Complements to

S Programming

by

W. N. Venables and B. D. Ripley

Springer (2000). ISBN 0-387-98966-8

26 August 2003

These complements have been produced to supplement *S Programming*. They will be updated from time to time. The definitive source is http://www.stats.ox.ac.uk/pub/MASS3/Sprog.

© W. N. Venables and B. D. Ripley 2000–3. A licence is granted for personal study and classroom use. Redistribution in any other form is prohibited.

Selectable links are in this colour.
Selectable URLs are in this colour.

Introduction

These complements are made available on-line to supplement the book. In particular they describe changes for later versions of R and for S-PLUS 6.x on Unix and on Windows. Detailed changes are given in the on-line errata files: in particular many of the aspects in which R was different in late 1999 have been removed.

The general convention is that material here should be thought of as following the material in the chapter in the book, so that new sections are numbered following the last section of the chapter, and figures and equations here are numbered following on from those in the book.

Contents

Chapter 3

The S Language: Advanced Aspects

3.3 Calling the operating system

S-PLUS 6.x under Windows

The `system` and `dos` commands are essentially the same as previous versions of S-PLUS under Windows.

Function `proc.time` now return a 5-element vector as on Unix versions, but the last two components[1] are always zero, and the CPU-time components are `NA` on Windows 95/98/ME machines.

There is a function `sys.time` which returns the elapsed time, and on Windows NT/2000/XP also the CPU time. This is inadequately documented, but inspection of the code shows that the return value is a two-element vector, the first element being the sum of the user and system CPU times (`NA` on Windows 95/98/ME machines) and the second showing the elapsed time.

`sys.time` is also available on Unix versions of S-PLUS 6.x.

[1] which on Unix refer to child processes

Chapter 4

Classes

4.1 Introduction to classes

The trap mentioned on page 77 for the S engines:

> There is an apparently undocumented trap in writing method functions.[1] The principal argument of the method function must have the same name as the principal argument of the generic function; thus the first argument of all `print` methods should be `x`, and of all `summary`, `predict`, `coef`, `...` methods should be `object`.

has more serious implications for S-PLUS 6 for Windows and 6.1 for Unix. The arguments not agreeing will generate errors.

As from R 1.7.0, `UseMethod` uses the class as reported by `class()`, not just the class attribute. This means it dispatches to classes such as `"matrix"` and `"numeric"`, and the reported class is never `NULL`. The example on page 79 can be amended to

```
Ttest <- function(z, ...) UseMethod("Ttest")
```

for use in R. This is different from the behaviour in the new S engine, which still needs

```
Ttest <- function(z, ...) {
  if(is.null(oldClass(z))) oldClass(z) <- data.class(z)
  UseMethod("Ttest")
}
```

as `UseMethod` dispatches only on the `class` attribute.

[1] at least in the S engines.

Chapter 5

New-style Classes

An implementation of the new-style (or 'formal' or 'S4') classes has been available as package `methods` for R since version 1.4.0, and as from version 1.7.0 this is loaded by default. The implementation is similar but differs in detail: notably the confusing class `"named"` is not used.

5.1 Creating a class

All the code in this section works except that using the unimplemented functions `dumpClass` and `hasSlot`.

5.3 Generic and method functions

Both old-style and formal classes coexist, and it is best to avoid confusion, so naming a function `print.fungi` would be a very bad idea, and would cause old-style dispatch from `print` rather than using `show` at all.

5.5 An extended statistical example revisited

This example works almost unchanged in R. There is no class called `"named"`, so we use `"numeric"` instead:

```
setClass("lda", representation(prior =   "numeric",
        counts =  "numeric",   means =   "matrix",
        scaling = "matrix",    lev =     "character",
        svd =     "numeric",   N =       "integer",
        call =    "call") )
```

and the third from last line of `lda.formula` needs to be

```
  Call$x <- as.call(Terms)
```

5.6 Group methods and another polynomial class

Group `"Logic"` (Table 5.1) does not exist.

R does not use `xyCall` (page 119).

