High Performance Functions With Rcpp

(make R coding faster with the integration of C++)

About

This presentation is created by Farheen Nilofer (https://in.linkedin.com/pub/farheen-nilofer/a5/5b9/58a) as part of the internship requirements at DecisionStats.org.
I would like to thank Mr. Ajay Ohri (https://in.linkedin.com/in/ajayohri) and Miss Sunakshi for their invaluable help and guidance.

Get Code here:
https://github.com/Farheen2302/RCPP_Code

Introduction

The usage and implementation of Rcpp package of R have been detailed in these slides.

Convert a R function to a Rcpp function which is the integration of both C++ and R to make R coding many folds faster than the usual speed of execution.

Every functionality is explained through many examples to make understand how the integration is done.

Prerequisites:

Languages:

1.Basics of C++ or C(both will work ,you can go for it http://www.learncpp.com/ or http://www.cplusplus.com/)

2. Basics of R (You can join 4 hours of R course at DataCamp)

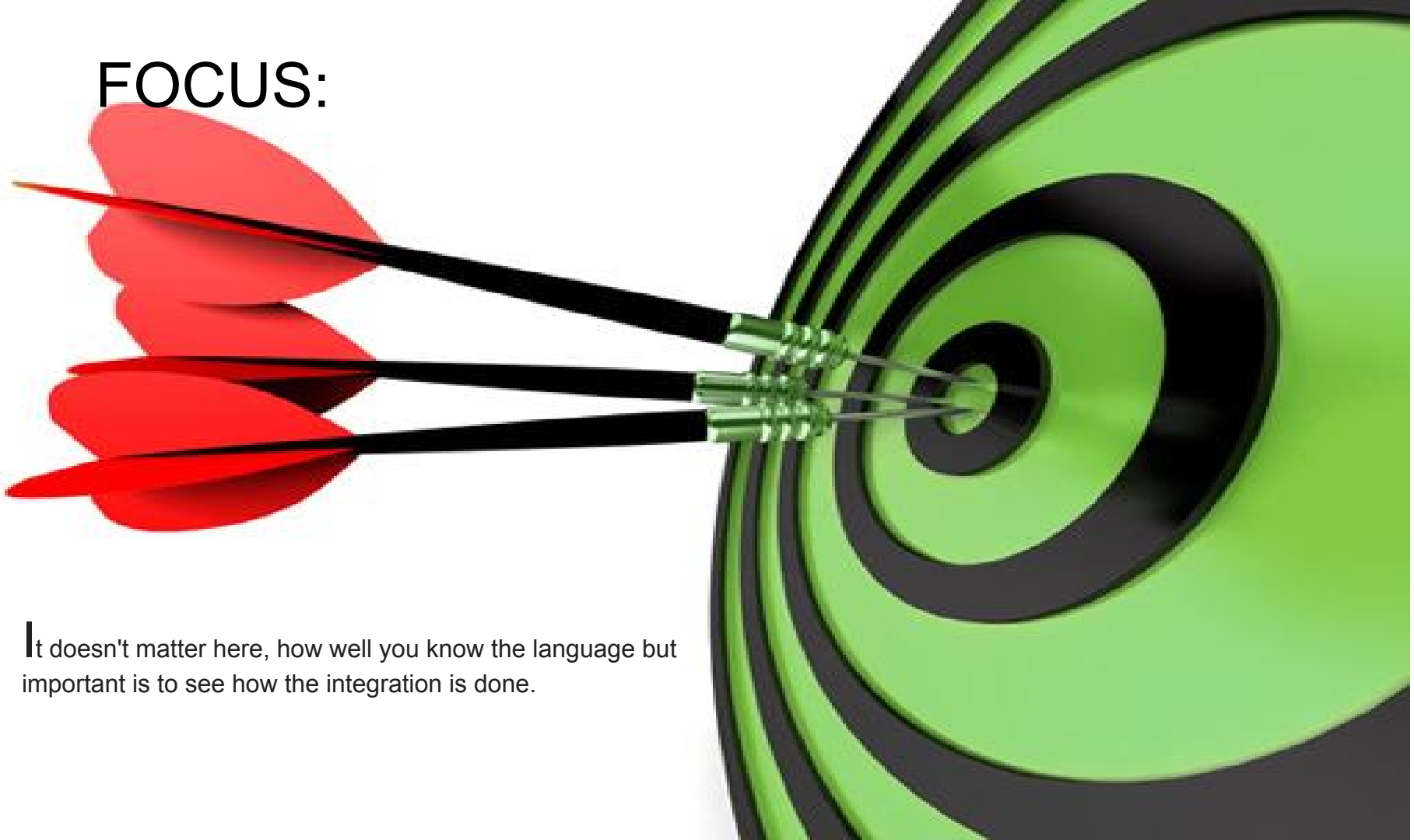
Tools:

