R and Other Languages

Edouard T. Djeutem, Biljana J. Stojkova
under supervision of Dave Campbell

August 8, 2013

1 Introduction

This project is consisted of two parts. The part I explains how to integrate R with C++ using Rcpp and inline libraries, and the second part explains how to integrate R with C using R CMD SHLIB which has integrated gcc compiler. Both parts are supported by working examples, including all neccessary explanations. There is a separate .R code file for each of these two parts:

- for the first part the .R file is named *UsingRcpp_Edouard_Biljana.R* along with the .csv data file named cobb.csv, and
- for the second part the .R file is named *Biljana_Edouard_ProjectI.R* and it goes along with the bootvar3.C file and bootvar3.dll file (which is compiled code of the .C file).

2 Part I: R and C++ using Rcpp and inline libraries

2.1 Rcpp and inline libraries overwiew

The Rcpp library provides R functions and C++ library that enables C++ and R integration using the .Call() interface provided by R. Rcpp supports all data types such as vectors, function, environment. Each of these types is matched to appropriate C++ class. Numeric vectors are represented as instances of Rcpp::NumericVector Class, DataFrame objects are represented as instances of Rcpp::DataFrame class. Conversion from C++ to R is enabled by the template class Rcpp::wrap which is flexible and transforms an arbitrary object into a SEXP. This Rcpp class enables implementation of C++ logic in terms of standard C++ types such as STL and then wrap them in order to be returned to R. The reverse coversion from R to C++ is drive n by Rcpp::as that offers similar level of flexibility as Rcpp::wrap. Rcpp also provides a framework known as Rcpp modules that allows using C++ classes and functions to the R level.

As of version *Rcpp*_0.8.1, usage of an extended function "cxxfunction" in enabled.The cxxxfunction requires inline library in addition to the Rcpp.The cxxfunction function enables easier deployment of C++ code with Rcpp. In more depth, it enforces the usage of .call() interface, adds the Rcpp namespace and features exception forwarding. It employs *BEGIN_RCPP* and *END_RCPP* macros to wrap the user code. In addition, with cfunction and cxxfunction, we can call external libraries and have them linked as well.

2.2 The cxxfunction

The main feature of our example is the function cxxfunction from the package inline. The cxxfunction provides dynamically creating R function with inlined C++ code using the .Call calling convention.

cxxfunction(sig = character(), body = character(), plugin = "default", includes = "", settings = getPlugin(plugin), ..., verbose = FALSE)

rcpp(...)

Arguments:

- sig - Signature of the function which is a named character vector
- body - A character vector containing C++ code that has to be compiled
- plugin -Name of the plugin to use. cxxfunction uses plugin system to assembly the code that it compiles.
- includes - includes auxiliary functions if there are any
- settings - Result of the call to the plugin
- verbose - additional arguments to the plugin, verbose output boolean

3 System requirements

3.1 Tools that have to be installed

- In order for Rcpp package to run, GNU Compiler Collection (or gcc) along with the corresponding g++ compiler for the C++ language need to be installed
- Depending on the operative system used, different tools should be installed:
  - Windows users need the Rtools package developed and maintained by Duncan Murdoch, which comprises all needed tools for Rcpp to run along with instructions and manuals.
  - OS X users need to install Apple Developer Tools (e.g. Xcode)
  - Linux users need to install the standard development packages such as r-base-dev package on Debian and Ubuntu

3.2 Install Rcpp on Windows 7

- Install R: it should be installed on a path that does not contain spaces
- Install Rtools also on a path that does not contain spaces. The version of the Rtools should be compatible with the version of R
- Edit PATH Environment Variable to point to the following addresses:
  - C:\Rtools\bin;
  - C:\Rtools\gcc-4.6.3\bin;
  - C:\R\R-2.15.1\bin;
  - C:\R\R-2.15.1\batchfiles;
  - C:\Rtools\MinGW \bin;

3.3 Additional installation tips

- Rcpp and inline libraries should be installed on the path with no spaces.
- After following all the previous steps, Rcpp and inline should work.
- If still there is a warning from CYGWIN including the keyword "nodosfilewarning", the new environment variable should be created with name "CYGWIN" and value "nodosfilewarning".
- There is a nice web resource that explains in details the process of installing Rcpp: `http://tonybreyal.wordpress.com/2011/12/07/installing-rcpp-on-windows-7-for-r-and-c-inte`

4 Working example overview

Our example is a performance study that compares running time of C++ for-loop written executed from R with several R methods such as traditional for-loop,apply function, parallelized function and matrix multiplication. The results and conclusions are as expected: Rcpp execution time is the smallest of all. Using vectors instead of for loop showed to be the second fastest method. Traditional for-loop exhibited the worst running.