Functions in R are manipulated rather differently (page 120), and `"{"` is not a separate mode which can be coerced to. The following version works in R:

```
setIs("polynomial", "function", coerce = function (from)
{
  p <- as.name("p")
  x <- as.name("x")
  from <- as(from, "numeric")
  if ((an <- length(from)) == 1) {
    from <- c(from, 0)
    an <- 2
  }
  statement <- call("{")
  statement[[i <- 2]] <-
    call("<-", p, call("+", from[an-1], call("*", x, from[an]) ) )
  for (ai in rev(from)[-(1:2)])
    statement[[i <- i + 1]] <-
      call("<-", p, call("+", ai, call("*", x, p) ) )
  statement[[i + 1]] <- p
  res <- function(x) {}
  body(res) <- statement
  res
})
```

but automatic coercion does not take place, as in the example at the top of page 121.

Chapter 6

Using Compiled Code

6.2 Writing compiled code to work with S

Under S-PLUS 6.x the header file `S.h` should be included. On Windows it is essential that all the necessary header files are included, as `stdcall` linkage is used and that depends on accurate header files. When writing your own header files, do include full prototypes: using `int foo();` will not work.[1]

Using C input/output

[pp. 132–3, see also p. 148.]

The header file `newredef.h` is automatically included by `S.h` on Windows versions of S-PLUS 6.x. If you need to suppress this, define `NO_NEWIO` before including `S.h`.

[1] unless `foo` has no arguments, when `int foo(void);` is better style.

Chapter 7

General Strategies and Extended Examples

7.8 John Conway's Game of Life

The Game of Life was invented by the mathematician John Horton Conway and reported in Martin Gardiner's *Mathematical Games* column of the October, 1970 issue of *Scientific American*. See, for example,

http://www.sciam.com/1999/0499issue/0499profile.html.

It is an autonomous pattern generator with the results displayed on an extended checkerboard. From any initial pattern of occupied cells on the board the next one is generated according to the following rules:

Death Each cell is considered to have eight neighbours. Any occupied cell with 0, 1, 4, 5, 6, 7 or 8 occupied neighbours is unoccupied in the next generation.

Survival If an occupied cell has 2 or 3 neighbours it is occupied again in the next generation.

Birth If an unoccupied cell has precisely 3 occupied neighbours it is occupied in the next generation.

We will use this well-known game to illustrate some useful programming techniques.

A natural way to represent the occupied cells would be as a list with components named `x` and `y`. This is one way to make them them easy to plot, for example, and enacting the rules is not very difficult. An even simpler way with some additional advantages is, paradoxically, to use complex numbers with integer real and imaginary parts. This allows the two numbers defining a cell to be handled as a single atomic quantity in arithmetic and utilities such as `match` and `unique` to be used with them directly without needing to combine the coordinates in some other way such as by pasting.

```
gen <- function(adults)
{
    adults <- as.complex(adults)
    nhbrs <- c(adults + 1, adults + 1 + (1i),
               adults + (1i), adults - 1 + (1i),
               adults - 1, adults - 1 - (1i),
               adults - (1i), adults + 1 - (1i))
    taken <- !is.na(match(nhbrs, adults))
    count <- matrix(taken, length(adults), 8) %*% rep(1, 8)
    alive <- !is.na(match(count, c(2, 3)))

    young <- unique(nhbrs[!taken])
    nhbrs <- c(young + 1, young + 1 + (1i),
               young + (1i), young - 1 + (1i),
               young - 1, young - 1 - (1i),
               young - (1i), young + 1 - (1i))
    taken <- !is.na(match(nhbrs, adults))
    born <- (matrix(taken, length(young), 8) %*% rep(1, 8)) == 3
    c(adults[alive], young[born])
}
```

Figure 7.1: Constructing a new generation in the Game of Life

Consider first a function to enact the rules above and produce the next generation from the current one. The only argument will be a vector of complex numbers representing the presently occupied cells. Such a function is shown in Figure 7.1.

