Statistics Programming

An Introduction to C++, Rcpp and Armadillo

References (1)

- The idea of this tutorial is to give you a simple, basic introduction to C++, Rcpp and Armadillo.
- Many more information on Rcpp, Armadillo and C++ can be found online at:
- Rcpp: `http://dirk.eddelbuettel.com/presentations.html` (Dirk gave a presentation on Rcpp last year at UCLA)
- Armadillo: `http://arma.sourceforge.net/` (great source of info on the C++ linear algebra library Armadillo)

References (2)

- Armadillo and Rcpp:
  `http://q-aps.princeton.edu/sites/default/files/q-aps/files/slides_day4_am.pdf`
  (great introduction to Rcpp Armadillo. Armadillo examples shown herein were obtained from this presentation)
- C++: `http://www.math.ucla.edu/~wittman/10a.1.10w/`(Prof. Todd Whittman used to teach C++ at UCLA. Lecture notes from an intro to C++ course he taught in 2010 are available online at his website. C++ notes in this presentation were based on these)
- This tutorial relies on material found in the abovementioned websites. Any errors are our own.

Getting Started (1)

Why Rcpp?

- Easy to learn and use.
- Speed gains for a variety of tasks. Rcpp does well exactly where R is deficient: loops, function calls.
- Getting started is easy. In RStudio, simply click on "File", "New File", "C++ File".
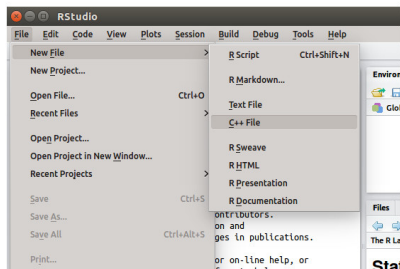
Getting Started (2)

The following cpp file gets created by RStudio:

```cpp
1  #include <Rcpp.h>
2  using namespace Rcpp;
3
4  // Below is a simple example of exporting a C++ function to R. You can
5  // source this function into an R session using the Rcpp::sourceCpp
6  // function (or via the Source button on the editor toolbar)
7
8  // For more on using Rcpp click the Help button on the editor toolbar
9
10 // [[Rcpp::export]]
11 int timesTwo(int x) {
12     return x * 2;
13 }
14
```

Code Structure (1)

This file already uses C++ language. Let's examine it closely. Your cpp files will have a structure similar to this one.

- The first line (`#include <Rcpp.h>`) tells the computer to include the C++ library Rcpp.h
- Libraries are called header files in C++ and end in .h.
- The second line (using namespace Rcpp;) tells the computer to use the Rcpp namespace. So if the computer comes across "unknown commands" throughout the code, it knows to look them up in the included library.
- Notice that this line ends with a semi-colon. Differently from R, every statement in C++ ends with a semi-colon, but not necessarily every line.

Code Structure (2)

- Next, we have a bunch of comments. Loosely speaking, you can insert comments in your cpp file by writing "//" before them.
- Important: // [[Rcpp::export]] – This is not a comment! When we actually source our cpp file (we'll see how this works in a second), the sourceCpp function parses a C++ file and looks for functions marked with the Rcpp::export. It then exports these functions to your R environment, so you can use call them in your R code.
- Thus, you should have this command (// [[Rcpp::export]]) before all functions you want to "export" to R.

Code Structure (3)

- Next, we have a function, written in C++.

```
int timesTwo(int x) {
return x * 2;
}
```

The (int x) part indicates that an integer, whose name is x, is passed to the function. (More on this below when we discuss functions.) The int in the beginning means that the program is also expecting an integer when we are done. The function will return the value of x multiplied by 2. The { } enclose the code.

Using your C++ functions in R (1)

- You can now write your C++ functions into this file, or another cpp file you create. Those functions marked with the // [[Rcpp::export]] attribute will be made available as R functions once your cpp file is sourced. Those not marked with this attribute will not.

- To source your cpp file, either click on the "Source File" button in RStudio, or, in your R code, type:

```
library(Rcpp)
sourceCpp('File_Name.cpp').
```

Where File_Name is the name you have chosen for your cpp file.

Using your C++ functions in R (2)

- Make sure your cpp file was saved in your working directory. You can find out what your working directory is by going in RStudio to "Tools", "Global Options".

