving the appropriate name for the graphsheet).

To find the existing settings of a graphsheet (say), use

```
nm <- guiGetArgumentNames("GraphSheet")
pr <- guiGetPropertyValue("GraphSheet", Name="GStest")
names(nm) <- pr
print(pr)
```

and this listing gives the corresponding arguments to be used with `guiModify` to alter the settings. However, it does not seem to be possible to find the set of allowable values for the settings except by trying them from a dialog box or menu and examining the history.

Using command-line graphics on a graphsheet

The traditional command-line graphics operations are mapped to objects in the object-oriented graphics model. In the default 'fast' mode the graphics calls in each top-level S expression are mapped to a single composite object. This is usually what is required, but can have some unexpected side-effects. One is that all the graphics calls from within a function are combined into a single object, and *nothing is displayed* until the function call finishes. Similarly, running a series of simulations in a `for` loop and plotting the results after each one will under 4.x not display any of the results until the whole series of simulations has completed. This is in contrast to the behaviour under S-PLUS 3.3 for Windows and for the Unix versions, under which the graphics are displayed immediately.

The work-around is to ensure that the graphics calls are split into multiple objects; this can only be done in 4.0 release 2 or later using an undocumented effect of the function `guiLocator`. If this is called with a negative argument it pauses (`guiLocator(-n)` pauses for `n` seconds), and then plots the pending graphics calls (which will appear in an object-browser view of the graphsheet as a single object). Two side-effects of the call to `guiLocator` are to move the focus to the graphsheet which is thus brought to the front and to start a new expression for the purposes of the Page creation option (the `AutoPageMode` property) of the graphsheet.

Chapter 6

Linear Statistical Models

6.7 Multiple comparisons

As we all know, the theory of p-values of hypothesis tests and of the coverage of confidence intervals applies to pre-planned analyses. However, the only circumstances in which an adjustment is routinely made for testing after looking at the data is in multiple comparisons of contrasts in designed experiments. Consider the experiment on yields of barley in our dataset `immer`[1]. This has the yields of five varieties of barley at six experimental farms in both 1931 and 1932; we will average the results for the two years. An analysis of variance gives

```
> immer.aov <- aov((Y1+Y2)/2 ~ Var + Loc, data=immer)
> summary(immer.aov)
          Df Sum of Sq Mean Sq F Value     Pr(F)
      Var  4      2655   663.7   5.989 0.0024526
      Loc  5     10610  2122.1  19.148 0.0000005
Residuals 20      2217   110.8
```

The interest is in the difference in yield between varieties, and there is a statistically significant difference. We can see the mean yields by a call to `model.tables`.

```
> model.tables(immer.aov, type="means", se=T, cterms="Var")
    ....
 Var
      M      P      S     T      V
 94.392 102.54 91.133 118.2 99.183

Standard errors for differences of means
          Var
        6.078
replic. 6.000
```

This suggests that variety `T` is different from all the others, as a pairwise significant difference at 5% would exceed $6.078 \times t_{20}(0.975) \approx 12.6$; however the comparisons to be made have been selected after looking at the fit.

[1] the Trellis dataset `barley` discussed in Cleveland (1993) is a more extensive version of the same dataset.

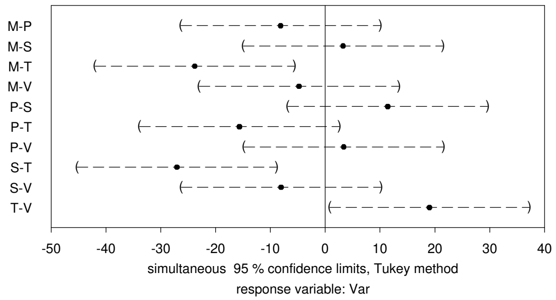


Figure 6.1: Simultaneous 95% confidence intervals for variety comparisons in the `immer` dataset.