The strategy is as follows:

- If the present population has n members, construct a vector of length $8n$ representing all the neighbours (possibly with repetitions). Using complex numbers this can be done in one step.
- Using `match` find which of these neighbouring cells are themselves occupied.
- For each member of the present population calculate the number of occupied neighbouring cells it has. This is a single matrix multiplication.
- Again using `match` select for survival into the next generation only those which have 2 or 3 occupied neighbours only.
- The potential births are the unoccupied neighbouring cells. Select these from the vector of neighbours and discard any duplicates.
- Using exactly the same technique as before, locate the potential birth cells which have exactly 3 occupied neighbours. These are the new births.
- The value of the function is a vector consisting of the surviving adults and the new births.

Note that we implicitly advocate here what Chambers calls a 'whole object' view of the problem: the alternative is to work at the individual cell level using slow explicit loops.

One of the many mildly amusing patterns to be discovered soon after the game was described is the so-called "Cheshire cat" game, the first eight generations of which are shown in Figure 7.2. Various cat-like patterns are displayed ending in a grin and a paw print.

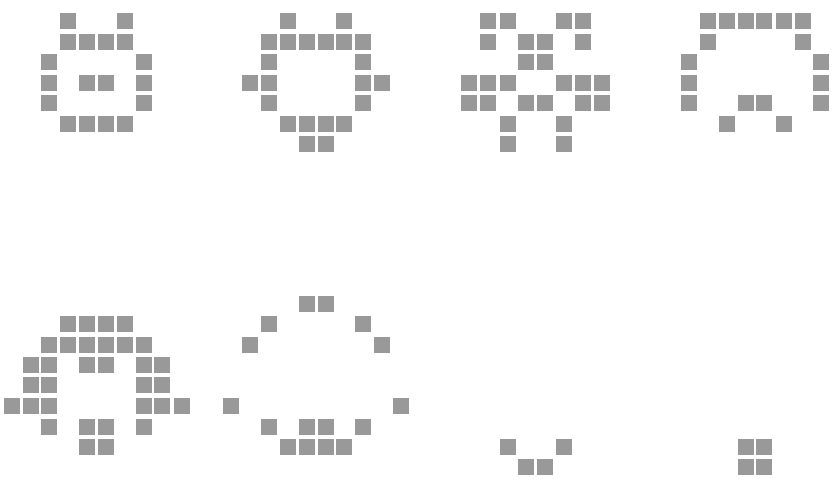

Figure 7.2: The "Cheshire cat" game of life pattern.

The game is best displayed dynamically, but unfortunately this is not very easy in S-PLUS or R since there is no graphics erase facility. It is possible to simulate erasing by over-plotting with colour 0, but this is sometimes unsatisfactory to have running in a loop. Erasing the whole screen and re-plotting may work well on some systems.

It is convenient to allow the user to select the initial occupied squares graphically. One way to do this is with a script such as:

```
plot(c(0,30)+0.5, c(0,30)+0.5, type="n", axes=F,
     xlab="", ylab="", xaxs = "i", yaxs = "i")
abline(h = 0:30+0.5, v = 0:30+0.5)
m <- locator(type="p", col=2, pch=15)
pop <- unique(round(m$x + 1i*m$y))
plot(pop, pch=15, col=2, axes=F, xlab="", ylab="",
     xlim=c(0,30), ylim=c(0,30))
```

To run the game itself the following shorter script can be run repeatedly.

```
for(i in 1:100) {
    pop <- gen(pop)
    plot(pop, pch=15, col=2, axes=F, xlab="", ylab="",
         xlim=c(0,30), ylim=c(0,30))
    if(!length(pop)) break
}
```

If points eventually lie outside the plotting region, though, this will generate large numbers of out-of-bounds plot warning messages.