R or better RStudio

You'll also need a working C++ compiler. To get it:

- On Windows, install Rtools.
- On Mac, install Xcode from the app store.
- On Linux, `sudo apt-get install r-base-dev` or similar.

FOCUS:



It doesn't matter here, how well you know the language but important is to see how the integration is done.

What is Rcpp?

Rcpp is a package in R.

-tool written by Dirk E and Romain Francois

Rcpp makes it very easy to integrate your R code with C++.

Rcpp provides easy ,clean and approachable API that will ultimately give you high performance code

link - http://www.rcpp.org/

Why we need Rcpp?

-code is just not fast enough.

-to improve the performance in R.

- Rcpp package help in integrating C++ with R and that help in making the code faster.

sources- http://adv-r.had.co.nz/Rcpp.html

Bottlenecks of R handled by C++

-problems of Data Structures and Algorithms the R doesn't support .

C++ has STL(Standard Template Library) to implement efficiently many data structures.

- Recursive functions or calling a function a million times are hardest time for R code to execute .

- overhead of a function in C++ is much lower than R.

-C++ makes it easy for loops to vectorise code easily whose subsequent iteration depend on previous ones.

What will be covered through these slides?

Using sourceCpp: sourceCpp() is a function of R to load files from the disk. sourceCpp() is Rcpp function -an extension of R source() to load C++ files in R.

Attributes and Classes: Attributes and classes to be used with Rcpp.

Missing Values: deal with R's missing values through C++.

Rcpp Sugar: avoid loops in C++ and write vectorized code (what is syntactic sugar- explain more)

The STL(Standard Template Library): use data structures and algorithms from STL,built in C++.

Examples: implemented examples to enhance the performance.

Putting Rcpp package:How to put C++ in R package.

Learning More: Many references to further learning is provided here.

1.Getting started with C++

When you run the below code,Rcpp will compile the C++ code and construct a R function that connect to the compiled C++ function.

What is compile?

Compile is a process to convert a high level language to a machine code that computer's processor uses.

How to run?

```
> library("Rcpp")
> cppFunction('int add(int a, int b, int c)
{
int sum = a + b +c;return sum;
}')
> add(4,5,6)
[1] 15
```

How to code in Rcpp?

This is the example to show how to go through the process of conversion.

R Code:

```
Rcode:
> oneR <-function()1L
> oneR()
[1] 1
```

C++ code:

Syntax of C++ code:
http://www.cplusplus.com/doc/tutorial/functions/

```
int oneR()
{
    return 1;
}
```

How to code in Rcpp?

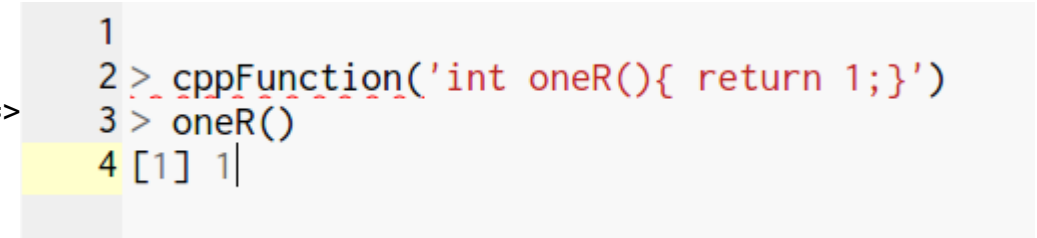
-include the whole compiled C++ program inside the parenthesis of **cppFunction()** within quotes ‘ ’ in the R console and hit enter.

