Faster R code using C

Dennis Prangle

3rd March 2010

Motivation and overview

Motivation:

- R code inefficient
- So use C(++) for computationally intensive tasks
- Interfacing R and C is fiddly
- Documentation hard to follow

Overview:

- Essentials for simple applications
- Some topics for larger applications
- Miscellaneous other useful things

Warning:

- My knowledge incomplete and may not be best practice!

Part 1 - Essentials

Single slide example - hello world (unix)

- Write C code: `hello.c`

  - include R.h header i.e. `#include <R.h>` (not always necessary)

  - put code into a function: `sayhello`
- Compile C code using R libraries

  - `R CMD SHLIB hello.c` (from terminal)
- Load C code into R

  - `dyn.load("hello.so")`
- Run C code

  - `.C("sayhello")` (in R)

Single slide example - hello world (windows)

- Write C code: `hello.c`

  - include R.h header i.e. `#include <R.h>` (not always necessary)

  - put code into a function: `sayhello`
- Compile C code using R libraries

  - `R CMB SHLIB hello.c` (from terminal)
- Load C code into R

  - `dyn.load("hello.dll")` (in R)
- Run C code

  - `.C("sayhello")` (in R)
- (sometimes) Unload C code

  - `dyn.unload("hello.dll")`

Writing C code to use from R with .C

- Code to be run should be placed in a function e.g `func`

  - a `main` function not needed

- (Following points don't apply if using .Call instead of .C)
- All arguments of `func` must be pointers
- Results should be stored in pointer locations

  - these can be read by R

  - use *original* pointer locations; pointing to something else doesn't work

  - nothing explicitly returned by `func`

Compiling C code to use from R

- Run `R CMD SHLIB file.c`
- Should work in Windows if R set up correctly
- But often problems (e.g. in JNL)
- Alternative: use a C compiler (e.g gcc) directly

  - allows more compilation options to be used

  - I don't know detail of how to do this!

  - (but note that in Unix, using `R CMD SHLIB` with -n option outputs commands it would use)

Running C code from R via .C

- `.C` is an R function used to call a C function
- To call C function `func` write:
  `.C("func",arg1,arg2,...)`
- Arguments must be in types C expects e.g. :
  `.C("func",as.integer(arg1),as.double(arg2), as.logical(arg3))`

  - or `storage.mode(x)="integer"; .C("func", x)`
- `.C` returns a list of final values of arg1, arg2, . . .
- Example: `add.c`
- Limitation: Only scalars and arrays can be passed using `.C`

  - and array length cannot be changed by C

  - `.Call` provides more flexibility (described later)

Part 2 - Other Major Topics

Accessing R functions from C

- C code compiled with R has access to various internal R functions

  - often coded themselves in C
- For example: random number generators, pdf and cdf functions
- Advantages:

  - don't have to write them yourself

  - consistent behaviour with rest of R
- Functions are described in "Writing R Extensions", chapter 6
- C code must include correct headers to use these functions e.g. `Rmath.h`

Accessing R functions from C
Random Numbers - Motivation

- C code can use various random number generators (RNGs)

  - e.g. libraries, write your own
- Advantages of R RNG:

  - good statistical properties

  - replication of runs easy by setting seed in R

  - R has several underlying RNGs; can use several to verify results

Accessing R functions from C
Random Numbers - Details

- Include `Rmath.h` header if required (see below)
- Before any use of RNG read in the state from R:
  `GetRNGstate()`
- Get random numbers
- Either use standard draws

  - `norm_rand()`, `unif_rand()`, `exp_rand()`

  - don't need `Rmath.h` header
- Or use `rnorm(mean,sd)` etc.

  - requires `Rmath.h` header

  - many other distributions available (see "Writing R Extensions", chapter 6)
- After all use of RNG write state back to R:
  `PutRNGstate()`

.Call

- `.Call` is an R function used to call a C function

  - alternative to `.C`

Motivation:

- `.C` can only pass scalars and arrays between R and C

  - difficult to use more complex objects e.g. matrices

  - difficult to have variable length arrays

- `.Call` allows C to use any R object

  - and makes more internal R functions accessible

Drawback:

- *Lots* of complicated syntax to learn i.e. specialised C commands for R objects

Part 3 - Miscellaneous

Interrupting Code

- C code can take much longer than expected
- Helpful to be able to abort and return to R

  - otherwise must kill entire R process
- Can be done by putting `R_CheckUserInterrupt()` in C code

  - somewhere it will be called often (e.g. in a main loop)

  - requires `R_ext/Utils.h` header
- Ctrl+C will now abort C code

Progress Display

- Helpful to tell user how far C code has got
- Can use `printf` function to output progress to terminal

  - to print immediately follow by calling `fflush(stdout)`

R function to handle a C call

- Check input
  - otherwise can get hard to interpret C errors
- Call C code
- Put output into a useful form and return

Drawbacks of calling C from R

- C code takes longer to write than R
- And it's harder to debug

  - for simple code just use `printf` (and `fflush`) statements

  - C debugging tools exist for complicated code e.g. gdb

  - powerful and supported by R but require time to learn
- C errors can crash entire R session (or worse!)

  - so put in lots of error handling
- C code can use up all memory and become very slow

  - plan code to avoid this

  - e.g. split job into chunks, or write results to file rather than keeping in memory

Some alternative approaches

- R can use C++ or Fortran code similarly
- Python
  - `scipy` library enables mathematical objects (e.g. matrices)
  - `rpy` or `rspython` library interfaces with R
  - lots of documentation on interfacing with C
- Don't interface; communicate via data files
  - practical when small number of long jobs

Example code

Available at:
http://www.maths.lancs.ac.uk/~prangle/CinR/

- `hello.c` - hello world example
- `add.c` - addition example
- `CinR.R` - R code for preceding examples
- `reject.c` and `reject.R` - rejection sampling example
  - illustrates most of presentation content

Bibliography

- "S Programming" (2000) - Venables and Ripley
  - useful details in chapter 6 and appendix A but not up to date
- "Software for Data Analysis: Programming with R" (2008) - John Chambers
  - chapter 11 covers including C code
- "Writing R Extensions" (on CRAN) chapters 5 and 6
  - only (almost) full reference on using C from R
  - has more details on everything mentioned here
  - technical; hard to use as a tutorial
  - updated regularly; check for latest version

Other Resources

- R-devel mailing list

  - `https://stat.ethz.ch/mailman/listinfo/r-devel`

  - also quite technical

- R source code

  - the last resort to see how things work, but sometimes the only option!

- Useful C libraries

  - NAG (numerical algorithms group)

  - GSL (GNU scientific library)

  - rcpp (C++ library): nicer C commands to handle R objects from .Call

  (n.b. need a “Makevars” or ”Makevars.win” file to compile code with libraries - see documentation)