We can attempt to improve the animation by over-plotting deleted points and only plotting new points. For example, we could use

```
dev.control("inhibit")
plot(pop, pch=15, col=2, axes=F, xlab="", ylab="",
     xlim=c(0,30), ylim=c(0,30))
for(i in 1:100) {
    pop0 <- pop
    pop <- gen(pop)
    gone <- is.na(match(pop0, pop))
    new <- !is.na(match(pop, pop0))
    points(pop0[gone], pch=15, col=0)
    points(pop[new], pch=15, col=2)
    if(!length(pop)) break
#   guiLocator(0)
}
```

(Users of S-PLUS 2000 and 6 for Windows will need `guiLocator(0)` at the end of the loop to ensure that the plot is updated dynamically.) The first line stops the recording of graphics calls and hence reduces the memory used by animations such as this. This disables `dev.copy` and `dev.print`, and possibly also the redrawing of the graphics device, so should be used with care.

Chapter 8

S Software Development

8.3 Creating on-line help

S-PLUS 6.x on Unix: The help files are written in the same SGML specification as S-PLUS 5.1, but converted for use both in HTML-based help and the Jhelp-based `help.start` system. For details, see the next section of these complements.

S-PLUS 6.x on Windows: The current Windows standard, Compiled HTML, is used. The `prompt` function generates files in the same SGML specification as on Unix, with rudimentary documentation in the *Programmer's Guide* pp. 575–7.

The Insightful documentation hints that conversion to Compiled HTML is possible, but does not explain how. We found the tools supplied for 6.0 to be unsatisfactory (and they needed a S-PLUS 6.x Unix installation), and wrote our own conversion scripts in Perl, described on page 13. Version 6.1 has a port to the Cygwin environment of the Unix-based tools in directory `SHOME\help\BuildHelpFiles`, but we continue to use our Perl script `S2html`.

8.4 S-PLUS libraries

Unix, S-PLUS 6.x

We dump and re-boot the chapter to store all the S objects in a few indexed files. If you don't want/need to do this, have converted the help files and use all the standard file extensions then

```
$ Splus CHAPTER
$ Splus make
```

will build the library section for you. (However, note steps 7 and 8.)

1. Create and move to a directory in the library with the section name.

2. Convert the help files if necessary, and arrange for them to have extension `.sgml` and be in the top-level directory. For example, by

```
for f in help/*.d
do
  Splus doc_to_S $f > ‘basename $f‘.sgml
done
```

3. Create a chapter by `Splus CHAPTER filename filename ...`, where the files specified are those to be compiled (and wildcards can be used).

4. Source the files containing S code, for example by

```
$ cat *.q | Splus
```

   Alternatively, you can specify the files to the `CHAPTER` command and use

```
$ Splus make install.fun
```

5. Dump the chapter by `Splus make dump` and remove the `.Data` directory.

6. Run

```
$ Splus CHAPTER filename filename ... *.sgml
$ Splus make boot
$ rm all.S *.Sdata DUMP_FILES
$ Splus make
```

   The last line will both compile the specified files (if any) and install the help files (if any).

7. Create a `README` file in the main directory describing briefly the purpose of the library and with one-line descriptions of the public objects, for use by `library(help=)`.

8. Tidy up by `rm -f .Data/.Audit .Data/.Last.value`.

If you want fuller control, the help can be installed by

```
$ Splus HINSTALL .Data *.sgml
$ Splus BUILD_JHELP
```

The first command installs the SGML files in directory `.Data/__Shelp` and converts them to HTML files in `.Data/__Hhelp`. The second command converts the help to a Jhelp help set in `.Data/__Jhelp`.

The `*.sgml` files have to be in the current directory and to have that extension.