-Nothing will happen on no error.

-Now call the function name as you do for R function call.

==========>

C++ implementation.



How C++ functions different from R

Observe the above R and C++ code:

- Syntax to C++ looks alike of R -no assignment operator C++
- declare the type of output to be returned by the C++ function.In the above code the function returned ‘int’.
- The classes for types of R vectors are NumericVectors, IntegerVectors, CharacterVectors and LogicalVectors
- Scalars and Vectors are different.The scalar equivalent of numeric,integer and character and logical vectors are double,int String and bool.
- use explicit return statement to return a value from the function.
- statement in C++ is terminated by ‘;’.

2.Convert R functions to C++ equivalent.

There are four different ways:

- Scalar input and scalar output
- Vector input and scalar output
- Vector input and vector output
- Matrix input and vector output

Scalar Input and Scalar Output

R code:

This simple R code and look at its C++ implementation in the next slide.

Rcpp1.cpp | *Rcpp2.cpp | Rcpp3.cpp

```
> signR <-function(x){
+     if(x>1){
+         1
+     }else if(x==0){
+         0
+     }else {
+         -1
+     }
+ }
```

Scalar Input and Scalar Output(cont..)

C++ code:

-C++ representation of above R code.

```
1 int signR(int x){
2     if(x>0)
3       return 1;
4     else if(x==0)
5       return 0;
6     else
7       return -1;
8   }
9
```

Scalar Input and Scalar Output(cont..)

If statement of C++ is same as

If statement of R

```
> cppFunction('int signR(int x){
+     if(x>0)
+       return 1;
+     else if(x==0)
+       return 0;
+     else
+       return -1;
+   }')
> signR(0)
[1] 0
```

Vector Input and Scalar Output(cont..)

R code:

```
> sum<- function(x){total <- 0
+     for(i in seq_along(x)){
+         total <-total + x[i]
+     }
+     total
+ }
> sum(x)
[1] 15
```

Vector Input and Scalar Output

The cost of loops in C++ is very less

Rcpp:

```
> cppFunction('double sum(NumericVector x){
+   double total = 0;
+   int i=0;
+   int len = x.size();
+   for(i=0;i<len;i++)
+   {
+     total = total + x[i];
+   }
+   return total;
+ }')
> x <- c(1,2,3,4,5)
> sum(x)
[1] 15
```

Some more difference between C++ and R.

C++ version is similar but bit different as in

1.C++ methods are called with a full stop as here done for calling 'size()'

2.'for' loop has a different syntax

3.In C++ vector indices start at '0' ,this is very common source of bug while converting R function to C++

4.Use ' = ' instead '<-'.

‘microbenchmark’ to compare the speed

install.packages(“microbenchmark”)

‘microbenchmark’ function from a package XC in R used to check the execution speed of our programs.

import ‘microbenchmark’ function through the command:

library(microbenchmark)

system.time() and Rbenchmark() also do the same.

```
> x <- runif(1e3)
> microbenchmark(
+     sum(x),
+     sumC(x),
+     sumR(x)
+ )
```

-minimum time taken by a C++ program is 4.272 and for the max value ,the least time 27.209

```
Unit: microseconds
    expr     min       lq     mean   median       uq      max neval cld
  sum(x) 630.703 668.2985 749.5731 699.4740 758.6200 2394.989   100   b
sumC(x)   4.272   5.0755   6.7247   6.3455   7.1195   27.209   100  a
sumR(x) 628.073 662.5360 715.4215 689.3375 763.9115  878.244   100   b
```

Vector Input Vector Output

R code:

```r
pdistR <- function(x, ys){
  sqrt((x - ys) ^ 2)
}
```

Rcpp Code:

```
C Code:
cppFunction('NumericVector pdistC(double x, NumericVector ys)
+ {
+  int n = ys.size();
+  NumericVector out(n);
+  for(int i = 0; i< n ; i++) {
+    out[i]=sqrt(pow(x - ys[i],2.0));
+  }
+  return out;
+ }')
> pdistC(x,ys)
[1] 1 2 3
```

Vector Input Vector Output(cont..)