4.1 Cobb Douglas production function model

We use the model of Cobb Douglas production function This model measures elasticity of the labor force and capital According to Cobb-Douglas production function, increased input in both capital and labor force should lead to increased production output In this example we work with some real data. Variables are: lnY is gross production output lnK is capital input lnL is labor force (number of employees) input The model is:
$Y = K^{\beta} * L^{\gamma} * Error^{U}$

so we take logarithm of both sides thus turning the model into double logarithmic multiple regression. We fit the model and the use all above mentioned methods to generate a dataset which is obtained from binding the original dataset with a variable that contains fitted values of our regressant

Running time of all methods is plotted in order to present results from our performance study.

4.2 Preparing C++ code for cxxfunction

We begin our performance study by loading the data from .csv file and fitting the double logarithmic model in order to obtain intercept and slope coefficients.

```
> install.packages("Rcpp",lib="G://",destdir="G://",repos="http://cran.r-project.org")

package Rcpp successfully unpacked and MD5 sums checked

> install.packages("inline",lib="G://",destdir="G://",repos="http://cran.r-project.org")

package inline successfully unpacked and MD5 sums checked

> install.packages("rbenchmark",repos="http://cran.r-project.org")

package rbenchmark successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\authorized user\AppData\Local\Temp\Rtmp8gCeji\downloaded_packages
```

```
> install.packages("snow",repos="http://cran.r-project.org")

package snow successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\authorized user\AppData\Local\Temp\Rtmp8gCeji\downloaded_packages

> install.packages("ggplot2",repos="http://cran.r-project.org")

package ggplot2 successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\authorized user\AppData\Local\Temp\Rtmp8gCeji\downloaded_packages

> library(Rcpp,lib="G://")
> library(inline,lib="G://")
> library(rbenchmark,lib="G://")
> library(snow)
> library(ggplot2)

> #read data from .csv file and put them into a data.frame
> cobb.dat1<-read.table("C:\\Users\\authorized user\\Dropbox\\STAT853\\FirstProject\\myProjec
> ds=data.frame(cbind(lnY=cobb.dat1$lnY,lnK=cobb.dat1$lnK,lnL=cobb.dat1$lnL))
> #fit the data into a double logarithmic model
> fitModel = lm(ds$lnY ~ ds$lnK+ds$lnL,data=ds)
> #read the coefficients
> coef=coefficients(fitModel)
```

4.3 Using cxxfunction

Our goal is to generate a new dataset obtained from binding the original dataset with fitted values from our regression model. So we write an auxiliary function *FittedY_cxx* that returns fitted values of the production output variable using original dataset and estimated regression model. *FittedY_cxx* is written in C++ syntax and this C++ syntax function in wrapped in string variable so that this string could be executed latter using inline cxxfunction function.

```
> FittedY_cxx_string <-
+   double FittedY_cxx (double lnK, double lnL){
+   double  prod =  0.45+0.34*lnK+0.41*lnL;
+   return(prod);
+   }
+
```