Function `multicomp` allows us to compute *simultaneous* confidence intervals in this problem, that is confidence intervals such that the probability that they cover the true values for all of the comparisons considered is bounded above, by 5% for 95% confidence intervals. We can also plot the confidence intervals (Figure 6.1), by

```
> multicomp(immer.aov, plot=T)
95 % simultaneous confidence intervals for specified
linear combinations, by the Tukey method

critical point: 2.9925
response variable: Var

intervals excluding 0 are flagged by '****'

    Estimate Std.Error Lower Bound Upper Bound
M-P    -8.15      6.08     -26.300       10.00
M-S     3.26      6.08     -14.900       21.40
M-T   -23.80      6.08     -42.000       -5.62 ****
M-V    -4.79      6.08     -23.000       13.40
P-S    11.40      6.08      -6.780       29.60
P-T   -15.70      6.08     -33.800        2.53
P-V     3.36      6.08     -14.800       21.50
S-T   -27.10      6.08     -45.300       -8.88 ****
S-V    -8.05      6.08     -26.200       10.10
T-V    19.00      6.08       0.828       37.20 ****
```

This does not allow us to conclude that variety `T` has a significantly different yield from variety `P`.

We may want to restrict the set of comparisons, for example to comparisons with a control treatment. The dataset `oats` is discussed on page 300; here we ignore the split-plot structure.

```
> oats1 <- aov(Y ~ N + V + B, data=oats)
```



```
> summary(oats1)
          Df Sum of Sq Mean Sq F Value    Pr(F)
        N  3     20020  6673.5  28.460 0.000000
        V  2      1786   893.2   3.809 0.027617
        B  5     15875  3175.1  13.540 0.000000
Residuals 61     14304   234.5
> multicomp(oats1, focus="V")

95 % simultaneous confidence intervals for specified
linear combinations, by the Tukey method

critical point: 2.4022
response variable: N

intervals excluding 0 are flagged by '****'

                       Estimate Std.Error Lower Bound
Golden.rain-Marvellous    -5.29      4.42      -15.90
   Golden.rain-Victory     6.88      4.42       -3.74
    Marvellous-Victory    12.20      4.42        1.55
                       Upper Bound
Golden.rain-Marvellous        5.33
   Golden.rain-Victory       17.50
    Marvellous-Victory       22.80 ****

> multicomp(oats1, focus="N", comparison="mcc", control=1)
    ....
              Estimate Std.Error Lower Bound Upper Bound
0.2cwt-0.0cwt     19.5       5.1        7.24        31.8 ****
0.4cwt-0.0cwt     34.8       5.1       22.60        47.1 ****
0.6cwt-0.0cwt     44.0       5.1       31.70        56.3 ****
```

Note that we need to specify the control level: perversely by default the last level is chosen. We might also want to know if all the increases in nitrogen give significant increases in yield, which we can examine by

```
> lmat <- matrix(c(0,-1,1,rep(0, 11), 0,0,-1,1, rep(0,10),
                 0,0,0,-1,1,rep(0,9)),,3, dimnames=list(NULL,
         c("0.2cwt-0.0cwt", "0.4cwt-0.2cwt", "0.6cwt-0.4cwt")))
> multicomp(oats1, lmat=lmat, bounds="lower", comparisons="none")
    ....
              Estimate Std.Error Lower Bound
0.2cwt-0.0cwt    19.50       5.1        8.43 ****
0.4cwt-0.2cwt    15.30       5.1        4.27 ****
0.6cwt-0.4cwt     9.17       5.1       -1.90
```

There are a bewildering variety of methods for multiple comparisons reflected in the options for `multicomp`. Miller (1981), Hsu (1996) and Yandell (1997, Chapter 6) give fuller details. Do remember that this tackles only part of the problem; the analyses here have been done after selecting a model and specific factors on which to focus: the allowance for multiple comparisons is only over contrasts of one selected factor in one selected model.

Chapter 8

Robust Statistics

8.3 Robust regression

S-PLUS 4.5 introduced a new method of robust regression, `lmRobMM` due to Yohai *et al.* (1991), which is also in S-PLUS 5.x. This comes with a full set of method functions, even for `add1` and `drop1`, so can be used routinely as a replacement for `lm`. The method used is an M-estimate with a re-descending ψ function and a starting value for the optimization that is chosen as a highly resistant 'S' estimator. The optimization algorithm uses a random search, so the results will not be exactly repeatable.