We create a new numeric vector of length n with a constructor: NumericVector out(n). Another useful way of making a vector is to copy an existing one:

NumericVector zs = clone(ys).

In C++ there is pow() to calculate power.

R code take 8 ms to execute and your C++ code takes 4ms but took 10mins time to write a C++.

What do you think is it worth it??

Yes ! When it comes calling same function a million times .

Matrix Input Vector output

Matrix could have equivalents like

NumericVector, IntegerVector,

CharacterVector, LogicalVector

Rcpp Code:==========>

```
> library("Rcpp")
> #Matrix input and Vector Output
> cppFunction('NumericVector rowsumC(NumericMatrix x) {
+             int row = x.nrow();
+             int col = x.ncol();
+             int i=0,j=0,total=0;
+             NumericVector out(col);
+             for(i = 0; i <row ; i++) {
+               total =0;
+               for(j = 0 ; j < col ; j++) {
+                 total = total + x(i, j);
+               }
+             out[i] = total;
+               }
+         return out;}')
>
```

DESCRIPTION:

There are two methods nrow and ncol for getting number of rows and number of columns.

```
1
2 > mat = matrix(c(1,2,3,4,5,6,7,8,9), nrow=3,ncol=3)
3 > rowsumC(mat)
4 [1] 12 15 18
5 > mat
6      [,1] [,2] [,3]
7 [1,]    1    4    7
8 [2,]    2    5    8
9 [3,]    3    6    9
10 >
```

From Inline to Stand-alone

Its tiresome to write code inside the function cppFunction().

Are you too?

Can’t we have some other method to do so?

Using sourceCpp()

The method we have earlier used was the inline method.

Sometimes when you need some sort of code immediately because in real word everything is so fast the we need to keep everything ready.

Using sourceCpp() (cont..)

There comes Stand alone functions.In this we need to already define the function and use it whenever required without the overhead of writing immediatly.

How to do that?

Ans-You need to add only two three lines of code to your C++ function and compile the function whenever needed using sourceCpp() function from your R console.

Using sourceCpp() (cont..)

Make sure to save your Rcpp file as .cpp extension

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

And keep in mind that there is a ‘gap’ between ‘//’ and ‘[[Rcpp::export]]’

```
// [[Rcpp::export]]
```

Using sourceCpp() (cont..)

Write Rcpp code in different file.

Rcpp1.cpp

```
1  #include <Rcpp.h>
2  using namespace Rcpp;
3  // [[Rcpp::export]]
4  double mean(NumericVector x)
5  {
6          int n = x.size();
7          double total = 0.0;
8          for(int i = 0; i < n ; i++)
9          {
10             total += x[i];
11         }
12     return total/n;
13 }
```

How to compile standalone function

To compile your code,use function ‘sourceCpp()’ from R console ,as shown below.

```
>
>
> sourceCpp('/home/farheen/Desktop/Rcpp_16_6/Rcpp1.cpp')
> x<-c(1,2,3,4,5,6,7)
> mean(x)
[1] 4
>
```

Note : You can see in the above image that C++ code is much faster than R’s in built mean function.

```
> microbenchmark(mean(x),meanC(x))
Unit: microseconds
     expr     min      lq     mean  median       uq     max neval cld
  mean(x) 341.788 342.275 343.4937 342.759 343.4055 367.799   100   b
 meanC(x) 170.391 170.828 172.0789 171.764 172.2225 191.828   100  a
>
```

Example 1:

How to pass function as argument?

