Using R API in C and Fortran.

EMCluster[†]

Ranjan Maitra, Volodymyr Melnykov, Wei-Chen Chen

Department of Statistics, Iowa State University

snoweye@iastate.edu

[†]: Provided by Dr. Ranjan Maitra and Volodymyr Melnykov.

Outline

1. Review and Motivation.
2. Controlling R objects in C.
3. R API in `R.h` and `Rmath.h`.
4. Dynamic library examples in C and Fortran.
5. Standalone examples in C and Fortran.

Reivew and Motivation

Review:

- Sigal Blay's web space at `http://www.sfu.ca/˜sblay/present.html`
- Chapter 5 of "Writing R Extensions" for controlling R objects.
- Chapter 6 of "Writing R Extensions" for R API including dynamic libraries and standalone.

Motivation:

- Computing efficience including time and memory.
- Dynamic or recursive programming.
- MCMC.

Controlling R objects in C

Basic steps:

1. In R, there are two methods to directly pass the R objects to C, .Call() and .External().
2. In C, there are two methods to handle R objects by including "Rdefines.h" and "Rinternals.h" as the header files.
3. R objects use a structure type SEXP as a pointer in C.
   - Allocate new R objects.
   - Protect new R objects.
   - Duplicate objects passed from R if any modification is required. Any object passed from R should be treated as read-only in C.
   - Computing. Use more R API here.
   - Unprotect all protected R objects.
4. Return the R object to R from C.

What else?

- Sec. 5.11 Evaluating R expressions from C.
  `SEXP eval(SEXP expr, SEXP rho);`
- Sec. 5.12 Parsing R code from C.
  `Rinternals.h` and `R_ext/Parse.h`.
- Control R objects in Fortran??

R API in `R.h` and `Rmath.h`

Functions:

- `Rprintf()` as `printf()` in C.
- `R_alloc()` as `malloc()` in C.
- `Free()` as `free()` in C.

Steps for Random Number Generators in Dynamic Library:

1. `GetRNGstate()`
2. `runif()`, `rnorm()`, ...
3. `PutRNGstate()`

Steps in standalone: (libRmath.so or libRmath.dll)

1. `#define MATHLIB_STANDALONE`
2. `set_seed(unsigned int, unsigned int)` or
   `get_seed(unsigned int *, unsigned int *)`
3. `unif_rand()`, `norm_rand()`, ...

srswor() in EMCluster

```
int srswor(int n, int k, int *y)
{
    /* Provide k out of n indices sampled at random without replacement */
    if (k > n) {
        printf("Error: k = %d  greater than n = %d  in srswor()\n", k, n);
        return 1;
    } else {
        int i, j;
        int *x;
        MAKE_VECTOR(x, n);
        for (i = 0; i < n; i++) x[i] = i;

        GetRNGstate();
        for (i = 0; i < k; i++) {
          j = n * runif(0, 1);
          y[i] = x[j];
          x[j] = x[--n];
        }
        PutRNGstate();

        FREE_VECTOR(x);
    }
    return 0;
}
```

Stable APIs

- Sec. 6.1 Memory allocation.
- Sec. 6.3 Random number generation.
- Sec. 6.7 Numerical analysis subroutines.
  `Rmath.h`, `R_ext/BLAS.h`, `R_ext/Lapack.h`, and `R_ext/Linpack.h`.
  Distribution functions, Mathematical functions, ...
- Sec. 6.8 Optimization.
- Sec. 6.9 Integration.
- Sec. 6.10 Utility functions.
- Sec. 6.17 Organization of header files.
- ...

C Wrapper for Frotran and vice versa

In section 6.6,

- `F77_SUB(name)` to define a function in C to be called from FORTRAN
- `F77_NAME(name)` to declare a FORTRAN routine in C before use
- `F77_CALL(name)` to call a FORTRAN routine from C
- `F77_COMDECL(name)` to declare a FORTRAN common block in C
- `F77_COM(name)` to access a FORTRAN common block from C
- See `R_ext/RS.h` for detail.

More examples

- Dynamic library in C.
- Standalone in C.
- Dynamic library in Fortran with a C wrapper.
- Standalone in Fortran with a C wrapper.

Dynamic library in C

```
#include <R.h>
#include <Rmath.h>

void callR(){
  int i;
  double mu, sigma, PHI_X, *X;

  mu = 0;
  sigma = 1;

  X = (double *) R_alloc(10, sizeof(double));

  Rprintf("Before sort\n");
  GetRNGstate();
  for(i = 0; i < 10; i++){
    X[i] = rnorm(mu, sigma);
    PHI_X = pnorm(X[i], mu, sigma, 1, 0);
    Rprintf("X: %f, PHI(X): %f\n", X[i], PHI_X);
  }
  PutRNGstate();
  R_rsort(X, 10);
  Rprintf("After sort\n");
  for(i = 0; i < 10; i++){
    PHI_X = pnorm(X[i], mu, sigma, 1, 0);
    Rprintf("X: %f, PHI(X): %f\n", X[i], PHI_X);
  }
}
```

Linked with "-lR -lRmath".

Standalone in C

```
#define MATHLIB_STANDALONE
#include <Rmath.h>

int main(){
  int i;
  unsigned int SEED1, SEED2;
  double mu, sigma, PHI_X, *X;

  mu = 0;
  sigma = 1;
  SEED1 = 12345;
  SEED2 = 67890;
  set_seed(SEED1, SEED2);

  X = (double *) malloc(10);
  for(i = 0; i < 10; i++){
    X[i] = rnorm(mu, sigma);
    PHI_X = pnorm(X[i], mu, sigma, 1, 0);
    printf("X: %f, PHI(X): %f\n", X[i], PHI_X);
  }
}
```

Linked with "-lRmath" only.

Dynamic library in Fortran with a C wrapper

Fortran:

```
c A subroutine in "callc.f"
      subroutine testit(x, y)
      real*8 normrnd, unifrnd, x, y

      call rndstart()
      x = normrnd()
      y = unifrnd()
      call rndend()

      return
      end
```

C:

```
#include <R.h>
#include <Rmath.h>

void F77_SUB(rndstart)(void) { GetRNGstate(); }
void F77_SUB(rndend)(void) { PutRNGstate(); }
double F77_SUB(normrnd)(void) { return rnorm(0, 1); }
double F77_SUB(unifrnd)(void) { return runif(0, 1); }
```

R:

```
.Fortran("testit", as.double(1), as.double(1))
```

Standalone in Fortran with a C wrapper

Fortran:

```
c A main function
      program main
      real*8 a, b

      call setseed(123, 456)
      call testit(a, b)

      print *, a, b
      end
```

```
c A subroutine
      subroutine testit(x, y)
      real*8 normrnd unifrnd, x, y

      x = normrnd()
      y = unifrnd()

      return
      end
```

C:

```
#define MATHLIB_STANDALONE
#include <R_ext/RS.h>
#include <Rmath.h>

void F77_SUB(setseed)(int a, int b){ set_seed(a, b); }
double F77_SUB(normrnd)(void){ return norm_rand(); }
double F77_SUB(unifrnd)(void){ return unif_rand(); }
```

Strategy

Initial programs from R (dynamic library):

1. Prepare and check the data in R.
2. Pass the objects to a wrapper in C.
3. Compute in C or Fortran and use R APIs.
4. Copy the results to the wrapper.
5. Return the results to R. Summarize and plot them.

Comments or Questions

Thank you!