Distributing the library

We distribute our libraries as source code and use a shell script to install it. For example, for `nnet` we use

```
#! /bin/sh
S=${S-Splus}
rm -rf .Data makefile*
$S CHAPTER
echo --reading in S files
cat s5/nnet.q s5/multinom.q | $S
$S make dump
rm -rf .Data makefile*
$S CHAPTER
$S make boot
rm all.S *.Sdata DUMP_FILES
echo "--compiling"
SHOME=‘$S SHOME‘; export SHOME
$S make
rm nnet.o
echo --Installing help
for f in help/*.d
do
   $SHOME/cmd/doc_to_S $f > ‘basename $f .d‘.sgml
done
$S make HELPSGML=*.sgml install.help
rm *.sgml
rm -f .Data/.Audit .Data/.Last.value
```

Windows, S-PLUS 6.x

The procedure is similar to that under Unix. There are three ways to do this, (i) within S-PLUS, (ii) from the command line and (iii) using the Microsoft Visual C++ ‘wizards’ written by Insightful. These are described in the file `help\Chapters.htm`, but unfortunately it seems to be rather inaccurate.

We prefer to use the command line. The following assumes that the S-PLUS `cmd` directory is in your path, as well as the appropriate compiler systems (Microsoft Visual C++ and Compaq Visual Fortran).

For a library section with source files with extension `.q`, `.s` or `.ssc` and C, C++ or Fortran source files (with extensions `.c`, `.cpp` or `.f`) the following will do all that is needed (except tidy up).

```
CHAPTER -b
```

However, we prefer to use a more detailed procedure that re-boots the chapter to put all the objects into a single large file `.Data\__Objects`.

1. Create and move to a directory in the library with the section name, say `mypkg`.

2. Create a chapter by

   ```
   CHAPTER
   ```

3. Source the files containing S code. The simplest way is

   ```
   CHAPTER -s
   ```

   to source all `.q`, `.s` and `.ssc` files, or

   ```
   CHAPTER -s file1.q file2.q
   ```

   to source specific files.
   You can also use the command-line facilities, such as

   ```
   sqpe < file.q
   type *.q | sqpe
   cat *.q | sqpe
   ```

   the last if you have a `cat` utility and using a Unix-like shell.
   Then tidy up by removing the files `.Data\.Audit` and `.Data\.Last.value`.

4. Dump and reboot the chapter by

   ```
   CHAPTER BOOT
   ```

   (or `CHAPTER -o`) then remove the files created for this by

   ```
   del *.Sdata
   ```

5. Compile the source files by

   ```
   CHAPTER -m [optional list of filenames]
   ```

   and tidy up by

   ```
   del *.obj vc60.* S.def S.exp S.lib make.mak chapter.mif
   ```

6. Make the help files. You will need our Perl script `S2html` [1] and Perl itself,[2] as well as Microsoft's HTML Help Compiler.[3] If you are starting from `.sgml` files use

   ```
   perl /path/to/S2html mypkg *.sgml
   ```

   However, if you are using older `.d` help files you can use

   ```
   perl /path/to/S2html mypkg *.d
   ```

   In either case, clean up by

[1] From http://www.stats.ox.ac.uk/pub/MASS3/Sprog/S2html
[2] From http://www.activestate.com/Products/ActivePerl/.
[3] This is supplied with most Windows compiler systems and has also been available for download, most recently from http://www.microsoft.com/office/ork/xp/appndx/appa06.htm and http://msdn.microsoft.com/library/en-us/htmlhelp/html/hwmicrosofthtmlhelpdownloads.asp.

```
del *.html mypkg.hhp mypkg.hhc mypkg.hhk
```

7. Create a file `README.TXT` in the main directory describing briefly the purpose of the library and with one-line descriptions of the public objects, for use by `library(help=mypkg)`.

Distributing the library

Library sections for Windows can be distributed in binary form. The convention is to use a `.zip` file. We do this from the library directory (the parent of the section) by using

```
zip -r9X sec.zip sec/.Data sec/S.dll sec/sec.chm sec/README.TXT
```

8.5 R packages

It is now preferred to add a `Title:` line to the description file rather than have a separate `TITLE` file.

Distributing the package – Unix

There is now a utility `R CMD build` to package up the source files of a package, with a few checks. This can also be used to package a binary distribution.

Distributing the package – Windows

The commands

```
Rcmd check pkg
Rcmd build pkg
```

are available from R 1.2.0 and are almost identical to their Unix equivalents. (`Rcmd build` does build a binary distribution by default.)