- EXAMPLE: Assume you saved the abovementioned file (the one with the TimesTwo function) in your R working directory. You can source this file and use the TimesTwo function as follows:

```
library(Rcpp)
sourceCpp('timesTwo.cpp')
timesTwo(30)
## [1] 60
```

evalCpp() and cppFunction()

I recommend you write all your functions in a cpp file, and then use sourceCpp to export them to R. However, you should also know that the following commands can be used directly in R:

- evalCpp() evaluates a single C++ expression.
- cppFunction() creates, compiles and links a C++ file.

C++ basics - Data Types

Before proceeding, let's take a brief look at the C++ language. This may turn out helpful when you are writing your own code.

Common C++ data types are:

- bool (Boolean, i.e., true or false). N. Bits: 1
- char (Character, i.e., letter). N. Bits: 8
- short (Short integer). N. Bits: 16
- int (Integer). N. Bits: 32
- float (Floating point number). N. Bits: 32
- double (Long float). N. Bits: 64
- long double (Very long float). N. Bits: 80

C++ basics - Classes

- Note: A class is a suped-up data type that may have special member functions defined in a library.
- The most common Rcpp classes are the "NumericVector" and "NumericMatrix" classes.
- The most common Armadillo classes are "mat" (matrices)and "vec" (vectors). Armadillo is the most popupar C++ linear algebra library. More on it later.
- A common C++ class is the "String Class". You can think of it as a vector of characters.

Rcout (1)

- The Rcpp.h header file comes packed with a number of functions. Rcout is one of the most popular functions found in it.
- The Rcpp Rcout statement writes whatever follows to the console (cout = console out). In C++, we have to specifically tell the computer to print stuff out on the screen, otherwise nothing will be printed. This can be accomplished through the usage of this function.

Rcout (2)

- EXAMPLES

```
Rcout << "Hello Students!";
# will print "Hello Students!" on the screen.
Rcout << "Hello Students!" << std::endl;
Rcout << "I love Statistics";
# will print "Hello Students!", jump to the next line,
and then print "I love Statistics" on the screen.
The std::endl does not appear on the screen.
Note that we need this to tell the computer to
jump to the next line. This is equivalent to "\n" in C++.

int x = 10;
Rcout << x;
# prints the value of x, 10, on the screen.
```

Caution (1)!!

Be Careful!

- Strings and other objects are generally indexed starting from zero (like in Python).
- Be careful what values you write into a variable, depending on its type. int x = 11.8; The integer x will be stored as 11!
- Avoid mixing integers and doubles, i.e., subtracting doubles from integers, or vice-versa.
- All variables in your C++ code need to be declared in your C++ code. What I mean by that is that you have to specify their types. This goes for counters as well. REMEMBER to declare every single variable you use.

Caution (2)!!

- Some Rcpp functions can behave oddly. For example, the abs() function in Rcpp supposedly returns the absolute value of its argument. However if you pass a double to it, it will return an integer to you!
- EXAMPLE: double x = 11.8; abs(x); will return 11!.
- If you pass a NumericVector to it, it will however return the right value. NumericVector x = 11.8; abs(x); will return 11.8.
- If you have problems with your code, make sure the "built-in" functions you are using in your cpp file are giving you what you want.

Functions (1)

- When declaring a function, we need to identify the data types of the parameters that are passed to the function, and the type of variable that our function returns.

```
type function_name ( type param1 , type param2 , ... ) {
**FUNCTION CODE**
}
```

Functions (2)

EXAMPLE:

```
int multiply2Integers ( int num1, int num2 ) {
int product = num1*num2;
return product;
}
```

- If a function doesn't return a value, you should give it the blank data type void. And end the function with simply return;

Functions (3)

You can have your Rcpp function return multiple variables by using the Rcpp data type "List".

```cpp
// Build and return a list
int a = 3;
double b = 4;

List ret; ret["one"] = a; ret["two"] = b;
return(ret);
```

When called in R, a function containing these statements will return a list with two elements. The first element will be named "one", storing the integer 3. The second element of that list will be named "two", storing the number 4 as a floating point number.

For Loops

The general form of a for loop in C++ is:

```
for (initalization; stopping condition; update) {
**STATEMENTS**
}

// Or, in other words:
for (int i = start; i <= end; i++) {
**STATEMENTS**
}
```

- NOTE: we need semi-colons in between the 3 pieces.
- We'll see examples of for loops further down.

