Foreign Language Interfaces
Self-Study Exercises

Valerie Obenchain

Fred Hutchinson Cancer Research Center

17-18 February, 2011

1 Introduction

These exercises will take you though the steps necessary to include compiled C code in your course package. For additional information on the material covered here see the Writing *R* Extensions Manual, Section 5: System and Foreign Language Interfaces, http://cran.r-project.org/doc/manuals/R-exts.html.

In this lab we will be using a C function that computes linkage disequilibrium (LD). Linkage disequilibrium refers to association between SNPs. Alleles at different loci sometimes appear together more or less often than expected based on their frequencies. Two markers are said to be in LD if there is an statistical association between the alleles.

To compute LD we need the haplotype phase. Because our data are simulated we do not have this information. Therefore our LD function computes a 'composite' LD which is a statistical measure of association across loci when haplotype phase is not known. See Weir and Cockerham, 1989; Weir 1996 pp. 94, 125.

We will start by compiling the C code and loading it into an *R* session for testing. Next the package NAMESPACE file will be modified so the dynamic library will be automatically loaded when the package is loaded. The final step will be to register the C method with *R*.

Exercise 1
In this excercise the C code is compiled into a shared object and loaded into an R session for testing.

Question 1

- *Move the C function composite_linkage_disequilibrium.c and the header composite_linkage_disequilibrium.h from* `StudentGWAS/src/` *into the src directory of your package. Compile the C function using R CMD SHLIB.*

Solution:

```
R CMD SHLIB composite_linkage_disequilibrium.c
```

Question 2

- *Before loading the shared object in the package, you may want to perform some basic testing. Any shared object can be loaded into R by using the* `dyn.load` *command. All functions in the shared object are now made available to R. Start an Rsession and load the shared library with* `dyn.load`*.*
- *Call the composite linkage disequilibrium function from R using the .C function.*

Solution:

```
dyn.load("composite_linkage_disequilibrium.so")

## Create sample data for testing
snps <- matrix(sample((1:3), replace=TRUE, 400), nrow=100, ncol=4)
nsnp <- ncol(snps)
nsub <- nrow(snps)
width <- 2
delta <- rep.int(0, (nsnp-width)*width)

out <- .C("composite_linkage_disequilibrium",
          snp = as.raw(snps),
          n_ind = as.integer(nsub),
          n_snp = as.integer(nsnp),
          width = as.integer(width),
          delta = as.double(delta))
out$delta
```

Question 3

Create a more convenient interface to the linkage disequilibrium function by writing an R wrapper function. By default the .C function returns a list. Make the wrapper function return a matrix with dimensions of snp by width. Include the snp names as rownames. Name the the wrapper `.cld.R` *and put it in the /R directory of your package.*

Solution:

```
> .cld <- function(data, width = 5)
+ {
+     nsub <- nrow(data)
+     nsnp <- ncol(data)
+     if (width > nsnp)
+         stop("Width must be less than the number of snps.")
```

```
+     delta <- rep.int(0, (nsnp-width)*width)
+     res <- .C("composite_linkage_disequilibrium",
+               snp = as.raw(data),
+               n_sub = as.integer(nsub),
+               n_snp = as.integer(nsnp),
+               width = as.integer(width),
+               delta = as.double(delta), PACKAGE="StudentGWAS")
+     matrix(res$delta, nrow=(nsnp-width), ncol=width, byrow=TRUE,
+            dimnames = list(colnames(data)[1:(nsnp-width)], NULL))
+ }
```

Exercise 2
In this exercise the package NAMESPACE is modified to automatically load the shared library when the package is loaded. All C code in the /src package directory is compiled into a single shared object with the same name as the package.

Question 4

- *Using* `useDynLoad`*, modify the NAMESPACE to load the dynamic library for the package*

Solution:

```
useDynLib{StudentGWAS}
```

Exercise 3
The purpose of this exercise is to register the C function with R.

When a shared object (or dynamic library) is loaded, R looks for a routine within that shared object named R_init_mypkg. If such a routine is present, R will invoke it. This is a convenient way of executing some code automatically when a shared object is loaded or unloaded. We use this function to register native routines with R. When .C, .Call or .Fortran is used, R must locate the specified native routine by looking in the shared object. Registering a native routine with R allows the use of a platform-independent mechanism for finding the routines in the shared object instead of an operating system-specific method to lookup the routine.

To register routines with R we use the C routine R_registerRoutines. It takes 5 arguments, the first is the shared object information followed by arrays describing the routines for each of the 4 different interfaces: .C, .Call, .Fortran and .External. These arrays are created with the appropriate entry from the following table :

```
.C            R_CMethodDef
.Call         R_CallMethodDef
.Fortran      R_FortranMethodDef
.External     R_ExternalMethodDef
```

Question 5

- *Create a file called R_init_StudentGWAS.c and put it in /inst/src*
- *Define the linkage disequilibrium function using R_CMethodDef*
- *Call R_registerRoutines using the array defined with R_CMethodDef*
- *Wrap the R_CMethodDef array declaration and the R_registerRoutines function in a function called R_init_StudentGWAS*

Solution:

```
 /* R_init_StudentGWAS.c file */

#include <R_ext/Rdynload.h>
#include "composite_linkage_disequilibrium.h"

void R_init_StudentGWAS(DllInfo *info)
{
    /* Create the R_CMethodDef array */
    R_NativePrimitiveArgType cld_t[5] =
        { RAWSXP, INTSXP, INTSXP, INTSXP, REALSXP };

    R_CMethodDef cMethods[] = {
        {"composite_linkage_disequilibrium",
         (DL_FUNC) &composite_linkage_disequilibrium,
         5, cld_t},
        {NULL, NULL, 0}
    };

    /* Register the routine */
    R_registerRoutines(info, cMethods, NULL, NULL, NULL);
}
```

Exercise 4

As an advanced exercise, create a C version of the heterozygosity function introduced in the EfficientR-lab section. Create an R wrapper for this C function and register the method.