Let us try it on the `phones` data.

```
> phones.lmr <- lmRobMM(calls ~ year, data=phones)
> summary(phones.lmr)
Final M-estimates.

Call: lmRobMM(formula = calls ~ year, data = phones)

Residuals:
    Min    1Q Median    3Q   Max
 -1.719 -0.46 0.2267 39.03 188.5

Coefficients:
               Value Std. Error  t value Pr(>|t|)
(Intercept) -52.3103   3.7851   -13.8199   0.0000
       year   1.0990   0.0636    17.2853   0.0000

Residual scale estimate: 2.027 on 22 degrees of freedom
Proportion of variation in response explained by model: 0.4898

Test for Bias
            Statistics P-value
 M-estimate      1.601   0.449
LS-estimate      0.243   0.886
> plot(phones.lmr)
```

This works well, rejecting all the spurious observations. The 'test for bias' is of the M-estimator against the initial S-estimator; if the M-estimator appears biased the initial S-estimator is returned.

The `compare.fits` function makes it easy to compare this fit with that from `lm` or similar functions.

```
> phones.lm <- lm(calls ~ year, data=phones)
> compare.fits(phones.lmr, phones.lm)
    ....
Coefficients:
            phones.lmr phones.lm
(Intercept)    -52.310  -260.059
       year      1.099     5.041

Residual Scale Estimates:
phones.lmr : 2.027 on 22 degrees of freedom
 phones.lm : 56.22 on 22 degrees of freedom
```

This also has `summary` and `plot` methods.

For Brownlee's stack loss data we get similar results to `rlm` (page 263) but with a smaller estimated scale.

```
> stack <- data.frame(stack.x, loss=stack.loss)
> stack.lmr <- lmRobMM(loss ~ ., data=stack)
> summary(stack.lmr, cor=F)
Final M-estimates.

Call: lmRobMM(formula = loss ~ ., data = stack)

Residuals:
   Min      1Q Median    3Q   Max
 -8.63 -0.6713 0.3594 1.151 8.174

Coefficients:
              Value Std. Error  t value Pr(>|t|)
(Intercept) -37.6525   5.0026    -7.5266   0.0000
   Air.Flow   0.7977   0.0713    11.1886   0.0000
 Water.Temp   0.5773   0.1755     3.2905   0.0043
 Acid.Conc.  -0.0671   0.0651    -1.0297   0.3176

Residual scale estimate: 1.837 on 17 degrees of freedom
```

and for the `hills` data we have

```
> summary(lmRobMM(time ~ dist + climb, data=hills,
                  weights=1/dist^2))
    ....
Coefficients:
             Value Std. Error t value Pr(>|t|)
(Intercept) -3.3745  4.8433    -0.6967  0.4910
       dist  5.5365  1.2347     4.4840  0.0001
      climb  0.0086  0.0037     2.3282  0.0264

Residual scale estimate: 0.7921 on 32 degrees of freedom

> summary(lmRobMM(ispeed ~ grad, data=hills))
Coefficients:
              Value Std. Error t value Pr(>|t|)
(Intercept)  5.0754  0.4210    12.0563  0.0000
       grad  0.0077  0.0016     4.8817  0.0000

Residual scale estimate: 0.8189 on 33 degrees of freedom
```