==========>>

Rcpp1.cpp | Rcpp2.cpp

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

// [[Rcpp::export]]
int func(Function pred,NumericVector x)
{
    int n = x.size();
    for(int i = 0 ; i < n ; i++)
    {
        NumericVector res = pred(x[i]);
        if(res[0])
            return i + 1;
        }
    return 0;
}

```

```
> sourceCpp('/home/farheen/Desktop/Rcpp_16_6/Rcpp2.cpp')
> x <-c(0,0,1,1,0)
> func(sign,x)
[1] 3
```

Example 2:

Rcpp1.cpp | *Rcpp2.cpp | Rcpp3.cpp

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

// [[Rcpp::export]]
NumericVector mini(NumericVector x, NumericVector y)
{
        int n = std::max(x.size(),y.size());
        NumericVector x1 = rep_len(x, n);
        NumericVector y1 = rep_len(y, n);
        NumericVector out(n);
        for(int i = 0 ; i < n ; i++)
        {
                out[i] = std::min(x1[i], y1[i]);
            }
        return out;
    }
```

```
> sourceCpp('/home/farheen/Desktop/Rcpp_16_6/Rcpp3.cpp')
> x1 <- c(1,2,3,4,5,5)
> y1 <- c(1,24,3,1,3,4,5,6)
> mini(x1,y1)
[1] 1 2 3 1 3 4 1 2
>
```

More practice……

Someone here who want more brushing can also try ..

all(),cumprod(),cummin(),cummax(),diff(),range,var and for prior knowledge visit wikipedia.

3.Attributes and other classes

NumericVector,IntegerVector,LogicalVector,CharcterVector are Vector classes.

Scalar classes like int,double,bool,String and same for matrices like IntegerMatrix,NumericMatrix,LogicalMatrix and CharacterMatrix.

As we know all R objects have attributes.And these attributes can be set ,removed or modified using a ‘attr()’

attr(x,”dim”)<- c(2,5)

```
> attr(x,"dim")<-c(2,5)
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

We will see ‘::class()’ which allows

us to create R vector using C++ scalar values.

Attributes and other classes(cont...)

Now see how we used ::create() to create a R vector from C++ scalar

```cpp
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
NumericVector attrbs()
{
    NumericVector out = NumericVector::create(1,2,3);
    out.names()=CharacterVector::create('a','b','c');
    out.attr("my-attr")="My_attributes";
    out.attr("class")="my-class";

    return out;
}

```

Attributes and other classes(cont...)

Output.:compiled stand alone function.

Checkout attributes,

```
sourceCpp('/home/farheen/Desktop/Rcpp_16_6/Rcpp4.cpp')
>
> attrbs()
abc\x9e*\004  bc\x9e*\004   c\x9e*\004
           1            2            3
attr(,"my-attr")
[1] "My_attributes"
attr(,"class")
[1] "my-class"
```

List and DataFrames(as(),lm(),inherit(),stop())

List and DataFrames one of the most important features in R.

-will see how to extract component from the list using as() and convert them into C++ equivalents.

Note lm(), inherit() and stop().

List and DataFrames(cont..)

Look at as(),

how it converts

R objects'

component to C++

```cpp
#include <Rcpp.h>

using namespace Rcpp;
// [[Rcpp::export]]
double mpe(List mod)
{
    if(!mod.inherits("lm"))
        stop("Only linear model accepted");

    NumericVector resid = as<NumericVector>(mod["residuals"]);
    NumericVector fitted =as<NumericVector>(mod["fitted.values"]);
    int n = resid.size();
    double err = 0;
    for(int i = 0 ; i < n ; i++)
    {
        err += resid[i]/(resid[i]+fitted[i]);
    }
    return err/n;
}
```

List and DataFrames(cont..)

Check out the output ,the errrrrr!