Checking the package

[Revised description as from R 1.2.0.]
It is important to check a package before distributing it, and R provides some help in doing so. The commands

```
R CMD check [-l lib.loc] mypkg
```

(Unix) and

```
Rcmd check mypkg
```

(Windows) will run all the examples from all the help files, and report if they ran successfully. They will also run any files in a directory `tests` in the sources, and perform a number of consistency checks including checking the `DESCRIPTION` file, trying to install the package and processing all the help files.

As from R 1.3.0 there are further checks, for undocumented and incorrectly documented objects, and for the correct use of `library.dynam`. R 1.4.0 will have yet further checks.

Chapter A

Compiling and Loading Code

A.1 Procedures with S-PLUS

S-PLUS 6.x on Unix

[This is almost unchanged from S-PLUS 5.1.]

There is only one method available, shared libraries, and the easiest way to include compiled code is to include the files in a call to `Splus CHAPTER`, for example

```
Splus CHAPTER convolve.c
Splus make S.so
```

This will create a shared library called `S.so` in the chapter. Then the next time S-PLUS is started in that chapter, `S.so` will be loaded. Also, whenever a chapter (including a library) is attached the system looks to see if it contains a file `S.so` and if so will load it.

It is possible to use `dyn.open` and `dyn.close` to load or unload a shared library, but this is not normally necessary. Sometimes it easiest to use `dyn.open` to re-load the routines after re-compiling them, although calling `synchronize` on the chapter will unload and re-load `S.so`.

If `Splus CHAPTER` is called with no arguments it will create a `makefile` which will compile and link in all the C and FORTRAN (or Ratfor) files in the directory. If a `makefile` or `Makefile` already exists it will be amended as S-PLUS sees necessary.

The flags for compilation can be changed by setting `CFLAGS`, `FFLAGS` or `CXXFLAGS` as appropriate: this may well be necessary as the default flags omit optimization.

S-PLUS 6.x on Windows

This works completely differently from all previous versions under Windows, and in a much more similar way to S-PLUS 6.x under Unix. When a chapter is attached, `S.dll` is loaded if it exists in the chapter, and it is unloaded when the chapter is detached. Nothing is needed in a `.First.lib` object.

It is possible to use `dyn.open` and `dyn.close` to load or unload a shared library, but this is not normally necessary. Sometimes it easiest to use `dyn.open` to re-load the routines after re-compiling them, although calling `synchronize` on the chapter will unload and re-load `S.dll`, and this is the only reliable way to do this if it was loaded when the chapter was attached.

Information on how to build a suitable DLL will be given in section A.5 of these complements.

A.4 Writing Dynamic Link Libraries for Windows

See section A.5 for S-PLUS 6.x for Windows.

Generating the DLL

Borland C++

Borland C++ 5.5 is available as a free download from http://www.borland.com/bcppbuilder/freecompiler/ and as part of C++ Builder 5. The following will make `conVC.dll` from `conVC.c` on page 241

```
bcc32 -u- -6 -O2 -WDE conVC.c
```

(Flag `-6` optimizes code for a Pentium II/III.)

You can build an import library for `Sqpe.dll` directly from the DLL by

```
implib Sqpe.lib %S_HOME%\cmd\Sqpe.dll
```

and then add `Sqpe.lib` to the `bcc32` command line, for example

```
bcc32 -u- -O2 -WDE -I%S_HOME%\include -D_MSC_VER VCrnd.c Sqpe.lib
```

You will need to define `_MSC_VER` or `__STDC__` to parse the S-PLUS headers correctly. Note that although this example will compile and load it will not work correctly, presumably because of further differences in calling conventions.

You can build an import library for `R.dll` by copying `R.exp` to `R.def` and using

```
implib R.lib R.def
```

and use this by, for example,

```
bcc32 -u- -O2 -WDE -I\R\rw1030\src\include VCrndR.c R.lib
```

This one does work for us.