The main C++ code retrieves columns from the original dataset, executes the auxiliary function *FittedY_cxx* in the C++ for-loop over provided dataset (*FittedY_cxx* returns fitted values of production output *lny_hat*) and then binds original dataset with fitted value *lny_hat*.

```
>   rcpp_string <-
+   // using Rcpp::DataFrame object retreive data from the original dataset in R
+   // and pass them in C++
+   Rcpp::DataFrame ds(the_ds);
+   // extract columns and put them in C++ vectors
+   // Rcpp::as template class used to pass dataset columns from R to C++
+   std::vector<double> lnY_c = Rcpp::as< std::vector<double> >(ds["lnY"]);
+   std::vector<double> lnK_c = Rcpp::as< std::vector<double> >(ds["lnK"]);
+   std::vector<double> lnL_c = Rcpp::as< std::vector<double> >(ds["lnL"]);
+
+   // initialize a new variable y_hat
+   // this is done on the C++ side only
+   std::vector<double> y_hat(lnL_c.size());
+
+   // iterate over data frame to calculate y_hat
+   // using the auxiliary function FittedY_cxx
+   for (unsigned int i = 0; i < y_hat.size() ; i++) {
+   y_hat[i] = FittedY_cxx(lnK_c[i],lnL_c[i]);
+   }
+
+   // export original dataset combined with y_hat from C++ in R using Rcpp::DataFrame::creat
+   return Rcpp::DataFrame::create( Named("lnY")= lnY_c, Named("lnK") =lnK_c,Named("lnL") = l
+
```

So we have main c++ code and auxiliary function in two strings variables that are waiting to be executed. We execute the both c++ code strings using inline function cxxfunction.

```
>  results_rcpp <- cxxfunction(signature(the_ds="data.frame"),
+                              body=rcpp_string, plugin="Rcpp",
+                              includes=c(#include <cmath>,FittedY_cxx_string))
```

Results are saved into a *results_rcpp* variable. Latter it will be used for the purpose of measuring the running time of Rcpp - inline approach.

4.4 Measuring the running time of other methods

The same dataset and the same model has been used to perform the same tasks that Rcpp function cxxfunction performed. Traditional for loop executed function FittedY (that returns fitted values of the production output variable) for each dataset point thus returning bonded original dataset with fitted values returned from FittedY function. Apply method executed function FittedY for each dataset point thus returning bonded original dataset with fitted values returned from FittedY function. Vectored version used matrix multiplication, thus avoiding for loops and apply. Since matrix multiplication is quite fast, we can see that this is the second best method after Rcpp.

```
>   # return fitted values from regression equation for production variable
>   # input are values of the variables from original dataset
>   # Analogous function to the C++ function FittedY_cxx_string used above.
>   FittedY = function(lnK,lnL) {
+     prod <- coef[1]+coef[2]*lnK+coef[3]*lnL
+     return(prod)
+   }
>   # add the fitted values of production to the new dataset
>   # using the traditional for loop
>   for_loop <- function(i,dataset=ds) {
+     i=i+1
+     prod <- vector(length=nrow(dataset),mode="numeric")
+
+     for (x in 1:nrow(dataset)) {
+       prod[x] <- FittedY(dataset$lnK[x],dataset$lnL[x] )
+     }
+     data.frame(cbind(dataset,y_hat=prod))
+   }
>   # add the fitted values of production to the new dataset
>   # using the apply function
>   my_apply = function(dataset) {
+     prod <- apply(ds, 1, function(x) {FittedY(x[2],x[3])})
+     data.frame(cbind(dataset,prod))
+   }
>   # add the fitted values of production to the new dataset
>   # using the matrix multiplication - vecorized version
>   my_vectorize = function(dataset) {
+     prod =as.matrix(cbind(matrix((rep(1,nrow(dataset)))),dataset[,c("lnK","lnL")]))%*%matri
+     data.frame(cbind(dataset,prod))
+   }
>
```

In the end we use the rbenchmark library that serves as a wrapper around system.time. This package allows user to set up counts of replications so we can measure how each method behaves for different number of replications. So here, each of the methods (forloop,*my_apply*,*results_rcpp*,*my_vectorize*) are executed for number of replications c(10,100,500,1000,1500,10000)

```
>   # Benchmark is a wrapper around system.time
>   # this package allows user to set up the number of replications
>   # so we can measure how each method behaves for different
>   # number of replications
>   # So here, each of the methods (forloop,my_apply,results_rcpp,my_vectorize)
>   # are executed for  number of replications c(10,100,500,1000,1500,10000)
>   sizeset=c(10,100,500,1000,1500,10000)
>   for (x in sizeset) {
+
+
+     bm_results <- benchmark(for_loop(ds),
+                             my_apply(ds),
+                             results_rcpp(ds),
+                             my_vectorize(ds),
+                             replications=x,
+                             columns=c(test, elapsed, replications))
+     if (x==10) {
+     mat=bm_results
+     }else{
+     mat=rbind(mat,bm_results)
+     }
+   }
```