C++ : If Statements

- As in R, an if statement will execute the statements that follow only if the boolean condition in parentheses is TRUE.
- C++ Syntax:

```
if (statement is TRUE) {
**STATEMENTS**
}
```

EXAMPLE

```
int x = 2;
if (x < 5) {
Rcout << "hello";
}
```

- Outputs "hello".
- Braces: without the braces, only the first line following the if statement will be executed.

C++ : If/Else Statements

- One can chain if/else statements.
- EXAMPLE

```
int x = 5;

if (x>0)
Rcout << "X is positive.";
else if (x<0)
Rcout << "X is negative.";
else
Rcout << "X is zero.";
```

C++ : Ternary Operator

- Special Syntax: You can also use the so-called "ternary operator", ?:, to implement an if-else statement.
- EXAMPLE: the function below is equivalent to the sign() function in R. It reads: "if d is less than zero, then, return -1; else if d is greater than 0, return 1; else return 0. If-else statements can be written in a compact way using the ternary operator.

```
double sign_cpp(double d){
    return d<0?-1:d>0? 1:0;
}
```

EXAMPLE : the function below returns the greatest value amongst two doubles.

```
double max_cpp(double a, double b){
 return a>b?a:b;
 }
```

C++ : Multiple If/Else Statements

- You can also write multiple If-Else Statements. The example below illustrates this:

```
void print_score(int gre_quant_score) {
if ( gre_quant_score >= 165 ) {
Rcout << "You may have a shot at UCLA";
}
else if (gre_quant_score >= 150 )
Rcout << "You should consider retaking the exam";
else if (gre_quant_score >= 130 )
Rcout << "How many drinks did you have before taking the test?";
else {
Rcout << "Surely your score reflects some computer glitch."
 << std::endl;
Rcout << "Forget about graduate school?";
}
return;
}
}
```

C++ math library

- The C++ math library is the C math library.
- To use it: `# include <cmath.h>`
- Raising x to power of n: pow(x,n)

Accessing Classes and Functions

You can access classes and functions from C++ namespaces using "::".
EXAMPLE: std::abs(), arma:: mat

```
double abs3(double x) {
  return std::abs(x);
  }
```

Basic Syntax: Rcpp (1)

- Source for this slide and the next one: Rcpp Quick Reference Guide - http://dirk.eddelbuettel.com/code/rcpp/Rcpp-quickref.pdf
- NumericVector

```cpp
// of a given size (filled with 0)
NumericVector xx(10);

NumericVector xx = NumericVector::create(
1.0, 2.0, 3.0, 4.0 );

//find size
xx.size()
```

Basic Syntax: Rcpp (2)

- NumericMatrix

```
// Matrix of 4 rows & 5 columns (filled with 0)
NumericMatrix xx(4, 5);

// Fill with value
int xsize = xx.nrow() * xx.ncol();
for (int i = 0; i < xsize; i++) {
xx[i] = 7;
}
// Same as above, using STL fill
std::fill(xx.begin(), xx.end(), 8);
// Assign this value to single element
// (1st row, 2nd col)
xx(0,1) = 4;

// Copy the second column into new object
NumericVector zz1 = xx( _, 1);
// Copy the submatrix (top left 3x3) into new object
NumericMatrix zz2 = xx( Range(0,2),
Range(0,2));

//find number of rows or cols
xx.nrow();
xx.ncol();
```

Rcpp: Basic Issues

- Say B is a NumericMatrix: B(Range(i,K), j) does not work. You can use B(Range(i,K),Range(j, j)) instead. Note that this will return a NumericMatrix, not a NumericVector.
- Typically you should avoid assigning a "one-dimensional" numeric matrix to a numeric vector. This will likely result in an error. Work with either matrices or vectors and avoid mixing them up.

Pointers!

The NumericMatrix and NumericVector classes actually represent numeric pointers.
The discussion below can be found at StackOverflow.
http://stackoverflow.com/questions/28664894/rcpp-copy-a-vector-to-a-vector

```cpp
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
void testRun() {
    IntegerVector Grp1 = IntegerVector::create(1,2,3,4);
    IntegerVector Grp2 = IntegerVector::create(3,4,5,6);

    Rf_PrintValue(Grp1);
    Grp1 = Grp2;
    Rf_PrintValue(Grp1);
    Grp2[3] = Grp2[3]+1;
    Rf_PrintValue(Grp1);
}
```

"and when I run testrun(), I get the output"

```
> Rcpp::sourceCpp('test.cpp')
> testRun()
[1] 1 2 3 4
[1] 3 4 5 6
[1] 3 4 5 7
```

Clone

Solution:

- Grp1 = clone(Grp2);
- This will create a new R object that then get assigned to Grp1.
- Also – good practice: declare stuff you read in from R and want to protect as const. We'll see an example below.