Other compilers

A key to using other compilers is to find the right flags. One wants a relocatable DLL, with `cdecl` linkage (preferable for S-PLUS but essential under R). Another key issue is to find out how the symbol names get re-mapped ('mangled'), which `pedump` will be able to show. It is normally best to use `.C` to called the compiled code, even if it were generated from FORTRAN, as that does no re-mapping of names under current versions of S-PLUS and R. Thus `.C` can be called with the re-mapped symbol name. (For example, you may need to append or prepend an underscore, and map to upper or lower case.)

A.5 Writing Dynamic Link Libraries for S-PLUS 6 for Windows

S-PLUS 6.x for Windows also uses DLLs, but prefers them to be built by Microsoft Visual C++ using `stdcall` conventions. DLLs built for use with S-PLUS 2000 may well not be usable with S-PLUS 6.x, although simpler ones not linking against the S-PLUS DLL should be.

Much of the advice in section A.4 remains useful, in particular the advice to check the exports on page 244. One of the commonest sources of problems is to build DLLs with no exported symbols.

You will need to install the development tools as part of the installation of S-PLUS: they are not installed by default.

Microsoft Visual C++

The simplest way to build an suitable `S.dll` is to use the `CHAPTER.exe` utility supplied with S-PLUS. Those who enjoy using VC++ projects can consult the *Programmer's Guide*: the installation of S-PLUS adds a new project types called S-PLUS Chapter DLL (.C & .Call) and S-PLUS Chapter DLL (.Fortran)

We will assume both the compiler(s) and `cmd` directory of the S-PLUS 6.x installation are in the path. (Running the batch file `vcvars32.bat` may be the most convenient way to set up the compiler.) Then to create `S.dll` from `convolve.c` all we need is

```
del chapter.mif
CHAPTER -m convolve.c
```

(More than one source file can be supplied on the second line.) The first line is needed if there are any changes to the list of files in use. These commands will also compile FORTRAN files if Compaq Digital Fortran has been installed.

For those who want to know more, the second line runs the commands

```
cl /nologo /MT /W3 /Gz /GX /O2 /FD /YX
   /D "WIN32" /D "_WINDOWS" /D"_MBCS" /D "WINDOWS_CONFLICT"
   /I "%S_HOME%\include" /I "%S_HOME%\spl"  /c
   -o convolve.obj -c convolve.c
```

```
%S_HOME%\cmd\spexport.exe -o S.def convolve.obj

link sqpe.lib spl.lib /DEF:S.def /out:S.dll /implib:S.lib
   /NOLOGO /SUBSYSTEM:windows /MACHINE:I386 /DLL /WARN:1
   /libpath:"%S_HOME%\lib" convolve.obj
```

The crucial option is `/Gz` which selects `stdcall` (Pascal-style) calling conventions. Note the use of Insightful's utility `spexport.exe` to create an exports definition file.

File `conVC.c` (page 241) can be used in exactly the same way.

It is not necessary to build a `stdcall` DLL. We could build a `cdecl` DLL in almost the same way as shown on page 241

```
cl /MT /Ox /D WIN32 /c conVC.c
link /dll /out:S.dll conVC.obj
```

and the `convolve.dll` we built there could be used by re-naming it to `S.dll`.

If one is using `stdcall` conventions it is essential that all the `extern` functions used are declared in header files *and* have the full argument list supplied in the prototype. Any mismatches with the prototype must be resolved.

If we want to call S-PLUS entry points the standard procedures in section A.4 work (although it is not necessary to use the macros `S_DOUBLEVAL` and `S_FLOATVAL`). Thus we can use `VCrndS4.c`, modified to use the S4-style entry points:

```
#include <S.h>
LibExport void urand(long *m, double *p)
{
  S_EVALUATOR
  int i;
  seed_in((long*)NULL, S_evaluator);
  for (i = 0; i < *m; i++)
    p[i] = unif_rand(S_evaluator);
  seed_out((long*)NULL, S_evaluator);
}
```

and can compile and test this by

```
chapter -m VCrndS4.c

> set.seed(123); runif(4)
[1] 0.8756982 0.5321866 0.6700785 0.9921576
> set.seed(123)
> .C("urand", as.integer(4), x=double(4))$x
[1] 0.8756982 0.5321866 0.6700785 0.9921576
```