8.4 Resistant regression

S-PLUS 4.x and S-PLUS 5.x have an alternative formula-based interface for `lmsreg` and `ltsreg`, and `print`, `summary` and `plot` methods. We can try these on the stack loss example.

```
> stack <- data.frame(stack.x, loss=stack.loss)
> lmsreg(loss ~ ., data=stack)
$coefficients:
 Intercept Air.Flow Water.Temp Acid.Conc.
    -39.25     0.75        0.5          0
$scale:
        Y
 1.207615
$residuals:
    1    2   3    4     5     6     7    8     9   10   11   12
 7.75 2.75 7.5 8.75 -0.25 -0.75 -0.25 0.75 -0.75 0.75 0.75 0.25
    13    14   15    16   17   18   19   20    21
 -2.25 -1.75 0.75 -0.25 0.25 0.25 0.75 2.25 -8.25
    ....
> ltsreg(loss ~ ., data=stack)
    ....
Coefficients:
 Intercept Air.Flow Water.Temp Acid.Conc.
 -36.2921    0.7362   0.3691     0.0081

Scale estimate of residuals:  1.038

Total number of observations:  21

Number of observations that determine the LTS estimate:  13
> plot(ltsreg(loss ~ ., data=stack))
```

Remember that the results are random, but this version of the code does seem to produce different answers.

The plot method gives a normal QQ-plot of the residuals and plots of the standarized residuals against fitted values, index and robust distance from the centre of the x values.

For the `hills` data we can use

```
> summary(ltsreg(time ~ ., data=hills))
    ....
Coefficients:
 Intercept    dist   climb
 -0.9477    4.7817  0.0086

Scale estimate of residuals: 3.033

Robust Multiple R-Squared: 0.9761

Total number of observations:  35

Number of observations that determine the LTS estimate:  19

Residuals:
  Min. 1st Qu. Median 3rd Qu.  Max.
 -11.8 -0.7073 0.6713   5.812 64.21

Weights:
  0  1
 10 25

> par(pty="s", mfrow=c(1,2))
> plot(ltsreg(time ~ ., data=hills), which=4:3)

> hills$ispeed <- hills$time/hills$dist
> hills$grad <- hills$climb/hills$dist
> ltsreg(ispeed ~ grad, data=hills)
Coefficients:
 Intercept   grad
 4.5707    0.0090
```

Two of the diagnostic plots are shown in Figure 8.1.

8.5 Multivariate location and scale

Function `cov.mcd` provides an alternative robust estimator of multivariate location and covariance or correlations developed by Rousseeuw (1984); see also Rousseeuw & Leroy (1987, p.262). MCD stands for 'minimum covariance determinant' and corresponds to taking the covariance of (about) half the points with the smallest determinant rather than with the smallest enclosing volume as in `cov.mve`. In a sense discussed in that paper, `cov.mve` generalizes LMS and `cov.mcd` generalizes LTS.

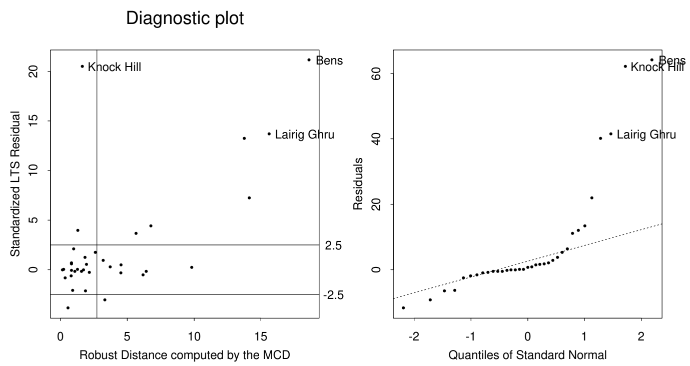


Figure 8.1: Diagnostic plots for an `ltsreg` fit to the `hills` dataset.

Chapter 12

Survival Analysis

S-PLUS 4.5 and S-PLUS 5.0 introduced further functions for survival analysis from a different author: these have a very substantial overlap with those already in S-PLUS but are more general in that they allow *truncation* as well as *censoring*. Either or both censoring and truncation occur when subjects only observed for part of the time axis. An observation T_i is right-censored if it is known only that $T_i > U_i$ for a censoring time U_i, and left-censored if it is only known that that $T_i \leqslant L_i$. (Both left- and right-censoring can occur in a study, but not for the same individual.) Interval censoring is usually taken to refer to subjects known to have an event in $(L_i, U_i]$, but with the time of the event otherwise unknown. Truncation is similar but subtly different. For left and right truncation, subjects with events before L_i or after U_i are not included in the study, and interval truncation refers to both left and right truncation. (Notice the inconsistency with interval censoring.)