Sample Code: mySweep (R)

```r
mySweep <- function(A, m)
{
  n <- dim(A)[1]

  for (k in 1:m)
  {
    for (i in 1:n)
      for (j in 1:n)
        if (i!=k  & j!=k)
          A[i,j] <- A[i,j] - A[i,k]*A[k,j]/A[k,k]

    for (i in 1:n)
      if (i!=k)
        A[i,k] <- A[i,k]/A[k,k]

    for (j in 1:n)
      if (j!=k)
        A[k,j] <- A[k,j]/A[k,k]

    A[k,k] <- - 1/A[k,k]
  }
  return(A)
}
```

Sample Code: mySweep

```cpp
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericMatrix mySweep(const NumericMatrix B, int m)
{

  NumericMatrix A = clone(B);
  int n = A.nrow();

  for (int k = 0; k < m;k++){
    for (int i = 0; i < n;i++){
      for (int j = 0; j < n;j++){
        if ((i!=k) & (j!=k))
        A(i,j) = A(i,j) - A(i,k)*A(k,j)/A(k,k);
      }
    }

    for (int i = 0; i < n;i++){
      if (i!=k)
      A(i,k) = A(i,k)/A(k,k); }

      for (int j = 0; j < n;j++){

        if (j!=k)
        A(k,j) = A(k,j)/A(k,k); }

        A(k,k) = - 1/A(k,k);
  }

  return A;
}
```
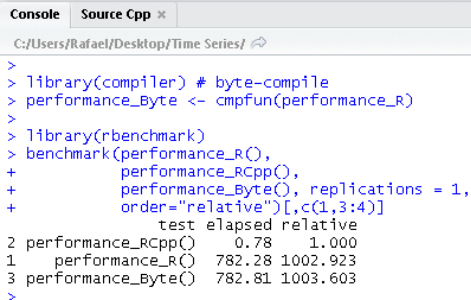
Performance Comparison ()

n = 100 p = 500

```
Console    Source Cpp ×                                                   ▬ ☐
C:/Users/Rafael/Desktop/Time Series/ ☞
>
> library(compiler) # byte-compile
> performance_Byte <- cmpfun(performance_R)
>
> library(rbenchmark)
> benchmark(performance_R(),
+           performance_RCpp(),
+           performance_Byte(), replications = 1,
+           order="relative")[,c(1,3:4)]
                test elapsed relative
2 performance_RCpp()    0.78    1.000
1    performance_R()  782.28 1002.923
3 performance_Byte()  782.81 1003.603
>
```

Why Armadillo (1)?

- The data structures (C++ types and Rcpp classes) we mentioned do not provide us with the tools to avoid element-wise implementations of linear algebra operations (e.g., matrix multiplication).
- We can use the C++ Linear Algebra library armadillo do deal with linear algebra operations (matrix multiplication, element-wise multiplications, transpose, etc).

Why Armadillo (2)?

- Armadillo Linear algebra classes (vec and mat) allow code to look like standard mathematical notation.
- Primary classes of interest mat and vec (elements are double).

Armadillo - Basic Example

- EXAMPLE: Computing the inner product between two vectors.

```
// [[ Rcpp :: export ()]]
double inner2 (vec x, vec y) {
mat inner_prod = x.t() * y ;
return(inner_prod(0)) ;
}
```

The function above receives two vectors of same length, x and y. Vectors are seen as "column vectors" in Armadillo. Then, it computes the transpose of x (a row vector) and multiplies that by y.