```
> sourceCpp('/home/farheen/Desktop/Rcpp_16_6/Rcpp5.cpp')
>
> mod <- lm(mpg ~ wt,data = mtcars)
> mpg(mod)
Error: could not find function "mpg"
> mpe(mod)
[1] -0.01541615

```

Why R function in Rcpp code?

Calling a R function could be very useful.

1.For parameter initialization.

2.Access a custom data summary.To recode Huh! Overhead!

3.Calling a plotting routine.Tedious help,right?

Calling a R function would be overhead too.

Its slower than C++ equivalent and even slower than R code.

Warning: Do it when it make sense,not because its available.

Why R function in Rcpp code?

Here is the simple Rcpp code to demonstrate how to call R function from C++ code.

```
1 #include <Rcpp.h>
2 using namespace Rcpp;
3 // [[Rcpp::export]]
4
5 NumericVector CallRfunc(NumericVector x,Function f)
6 {
7   NumericVector res = f(x);
8   return res;
9 }
```

Why R function in Rcpp code?(cont..)

Look at the output ,it is similar to R function output below.

```
> x <- rnorm(1e5)
> CallRfunc(x,func)
[1] 1.077731
[1] 0.6233562
[1] 0.06276959
[1] 1.491707
[1] -0.1173361
```

Why R function in Rcpp code?(cont..)

This is the R function which is called by 'CallRfunc()' above

```
1 func <-function(x) {
2 +     i <- 0
3 +     y <- 5
4 +     for(i in seq(y)) {
5 +         print (x[i])
6 +     }
7 + }
8 >
9 > func(x)
10 [1] 0.6233562
11 [1] 0.06276959
12 [1] 1.491707
13 [1] -0.1173361
14 [1] -0.6419487
15 >
```

Why R function in Rcpp code?(cont..)

There are classes for many more specialised language object.

Environment, ComplexVector, RawVector, DottPair, Language, Promise, Symbol, WeakReference and so on.

4.Missing Values

How to deal with missing values in Rcpp?

Lets find out!

We need to get two things.

1.How R's missing values behaves in C++ Scalar(int ,double..).

2.How to get and set Missing values in Vector(NumericVector,..).

Scalars

R's missing values are first coerced in C++ and then back to R vector.

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

// [[Rcpp::export]]

List scal_mis()
{
    int int_s=NA_INTEGER;
    String chr_s=NA_STRING;
    bool lgl_s = NA_LOGICAL;
    double num_s = NA_REAL;

    return List::create(int_s,chr_s,lgl_s,num_s);
}
```

Scalars(cont..)

With the exception of bool,things are pretty good here.

```
> sourceCpp('/home/farheen/Desktop/Rcpp_16_6/Rcpp7.cpp')
> scal_mis()
[[1]]
[1] NA

[[2]]
[1] NA

[[3]]
[1] TRUE

[[4]]
[1] NA
```

Integers

With Integers ,we store missing values as smallest values

As C++ don’t know the this special behaviour ,playing with it gives an incorrect value.

evalCpp {Rcpp}

Evaluates a C++ expression. This creates a C++ function using cppFunction and calls it to get the resul

```
> evalCpp('NA_INTEGER + 1')
[1] -2147483647
>
```

Doubles

With doubles , ignore the missing values and work with R's NaN(Not A NUMBER)

It is R's NA is special type of NaA.It has characterstic that if it is involved in logical expression it gives FALSE.

With LOGICAL EXPRESSION=>

```
> evalCpp("NAN == 1")
[1] FALSE
> evalCpp("NAN < 1")
[1] FALSE
> evalCpp("NAN > 1")
[1] FALSE
> evalCpp("NAN == NAN")
[1] FALSE
>
```

Doubles(cont...)

With Boolean Values

Here NAN acts as a TRUE value

```
> evalCpp("NAN && TRUE")
[1] TRUE
> evalCpp("NAN && FALSE")
[1] FALSE
> evalCpp("NAN || TRUE")
[1] TRUE
> evalCpp("NAN || FALSE")
[1] TRUE
>
```

Doubles(cont...)