We can consider the corresponding contributions to the likelihood. Suppose we have interval truncation on $[0 \leqslant L_i, R_i \leqslant \infty)$. Then the contributions are

(a) $f(t_i)/[F(U_i) - F(L_i)]$ for an observed event at t_i.

(b) $[F(C_i) - F(L_i)]/[F(U_i) - F(L_i)]$ for an event left-censored at C_i), that is known to occur in the observation interval prior to C_i.

(c) $[F(U_i) - F(C_i)]/[F(U_i) - F(L_i)]$ for an event right-censored at C_i but know to occur before the end of the observation interval.

(d) $[F(D_i) - F(C_i)]/[F(U_i) - F(L_i)]$ of an event interval-censored in $(C_i, D_i]$.

Clearly right-censoring with right-truncation or left-censoring with left-truncation will rarely make sense.

12.1 Estimators of survival curves

The new functions use functions `censor` which is almost equivalent to `Surv` but whose output is tailored for the new functions (and is incompatible with that from `Surv`, so the correct function must be used). There is a function `kaplanMeier` which computes Kaplan-Meier estimates of the survival curve in a very similar way to `survfit`. For example, compare


```
> kaplanMeier(censor(time, cens) ~ treat, data=gehan,
  conf.interval="log-log")
treat=6-MP
Number Observed: 21
Number Censored: 12
Confidence Type: log-log
            Survival Std.Err 95% LCL 95% UCL
(-Inf,  6]     1.000   0.000   1.000   1.000
(   6,  7]     0.857   0.076   0.666   0.943
(   7, 10]     0.807   0.087   0.622   0.907
(  10, 13]     0.753   0.096   0.576   0.864
(  13, 16]     0.690   0.107   0.521   0.810
(  16, 22]     0.627   0.114   0.471   0.749
(  22, 23]     0.538   0.128   0.395   0.661
(  23, 35]     0.448   0.135   0.328   0.561

treat=control
Number Observed: 21
Number Censored: 0
Confidence Type: log-log
             Survival Std.Err 95% LCL 95% UCL
(-Inf,   1]     1.000   0.000   1.000   1.000
(   1,   2]     0.905   0.064   0.704   0.972
(   2,   3]     0.810   0.086   0.626   0.909
(   3,   4]     0.762   0.093   0.588   0.870
(   4,   5]     0.667   0.103   0.513   0.781
(   5,   8]     0.571   0.108   0.442   0.682
(   8,  11]     0.381   0.106   0.302   0.459
(  11,  12]     0.286   0.099   0.232   0.342
(  12,  15]     0.190   0.086   0.160   0.223
(  15,  17]     0.143   0.076   0.122   0.165
(  17,  22]     0.095   0.064   0.084   0.108
(  22,  23]     0.048   0.046   0.043   0.052
(  23, Inf)     0.000   0.000      NA      NA
```

There are appears to be no way to plot the results; the GUI item for Kaplan-Meier... is an interface to `survfit`. The functions `qkaplanMeier` estimates quantiles by linear interpolation on the results of a call to `kaplanMeier`.

The advantage of `kaplanMeier` comes with interval-censored data, which it can handle and `survfit` cannot. As a simple example, suppose that the `leuk` data had only be recorded in 4-week periods.

```
mn <- 4 * (leuk$time %/% 4)
kaplanMeier(censor(mn, mn + 4, rep(3, length(mn))) ~ 1)
Number Observed: 33
Number Censored: 33
Confidence Type: log
             Survival Std.Err 95% LCL 95% UCL
(-Inf,   0]     1.000   0.000   1.000   1.000
(   4,   4]     0.818   0.067   0.717   0.933
```

```
(   8,   8]    0.636   0.084   0.540   0.750
(  12,  16]    0.606   0.085   0.513   0.716
(  20,  20]    0.515   0.087   0.434   0.611
(  24,  24]    0.455   0.087   0.384   0.539
(  28,  28]    0.424   0.086   0.358   0.502
(  32,  36]    0.394   0.085   0.333   0.465
(  40,  40]    0.364   0.084   0.309   0.428
(  44,  56]    0.333   0.082   0.284   0.391
(  60,  64]    0.273   0.078   0.234   0.317
(  68, 100]    0.182   0.067   0.159   0.207
( 104, 108]    0.152   0.062   0.134   0.171
( 112, 120]    0.121   0.057   0.108   0.135
( 124, 132]    0.091   0.050   0.082   0.100
( 136, 140]    0.061   0.042   0.056   0.066
( 144, 156]    0.030   0.030   0.029   0.032
( 160, Inf)    0.000   0.000      NA      NA
```