In the end we perform paralleling using *my_parallel* function defined above

```
>   # add the fitted values of production to the new dataset
>   # using the parallelized code
>   my_parallel = function(param,dataset2=ds) {
```

```
+     vec=matrix(1:param)
+     c(system.time(apply(vec,1,for_loop,dataset=dataset2))[[1]],param)
+   }
>   #Parallelizing
>   cl =makeCluster(2, type = "SOCK")
>   clusterExport(cl, list = ls())
>   ffited = unlist(clusterApplyLB(cl, sizeset,my_parallel))
>   stopCluster(cl)
>   ffited=t(matrix(ffited,nrow=2))
>   colnames(ffited)=c("elapsed", "replications")
>   ffited=data.frame(ffited)
>   ffited$test=matrix(rep("My_parallel",nrow(ffited)))
>   mat=rbind(mat,ffited)
>
```

4.5 Presenting the plot and conclusions

In the end we generate a plot that present results from our performance study. From the plot, it can be clearly seen that

- the Rcpp methods have the ”best” running time.
- the traditional for loop have the ”Worst” execution time.
- The second best running time is for the vectored functions, using matrix multiplication.
- Apply and paralleling are between vectored functions and traditional for-loop method.

5 Part II: R and C

In this part we are using R CMD SHLIB to compile the bootvar3.c if the computer is 32 bits or R –arch x64 CMD SHLIB if the computer is 64 bits.

Normally to compile a C code we use gcc compiler which is already integrated in R. That is why we can use R CMD SHLIB bootvar3.c instead of gcc -o bootvar3.c.

Alternatively, we can simply set the working directory properly using setwd() and system(“R CMD SHLIB bootvar3.c”). If everything goes smoothly, a file bootvar3.dll will be generated, which is actually compiled C code that is written in C. Then we load in R the C code by using dyn.load(”bootvar3.dll”). That dll is indeed loaded using is.loaded(”bootvar3.dll”).

5.1 Working example overview

We are going to estimate Bootstrap variances using vectorized code written in R and a compiled version from C and then we will compare running time for different number of replications and sample size and present in plot.

Consider a sample $\{Y_t\}_1^T$ with $Y_t \sim iid\mathcal{N}(0, \sigma^2)$ and the following estimator of the variance:

$$\hat{\sigma}^2 = \frac{1}{T}\sum_{t=1}^{T}(Y_t - \bar{Y})^2 \tag{1}$$

This is the same example we covered in class.

