C programming III

Karl W Broman

Department of Biostatistics
Johns Hopkins University

http://www.biostat.jhsph.edu/~kbroman

Using R's library

1. Random number generation

The simple functions:

```
double unif_rand();
double norm_rand();
double exp_rand();
```

Surround any calls to R's random number generation with `GetRNGstate();` and `PutRNGstate();` which read and write `.Random.seed`.

You can also get at `rnorm`, `rgamma`, etc. See `Rmath.h`.

```
double rnorm(double mu, double sigma);
double rgamma(double a, double scale);
```

The header files (at `/usr/local/lib/R/include`) are not always sufficiently informative.
Look at `/tmp/R/R-1.8.1/src/nmath` (if it's still there.)

An example

Simulate from the normal/Poisson mixture discussed last time.

C code:

```
#include <stdlib.h>
#include <math.h>
#include <R.h>
#include <Rmath.h>
#include "npmixsim.h"

void npmixsim(double **y, int n_grps, int *n_obs, double *param)
{
  int i, j;
  double a=param[0], b=param[1], sigma=param[2];
  double *lambda=param+3;

  GetRNGstate();

  for(i=0; i<n_grps; i++)
    for(j=0; j<n_obs[i]; j++)
      y[i][j] = rnorm(a+b*rpois(lambda[i]),sigma);

  PutRNGstate();
}
```

An example, part 2

Wrapper:

```
void R_npmixsim(double *y, int *n_grps, int *n_obs, double *param)
{
  double **Y;
  int i;

  Y = (double **)R_alloc(*n_grps, sizeof(double *));
  Y[0] = y;
  for(i=1; i < *n_grps; i++)
    Y[i] = Y[i-1] + n_obs[i-1];

  npmixsim(Y, *n_grps, n_obs, param);
}
```

The R code

```
npmixsim <-
function(n=c(30,30,30,30), theta=c(10,1,0.3, 0.1, 1.1, 2.1, 3.1))
{
  if(length(n)+3 != length(theta))
    stop("length(n) + 3 != length(theta)")

  # load the C code
  if(!is.loaded(symbol.C("R_npmixsim"))) {
    lib.file <- file.path(paste("npmixsim", .Platform$dynlib.ext, sep
    dyn.load(lib.file)
    cat(" -Loaded ", lib.file, "\n")
  }

  output <- .C("R_npmixsim",
               y=as.double(rep(0,sum(n))),
               as.integer(length(n)),
               as.integer(n),
               as.double(theta))

  y <- vector("list",length(n))
  csn <- c(0,cumsum(n))
  for(i in 1:length(n))
    y[[i]] <- output$y[(csn[i]+1):csn[i+1]]

  y
}
```

Using R's library

2. Distribution functions

One can easily get access to oodles of distribution functions that behave just like the versions in R. You need to include `Rmath.h`

Here are the functions for the normal distribution:

```
double dnorm(double x, double mu, double sigma, int give_log);

double pnorm(double x, double mu, double sigma,
             int lower_tail, int give_log);

double qnorm(double p, double mu, double sigma,
             int lower_tail, int log_p);
```

There's also beta, binomial, chi-squared, exponential, F, gamma, etc. See "Writing R extensions", the `Rmath.h` file, or the source code itself: e.g., `/tmp/R/R-1.8.1/src/nmath/dnorm.c`

Using R's library

3. Various mathematical functions

```
double gammafn(double x);
double lgammafv(double x);

double choose(double n, double k);
double lchoose(double n, double k);

double log1p(double x);   /* ln(1+x) for small x */
double expm1(double x);   /* exp(x)-1 for small x */
```

4. Various mathematical contants (accurate to 30 digits)

```
M_E        /* e        */
M_LOG2E    /* log2(e)  */
M_LOG10E   /* log10(e) */
M_LN2      /* ln(2)    */
M_LN10     /* ln(10)   */
M_PI       /* pi       */
M_PI_2     /* pi/2     */
M_SQRT2    /* sqrt(2)  */
etc.
```

Using R's library

5. Optimization

Can get at the code underlying the R function `optim()`, for doing optimizationby Nelder-Mead, BFGS, conjugate gradients, limited-memory BFGS, and simulated annealing (functions `nmmin`, `vmmin`, `cgmin`, `lbfgsb`, and `samin`, respectively). See `src/main/optim.c`.

6. Integration

Can get at the code underlying the R function `integrate()`. The C functions are `Rdqags` and `Rdqagi`, and are in `src/appl/integrate.c`.

Using R's library

7. Sorting routines

There are a bunch of sorting routines:

```
void R_isort(int *x, int n);
void R_rsort(double *x, int n);
```

Also `iPsort`, `R_qsort`, and other things. See "Writing R extensions".

8. Numerical linear algebra

The BLAS, LINPACK, and EISPACK linear algebra functions are included in R.

Look at `/usr/local/lib/R/include/R_ext/Linpack.h`

This is FORTRAN; you need to surround each function call with `F77_CALL`.

For example:

```
F77_CALL(dpoco)(work1, &nparm1, &nparm1, &rcond, param, &info);
```

(When calling FORTRAN, everything must be a pointer.)