which is not altogether helpful since many of those intervals are empty.

12.2 Parametric models

The new analogue to `survreg` is `censorReg`, which has a GUI interface to specify its many parameters. This has a longer list of distributions (`"extreme"`, `"weibull"`, `"gaussian"`, `"lognormal"`, `"logistic"`, `"loglogistic"`, `"exponential"`, `"logexponential"`, `"rayleigh"` and `"lograyleigh"`). Remember (page 352) that with `survreg` there is a confusion as to whether the names refer to the distribution of T or of $\log T$. That confusion is even worse here, with what is to `survreg` the `"exponential"` and `"rayleigh"` distributions becoming `"logexponential"` and `"lograyleigh"` (and the details are not documented at all)! Examining the code (in the misleadingly-titled function `make.distribution`) shows that `distribution =`

1. `"weibull"`, `"lognormal"` and `"loglogistic"` give accelerated life models for those distributions.
2. `"logexponential"` and `"lograyleigh"` give accelerated-life models for the exponential and Rayleigh distributions respectively.
3. `"extreme"`, `"gaussian"`, `"logistic"`, `"exponential"` and `"rayleigh"` give additive models for those distributions, that is

$$T \sim \boldsymbol{\beta}^T \boldsymbol{x} + \sigma\,\epsilon$$

   known as the identity 'link' in `survreg`.

Let us consider a simple example using `gehan`. We can fit a Weibull model by


```
> options(contrasts=c("contr.treatment", "contr.poly"))
> summary(censorReg(censor(time, cens) ~ treat, gehan))
Call:
censorReg(formula = censor(time, cens) ~ treat, data = gehan)

Distribution:  Weibull

Standardized Residuals:
              Min   Max
Uncensored 0.046 3.359
  Censored 0.095 1.056

Coefficients:
             Est. Std.Err. 95% LCL 95% UCL z-value   p-value
(Intercept)  3.52    0.252    3.02   4.009   13.96 2.61e-044
      treat -1.27    0.311   -1.88  -0.658   -4.08 4.51e-005

Extreme value distribution: Dispersion (scale) = 0.73219
Observations: 42 Total; 12 Censored
-2*Log-Likelihood: 213
```

If we compare this with the result on page 355,

```
> summary(survreg(Surv(time, cens) ~ treat, gehan))
Call:
survreg(formula = Surv(time, cens) ~ treat, data = gehan)
Deviance Residuals:
   Min    1Q Median    3Q  Max
 -2.06 -1.05 -0.222 0.841 1.51

Coefficients:
            Value Std. Error z value         p
(Intercept)  3.52      0.252   13.96 2.61e-044
      treat -1.27      0.311   -4.08 4.51e-005

Extreme value distribution: Dispersion (scale) = 0.73219
Degrees of Freedom: 42 Total; 39 Residual
-2*Log-Likelihood: 94.1
```

we see the agreement is good apart from the log-likelihoods. We can look at this more precisely:

```
> censorReg(censor(time,cens) ~ treat, gehan)$loglik
[1] -116.41 -106.58
> survreg(Surv(time,cens) ~ treat, gehan)$loglik
[1] -57.671 -47.064
> censorReg(censor(time,cens) ~ treat, gehan,dist="logexp")$loglik
[1] -116.77 -108.52
> survreg(Surv(time,cens) ~ treat, gehan, dist="exp")$loglik
[1] -57.251 -49.009
```