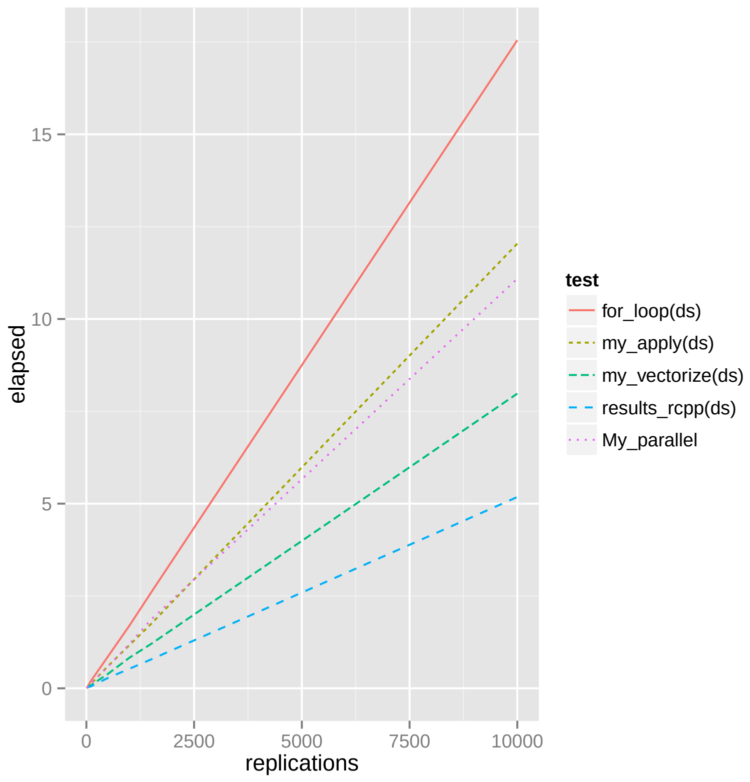

Figure 1: Performance Study, R and C++

```
> #Checking the Available packages
> checkpackages=function(package){
+   if (!package %in% installed.packages())
+     install.packages(package,repos="http://cran.r-project.org")
+ }
> listpackage=c("reshape","gglot2","snow")
> lapply(listpackage,checkpackages)

[[1]]
NULL

[[2]]
NULL

[[3]]
NULL

> #Set the work directory such that R can access the complied C code
>
> setwd("C://Users//authorized user//Dropbox//ProjectIStat890//Submission")
> library(reshape)
> library(ggplot2)
> library(snow)
```

Next, we call a Wrapper to call the C subroutine

Input Y: Matrix of observations. Each locum corresponds to a sample Output VarY: Bootstrap variance

```
> bootvarC <- function(Y)
+ {
+   .C("bootvar3",
+      as.double(Y),
+      as.integer(nrow(Y)),
+      as.integer(ncol(Y)),
+      as.double(Y),
+      varY = double(ncol(Y)) # Should always be equal to the length of the vector we want ba
+   )$varY
+ }
```

We write a function to estimate bootstrap std by VECTORIZATION Input Y: Matrix of observations. Each column correspond to a sample Output : Bootstrap variance

```
> bootVectorize = function(Y){
+   # variance calculated with the wrong denominator
+   res=(Y-mean(Y))
+   # Recall that linear algebra is way way way way way way better than using a for loop
+   return(res%*%res/length(Y))
+ }
```

And a function to evaluate running time

Input Y: Matrix of observations. Each locum correspond to a sample. Output : Bootstrap variance.

```
> runtime = function(param,mu0=0,sd0=1){
+   nn=param[1];
+   BB=param[2];
+   Y=rnorm(n=nn,mean=mu0,sd=sd0)
+   bootsample=matrix(rnorm(n=nn*BB,mean=mean(Y),sd=var(Y)),nrow=nn,ncol=BB)
+   return(list(system.time(apply(bootsample,2,bootVectorize))[[1]],"Vec",
+               system.time(bootvarC(bootsample))[[1]],"C",nn,BB))
+ }
```

Main Program

This part compiles C code directly in R using R CMD SHLIB, so the C code is compiled into a .dll file which is then loaded back to R.

```
>   #compile the C code directly in R or in command line using
>   system("R --arch x64 CMD SHLIB bootvar3.c") #system("R  CMD SHLIB bootvar3.c") if 32bits 
>   #Upload the compiled C code
>   dyn.load("bootvar3.dll")
>   #Checking that the dll is indeed uploaded
>   is.loaded("bootvar3")

[1] TRUE
```

Then we test whether R and C code give the same results.

```
>     n=25;
>     B=1000;
>     Y=rnorm(n=25,mean=0,sd=1)
>     bootsample=matrix(rnorm(n=n*B,mean=mean(Y),sd=var(Y)),nrow=n,ncol=B)
>     #Executing the C code
>     sigmaboot=bootvarC(bootsample)
>     #R code
>     bootV1   = apply(bootsample,2,bootVectorize)
>     #Ploting the results to see if they differ
>     varlist=list(sigmaboot,bootV1)
>     varname=c("C","R")
>     stat.df=NULL
>     for (k in 1:length(varlist)){
+       temp.df=varlist[[k]]
+       temp.df=melt(data.frame(temp.df))
+       temp.df$var=matrix(rep(varname[k],nrow(temp.df)))
+       stat.df=rbind(stat.df,temp.df)
+     }
>     p = ggplot(stat.df, aes(x=value, colour=var))
>     p=p + geom_density() + labs(list(title = "bootstrap variance estimates ", x = "", y = "
>     print(p)
```