How to use Armadillo

- Previously:

```
# include <Rcpp.h>
using namespace Rcpp;
```

Now:

```
# include <RcppArmadillo.h>
// [[ Rcpp :: depends ( RcppArmadillo )]]
using namespace Rcpp; using namespace Arma;
# We don't need <Rcpp.h> any longer.
```

Previously, in your R code: library(Rcpp) Now, in your R code:
library(RcppArmadillo)

Other Armadillo Examples

- Other Examples:

```
// [[ Rcpp :: export ()]]
mat aa (mat x) {
return(x % x) ;
}
Conducts element−wise multiplication.

// [[ Rcpp :: export ()]]
mat aa(mat x) {
return( exp(x) ) ;
}
//Element−wise applications of functions
//(in this case the exp() function)
```

Going from R to Rcpp Armadillo

Table 1: R to Armadillo

R Code	Armadillo	Note
A[1,1]	A(0,0)	Indexing in Armadillo starts at 0.
A[k,k]	A(k-1,k-1)	Indexing in Armadillo starts at 0.
dim(A)[1]	A.n_rows	
dim(A)[2]	A.n_cols	
A[,k]	A.col(k)	This is a conceptual example only;
A[k,]	A.row(k)	Exact conversion from R to Armadillo requires
A[,p:q]	A.cols(p,q)	taking into account that indexing starts at 0.
A[p:q,]	A.rows(p,q)	
A[p:r,q:s]	A.submat(p,r,q,s)	
t(A)	A.t()	
matrix(0,nrow = k, ncol = k)	A = zeros<mat>(k,k)	
matrix(1,nrow = k, ncol = k)	A = ones<mat>(k,k)	
A*B	A%B	Element-wise multiplication
A/B	A/B	Element-wise division
solve(A,B)	solve(A,B)	
A+1	A++	
A-1	A–	
matrix(c(1,3,2,4),nrow = 2)	A << 1 << 2 << endr <<3 <<4 endr;	element initialisation
rbind(A,B)	X = join_rows(A,B)	
cbind(A,B)	X = join_cols(A,B)	
A	Rcout << A	

Putting it all together (1)

How would you implement the R code below using instead both R and Rcpp/Armadillo?
Sample R code (Pre-Rcpp)

```
n = 100
p = 200
s = 10
T = 1000
epsilon = .1

X = matrix(rnorm(n*p), nrow=n)
beta_true = matrix(rep(0, p), nrow = p)
beta_true[1:s] = 1:s
Y = X %*% beta_true + rnorm(n)

beta = matrix(rep(0, p), nrow = p)
db = matrix(rep(0, p), nrow = p)
beta_all = matrix(rep(0, p*T), nrow = p)

R = Y
ss = rep(0, p)
for (j in 1:p)
    ss[j] = sum(X[, j]^2)

for (t in 1:T)
    {
        for (j in 1:p)
          db[j] = sum(R*X[, j])/ss[j]
        j = which.max(abs(db))
        beta[j] = beta[j]+db[j]*epsilon
        R = R - X[, j]*db[j]*epsilon
        beta_all[, t] = beta
}

matplot(T:1, t(beta_all), type = 'l')
```

Putting it all together (2)

Writing the loops above in C++ (with Rcpp and Armadillo):

```cpp
// [[Rcpp::depends(RcppArmadillo)]]

#include <RcppArmadillo.h>

using namespace Rcpp; using namespace arma;

// [[Rcpp::export]]
mat beta_stage_cpp_arma(int T, int p, double epsilon,
vec ss, vec R,
vec beta, NumericVector db, mat X, mat beta_all){

int c = 0;

for(int j =0; j<p; j++) {
  ss(j) = sum(X.col(j)%X.col(j));
}

for(int t = 0;t<T;t++){

  for(int j = 0;j<p;j++){

  db(j) = sum(R%X.col(j))/ss(j);}

  c = which_max(abs(db));
  beta(c) = beta(c) +db(c)*epsilon;
  R = R - X.col(c)*db(c)*epsilon;
  beta_all.col(t) = beta;          }

  return beta_all;
}
```

Putting it all together (3)

And our final R code will be:

```
library(RcppArmadillo)
sourceCpp("beta_stage_cpp_arma.cpp")

n = 100
p = 200
s = 10
T = 1000
epsilon = .1

X = matrix(rnorm(n*p), nrow=n)
beta_true = matrix(rep(0, p), nrow = p)
beta_true[1:s] = 1:s
Y = X %*% beta_true + rnorm(n)

beta = matrix(rep(0, p), nrow = p)
db = matrix(rep(0, p), nrow = p)
beta_all = matrix(rep(0, p*T), nrow = p)

R = Y
ss = rep(0, p)

beta_all = beta_stage_cpp_arma(T, p, epsilon, ss,
 R, beta, db, X, beta_all)

matplot(T:1, t(beta_all), type = 'l')
```