so the increase in log-likelihood over the null model is about the same. Part of the answer is that `survreg` is quoting the log-likelihood regarding $\log T$ as the data, whereas `censorReg` is (more naturally) regarding T as the data. The formula for the likelihood ((12.1) on page 344) shows that it is a product of terms for censored observations, which are probabilities, and of terms for uncensored observations, which are densities. The latter are affected by the transformation of T, so

$$L(\text{parameters}; (\log T_i)) = L(\text{parameters}; (T_i)) \times \prod_{\delta_i=1} T_i$$

We can check this for the `gehan` data

```
> attach(gehan)
> sum(log(time[cens==1]))
[1] 59.515
```

which is precisely the difference in the log-likelihood for the fitted models, and for the null model for the exponential distribution. In the Weibull case, `survreg` is quoting the null-model log-likelihood at the shape parameter it fits to the full model, which is not statistically meaningful.

The advantages of `censorReg` come from its wider range of options. As noted above, it allows truncation, by specifying a call to `censor` as the `truncation` argument. Distributions can be fitted with a *threshold*, that is a parameter $\gamma > 0$ such that the failure-time model is fitted to $T - \gamma$ (and hence no failures can occur before time γ). It the parameter `threshold = T`, γ is estimated as 90% of the smallest observed failure time; if `threshold = "Linearized-qq"` γ is chosen by optimizing the linearity of a QQ-plot of the fitted response and a Kaplan-Meier estimate of survival.

There is a `plot` method for `censorReg`, which appears to require the syntax

```
gehan.cr <- censorReg(censor(time, cens) ~ factor(treat), gehan)
plot(gehan.cr)
```

This produces up to seven figures, of residuals against fitted values, the square root of the absolute value of the residuals against fitted values, and the response against the fitted values (all of which are useless here as the fitted values are the mean for the appropriate group and so take just two values), Weibull probability plots of the residuals and by group, a so-called *stress plot* (not for a factor variable) and a figure showing probability plots by group for each of six distributions (Weibull, log-normal and log-logistic and the corresponding additive models). This function often fails for correctly specified models.

The plot methods are available separately as functions `probplot.censorReg`, `stressplot.censorReg` and `probplot6.censorReg`. The probability plots for the `gehan` example are shown in Figures 12.1 and 12.2.

To show a stress plot (Figure 12.3) we had to resort to the following manipulations and model.

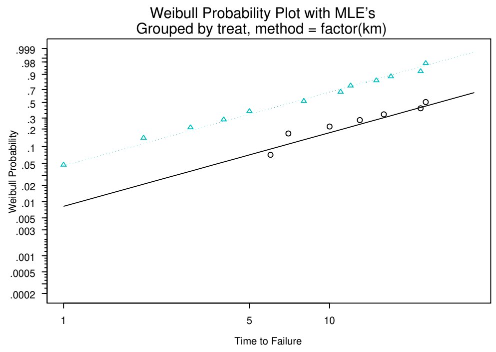


Figure 12.1: Weibull probability plot for `gehan` dataset. The two groups correspond to the two treatments: only the uncensored observations are plotted.