Time performance evaluation

```
>   #vector of sample size
>   nset=c(25,50,75,100,125,200)
>   #vector of number of replications
>   Bset=c(25,1e2,1e3,1e4,1e5,2e5)
>   simset= expand.grid(n =nset, B =Bset)
>   runtime=data.frame(simset)
>   for (k in 1:nrow(simset)){
+     Y=rnorm(n=simset[k,1],mean=0,sd=1)
+     bootsample=matrix(rnorm(n=simset[k,1]*simset[k,2],mean=mean(Y),sd=var(Y)),nrow=simset[k
+     runtime$vec[k]=system.time(apply(bootsample,2,bootVectorize))[[1]]
+     runtime$C[k]=system.time(bootvarC(bootsample))[[1]]
+   }
>   #Ploting the running time
>   time.df=melt(runtime,id=c("n","B"))
>
```

5.2 Resources

http://stackoverflow.com/questions/4916853/
problem-with-loading-compiled-c-code-in-r-x64-using-dyn-load http://www.stat.ubc.ca/~webmaste/howto/faq/windows/rlinkwin.html

http://www.stat.nctu.edu.tw/~misg/SUmmer_Course/C_language/Ch15/RCallCinWindows.txt

http://www.info-emilio.net/programming/interfacing-c-in-r/

http://www.sfu.ca/~sblay/R-C-interface.txt

http://romainfrancois.blog.free.fr/index.php?category/R-package/Rcpp

Other Resources

Venables and Ripley (2000, Appendix A),"S programming", Springer

"An Introduction to the .C Interface to R" by Roger D. Peng Jan de Leeuw, UCLA Department of Statistics.

```
>   ggplot(time.df, aes(x=B, y=value, colour =variable))+ facet_wrap( ~n)+geom_line()
>   labs(list(title = "", x = "Number Replications" , y = "Time"))

$title
[1] ""

$x
[1] "Number Replications"

$y
[1] "Time"

attr(,"class")
[1] "labels"
```

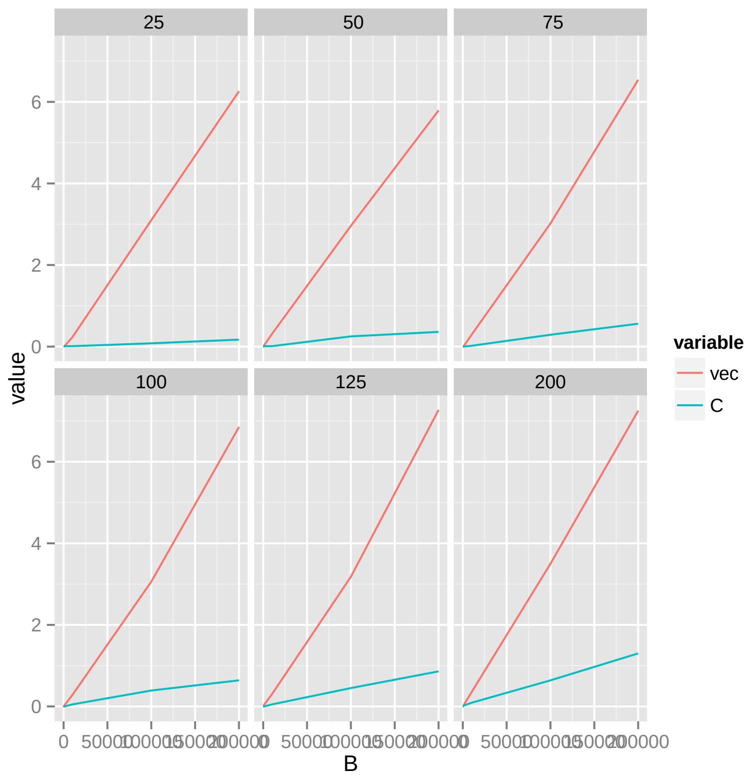

Figure 2: Performance Study, R and C