In context with Numeric Values NA are propagated.

Look at the output …

all NANs.

```
>
> evalCpp("NAN + 1")
[1] NaN
> evalCpp("NAN - 1")
[1] NaN
> evalCpp("NAN / 1")
[1] NaN
> evalCpp("NAN * 1")
[1] NaN
>
```

String

String is Scalar string ,itself introduced by Rcpp.

String knows how to deal with missing values.

Boolean

One thing to note here is that C++ have only two value TRUE and FALSE while R's logical Vector have three values TRUE, FALSE and NA.

VECTOR

-Missing values specific to the type of Vector like

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

// [[Rcpp::export]]
List missing_values() {
  return List::create(
    NumericVector::create(NA_REAL),
    IntegerVector::create(NA_INTEGER),
    LogicalVector::create(NA_LOGICAL),
    CharacterVector::create(NA_STRING));
}
```

VECTOR(cont...)

missing values.

```
> sourceCpp("/home/farheen/Desktop/Rcpp_16_6/Rcpp9.cpp")
> missing_values()
[[1]]
[1] NA

[[2]]
[1] NA

[[3]]
[1] NA

[[4]]
[1] NA

```

Each element in the list is of specific type.

```
> str(missing_values())
List of 4
 $ : num NA
 $ : int NA
 $ : logi NA
 $ : chr NA
```

VECTOR(cont...)

In this function you can see each item in the vector need to be check .

```cpp
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
LogicalVector is_naC(NumericVector x)
{
   int n = x.size();
   LogicalVector out(n);

   for(int i = 0 ; i < n ; i++)
    {
        out[i] = NumericVector::is_na(x[i]);
    }

    return out;
}
```

VECTOR(cont...)

Each element is specified if it is NA or not through logical Values.

```
1 > sourceCpp('/home/farheen/Desktop/Rcpp_16_6/Rcpp10.cpp')
2 > is_naC(x)
3 [1] FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE
4 >
```

5.Rcpp Sugar

Rcpp provide lot of Syntactic Sugar to ensure C++ function work very similar to their R equivalent.

Sugar functions can be roughly broken down into

- logical summary functions
- arithmetic and logical operators
- vector views
- other useful functions]

Lets begin

Rcpp Sugar(cont...)

ARITHMETIC AND LOGICAL OPERATORS:

All basic arithmetic and logical operators are vectorised as: `+` ,`*`, `-`, `/`, `pow`, `<`, `<=`, `>`, `>=`, `==`, `!=`, `!`.

Use sugar as following:

R function implementing pdistR

Now we could sugar to considerably simplify the code

```
1 pdistR <- function(x, ys) {
2     sqrt((x - ys) ^ 2)
3     }
4
5 >x <- c(1,23,4,5)
6 > ys <- 3
7 > pdistR(x, ys)
8 [1]  2 20  1  2
```

Rcpp Sugar(cont...)

Sugar implementation in C++

Without sugar we need to use loop

```
1 #include <Rcpp.h>
2 using namespace Rcpp;
3
4 // [[Rcpp::export]]
5 NumericVector pdistC(NumericVector x, double ys)
6 {
7     return sqrt(pow((x - ys),2));
8 }
```

Rcpp Sugar(cont...)

LOGICAL SUMMARY FUNCTION

Use sugar to write an efficient function .

Sugar function like any(),all() , is.na() are very efficient .

This will do the same amount of work regardless of the position of the missing values.

Proved in below microbench program

```
any_naR <- function(x) any(is.na(x))

> any_naR(x)
[1] FALSE
> x
[1]  1 23  4  5
>
```

Rcpp Sugar(cont...)

Sugar implementation in C++

Below microbench() shows the execution of both R and C++ code

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

// [[Rcpp::export]]
bool any_naC(NumericVector x) {
  return is_true(any(is_na(x)));
}

> sourceCpp('/home/farheen/Desktop/Rcpp_16_6/Rcpp14.cpp')
> any_naC(x)
[1] FALSE
> x
[1] 3
> x <- c(1,2,3,4,5,6,NA,34)
> any_naC(x)
[1] TRUE
>
```

Rcpp Sugar(cont...)