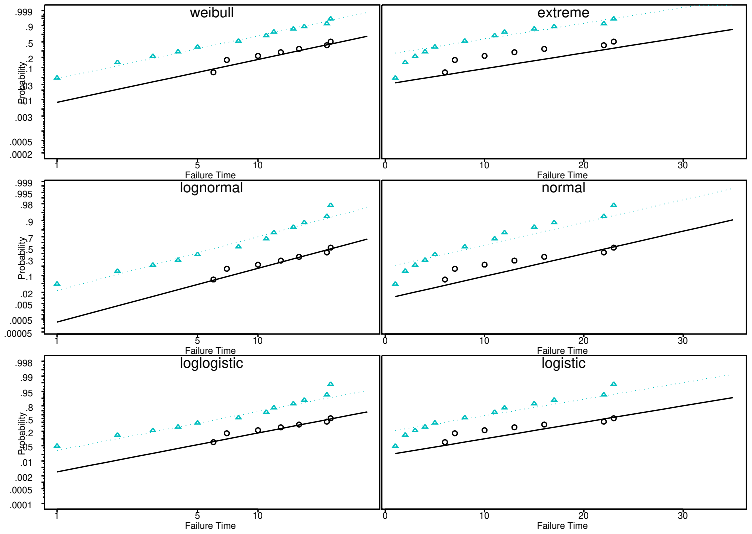


Figure 12.2: Six probability plots for the `gehan` dataset produced by `probplot6.censorReg`. Details as Figure 12.1.

```
leuk <- leuk
attach(leuk); leuk$lwbc <- log(wbc); detach()
plot(censorReg(censor(time) ~ lwbc, data=leuk))
```

This is a graphical version of the prediction method, showing quantiles against a single numerical covariate (although plotted with x and y axes reversed).

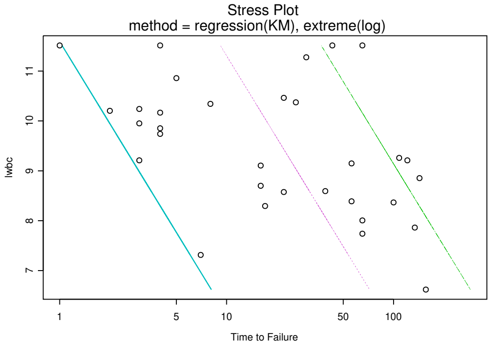


Figure 12.3: 'Stress plot' for a Weibull fit to the `leuk` dataset. The lines correspond to probabilities of 10%, 50% and 90% from left to right.

There is a `predict` method that allows prediction of the times at which the probability of an event is as given (by default 10%, 50% and 90%), or the probabilities at specified times.

```
> predict(gehan.cr)
$"factor(treat)=control":
    Estimate Std.Err  95% LCL 95% UCL
0.1   1.8233 0.36576  0.89027  3.7342
0.5   7.2427 0.18407  5.04919 10.3891
0.9  17.4445 0.17041 12.49137 24.3618

$"factor(treat)=6-MP":
    Estimate Std.Err 95% LCL 95% UCL
0.1   6.4752 0.32912  3.3971  12.343
0.5  25.7215 0.24442 15.9310  41.529
0.9  61.9521 0.29598 34.6829 110.662

> predict(gehan.cr, q=seq(10,30,10), type="probability")
$"factor(treat)=control":
   Estimate Std.Err 95% LCL 95% UCL
10  0.65934 0.36507 0.48622 0.79833
20  0.93767 0.72760 0.78327 0.98428
30  0.99200 1.46972 0.87432 0.99955

$"factor(treat)=6-MP":
   Estimate Std.Err  95% LCL 95% UCL
10  0.17366 0.42656 0.083482 0.32654
20  0.38834 0.42417 0.216589 0.59317
30  0.57482 0.50317 0.335227 0.78376
```

References

Cleveland, W. S. (1993) *Visualizing Data*. Summit, NJ: Hobart Press. [17]

Hsu, J. C. (1996) *Multiple Comparison Procedures: Theory and Methods*. London: Chapman & Hall. [19]

Miller, R. G. (1981) *Simultaneous Statistical Inference*. New York: Springer-Verlag. [19]

Rousseeuw, P. J. (1984) Least median of squares regression. *Journal of the American Statistical Association* **79**, 871–881. [24]

Rousseeuw, P. J. and Leroy, A. M. (1987) *Robust Regression and Outlier Detection*. New York: John Wiley and Sons. [24]

Yandell, B. S. (1997) *Practical Data Analysis for Designed Experiments*. London: Chapman & Hall. [19]

Yohai, V., Stahel, W. A. and Zamar, R. H. (1991) A procedure for robust estimation and inference in linear regression. In *Directions in Robust Statistics and Diagnostics, Part II*, eds W. A. Stahel and S. W. Weisberg. Springer-Verlag. [21]