The header files are set up to use `stdcall` calls where required, so we can also make a `cdecl` DLL by

```
cl /MT /Ox /D "WIN32" /I "%S_HOME%\include" /c VCrndS4.c
link /dll /out:S.dll VCrndS4.obj "%S_HOME%\lib\Sqpe.lib"
```

This does rely on the header files having the correct declarations, and it is wise to check. Most are in the header `S_extern.h` which has

```
LibExtern void S_STDCALL
   seed_in(long *seed_data, s_evaluator *S_evaluator);
LibExtern void S_STDCALL
   seed_out(long *seed_data, s_evaluator *S_evaluator);
LibExtern RETURN_DOUBLE S_STDCALL
   unif_rand(s_evaluator *S_evaluator);
```

The `S_STDCALL` declaration is the critical part.

GNU compilers

The Mingw version of `gcc` which is used to build R can also be used to build DLLs for S-PLUS 6.x. It is most natural to use the `cdecl` conventions. File `conVC.c` provides a simple example. With recent compiler versions[1]

```
gcc -shared -o S.dll conVC.c
```

compiles the file and generates the DLL. However, to use `VCrndS4.c` (which calls entry points in `Sqpe.dll`) we need to use the `stdcall` conventions *via*

```
gcc -c -mrtd -Ic:/S/splus6/include VCrndS4.c
gcc -shared -o S.dll VCrndS4.o -Lc:/S/splus6/lib/mingw -lSqpe
```

using the import library supplied with the S-PLUS development tools. (With recent versions of the Mingw system this causes conflicts with the header files. We resolved these by commenting out the definitions of `erf`, `erfc` and `lgamma` in `S_externs.h`: if your code uses these do take care to check that the correct copies are linked in.)

We can also use the FORTRAN example `testit.f` from page 243; the only change needed is either to rename `testit.dll` to `S.dll` or use

```
> dyn.open("testit.dll")
> .Fortran("testit", as.single(1:5), as.integer(5), as.integer(-2))[[1]]
[1] 1.0000000 0.2500000 0.1111111 0.0625000 0.0400000
> dyn.close("testit.dll")
```

to load and unload the DLL. The simplest way to make the DLL is

```
g77 -O2 -c testit.f
g77 -shared -o testit.dll testit.o
```

What does not work in this simple way is to use FORTRAN code that calls FORTRAN entry points in S-PLUS, because those entry points are `stdcall`. You may be able to use `-mrtd` which switches all calls to `stdcall` but this may be incompatible with other compiled code (for example in system libraries). Fortunately little FORTRAN source does this, but watch out for calls to LINPACK and EISPACK and BLAS routines.

[1] 2.95.2-1 from January 2000 or later.

Watcom C++

The *Programmer's Guide* explains how to use Watcom C++/Fortran 10.5 to build `S.dll`. As those compilers are no longer on sale we will not give details here, but will note that we expect problems with returning doubles and floats as described on page 244, and although the *Programmer's Guide* claims the macros `S_DOUBLEVAL` and `S_FLOATVAL` in `compiler.h` covers this, they do nothing for Watcom compilation (or for any other compiler on this version of S-PLUS).

Borland C++

The Borland examples given earlier for use with S-PLUS 2000 also work under S-PLUS 6.x virtually unchanged.

```
bcc32 -u- -6 -O2 -WDE conVC.c
ren convVC.dll S.dll
```

Also, the following does work correctly:

```
implib Sqpe.lib %S_HOME%\cmd\Sqpe.dll
bcc32 -u- -O2 -WDE -I%S_HOME%\include -D_MSC_VER VCrndS4.c Sqpe.lib
ren convVC.dll S.dll
```

Command-line flag `-p` will select Pascal calling conventions, if needed.

Index

Entries in `this font` are names of S objects.