

Calling Fortran Subroutines in R on a PC

Refer to Brian Habing's webpage for downloading and installing Fortran from the MinGW website. The material presented here is consistent with the portion of Brian's website that deals with running Fortran subroutines in R, with only minor modifications. I thought it appropriate to narrow the focus though, since generally Fortran is already available on the department's PC's. In addition, Brian's first example demonstrates running a Fortran *program* on a PC, as opposed to compiling a Fortran *subroutine*, which is where the primary interest of most R users would lie.

Consider a simple algorithm that computes efficient estimates of \bar{x} and the numerator of s^2 iteratively for $i = 1, \dots, n$:

For starting values $\bar{x}_0 = x_1$ and $s_0^2 = 0$ from a random sample x_1, \dots, x_n , update the estimators as follows:

$$\bar{x}_i = \bar{x}_{i-1} + \frac{x_i - \bar{x}_{i-1}}{i}, \quad i = 1, \dots, n$$

$$s_i^2 = s_{i-1}^2 + (x_i - \bar{x}_{i-1})(x_i - \bar{x}_i), \quad i = 1, \dots, n$$

A Fortran subroutine to implement this algorithm follows. It would be saved in a regular text file using a ".f" extension (in this case, it is saved in the file `mands.f`). Note that the subroutine cannot work independently—the data is not input from the subroutine, but provided to it—and that its arguments includes input and output variables.

```
subroutine mand(n,x,dm,s)
implicit double precision(a-h,o-z)
dimension x(n)
dm=x(1)
ds=0.d0
do 100 i=1,n
  dmold=dm
  dm=dm+(x(i)-dm)/i
  s=s+(x(i)-dmold)*(x(i)-dm)
100 continue
s=sqrt(s/(n-1))
return
end
```

At this point, we compile the program and save it as object code using the PC's command line editor. If using Fortran bothers you, then this part, using Fortran in MS-DOS, should really bother you. But don't worry; we're not in the command line editor for long. I'll follow Brian's instructions here with a couple additional observations along the way.

1. Use either **My Computer** or **Explorer** to make a directory called **Programs** inside the **C:\MinGW** directory.
2. Open either **Notepad** or **Wordpad** and enter the above Fortran code in a new file. Save the file as **mands.f**. In **Wordpad**, you should choose **Text Document** as your file type, while in **Notepad**, you should use file type **All Files**.
3. Choose **Run** under the **Start** menu and enter **command** or **cmd**.
4. Enter the following DOS commands to change to the correct directory:

```
c:  
cd c:\mingw\programs
```

5. Compile your code by entering:

```
g77 -c mand.s.f
```

Note that Brian include a **-O** option which optimizes the object code. This can be left out. This file creates an object file with the output name **mands.o**. The name could be specified by using the **-o** option.

6. Create a **dll** file. It can be loaded and executed in R using either the **.C** or **.Fortran** command.

```
dllwrap --export-all-symbols mand.s.o -o mand.dll
```

7. You may get messages that look like error messages even when the program runs correctly. Just confirm that object and **dll** files have been created before continuing. Return to R, generate a random data set, dynamically load the **dll** file, call the program, and give the output variables useful names.

```
dyn.load("c:/MinGW/Programs/mands.dll")
x1=rnorm(20)
mands=.C("mands_", as.integer(length(x1)),as.double(x1),
as.double(0.0),as.double(0.0))
names(mands)=c("Sample Size","Sample","Mean","Standard Deviation")
```

The output file is a list that you can then inspect; Brian's example returns the first 5 entries in the first element of the list, while I choose to save the list as a variable and assign names to the list elements. When using `.Fortran`, the underscore that appears in the `.C` call is optional. Note that you have to identify the variable format correctly for each variable sent to the subroutine, but you can send either constants or R variables. Brian noted that the `.Fortran` command was not reliable; I had trouble with it in Splus back in 1996, so I guess problems still persist. It worked fine for this simple subroutine.

Brian ends with a function that saves you the trouble of updating the arguments in the `.C` call; it looks pretty useful and could be saved (along with the `dyn.load` command) in a `.First` file if you're going to be using the subroutine frequently. Note that use of a `.First` file may not be so necessary given the ease with which your session's history and workspace can be saved.