Changing the position of missing values.

```
> any_r untitled-1 function(x) any(is.na(x))
> x0 <- runif(1e5)
> x1 <- c(x0, NA)
> x2 <- c(NA, x0)
>
> microbenchmark(
+     any_naR(x0), any_naC(x0),
+     any_naR(x1), any_naC(x1),
+     any_naR(x2), any_naC(x2)
+ )
```

Rcpp Sugar(cont...)

-execution time of each program have hardly any effect on the execution time of R code due to missing value place.

```
Unit: microseconds
         expr      min        lq      mean   median        uq       max neval cld
 any_naR(x0)  895.445 1013.2000 1054.2477 1046.034 1051.2110  2911.458   100  b
 any_naC(x0) 2008.266 2011.9745 2019.3772 2016.564 2022.3300  2142.938   100   c
 any_naR(x1)  895.095 1036.7700 1128.7003 1046.463 1053.0965  2926.561   100  b
 any_naC(x1) 2007.986 2011.4295 2018.4665 2016.497 2023.3060  2058.282   100   c
 any_naR(x2)  560.791  671.4505  818.6814  714.271  719.8905  2745.007   100 ab
 any_naC(x2)    3.034    3.9560  369.9380    5.326    6.9315 36387.571   100 a
>
```

Rcpp Sugar(cont...)

VECTOR VIEW

There are many functions which Sugar provide to view the vector.

head(), tail(), rep_each(), rep_len(), rev(), seq_along(), and seq_len().

In R these would all produce copies of the vector, but in Rcpp they simply point to the existing vector which makes them efficient.

There's grab bag of sugar functions.

- Math functions: abs(), acos(), asin(), atan(), beta(), ceil(), ceiling(), choose(), cos(), cosh(), digamma(),exp(), expm1(), factorial(), floor(), gamma(), lbeta(), lchoose(), lfactorial(), lgamma(), log(), log10(),log1p(), pentagamma(), psigamma(), round(), signif(), sin(), sinh(), sqrt(), tan(), tanh(), tetragamma(),trigamma(), trunc().
- Scalar summaries: mean(), min(), max(), sum(), sd(), and (for vectors) var().
- Vector summaries: cumsum(), diff(), pmin(), and pmax().
- Finding values: match(), self_match(), which_max(), which_min().
- Dealing with duplicates: duplicated(), unique().
- d/q/p/r for all standard distributions.

Finally, noNA(x) asserts that the vector x does not contain any missing values.

6.The STL(Standard Library Templates)

The real strength of C++ is when you use STL for algorithm and Data Structure.

If you need an algorithm or data structure that isn't implemented in STL, a good place to look is boost.

Iterators are used heavily in STL .Many function either accept or return iterators.

They are the next step up from basic loops, abstracting away the details of the underlying data structure

The STL(cont...)

Using ITERATORS

Iterators are used heavily in STL . Many function either accept or return iterators.

They are the next step up from basic loops, abstracting away the details of the underlying data structure

```
1  #include <Rcpp.h>
2  using namespace Rcpp;
3  // [[Rcpp::export]]
4  double sum4(NumericVector x)
5  {
6      double total = 0.0;
7      NumericVector::iterator it;
8      for(it = x.begin() ;it != x.end();++it)
9      {
10         if(NumericVector::is_na(*it) == FALSE)
11           {
12             total += *it;
13           }
14     }
15
16     return total;
17 }
```

The STL(cont...)

```
> sourceCpp('/home/farheen/Desktop/Rcpp_16_6/Rcpp16.cpp')
> sum4(x)
[1] 55
> x
[1]  1  2  3  4  5  6 NA 34
>
```

The STL(cont...)

Iterators also allow us to use the C++ equivalents of the apply family of functions. For example, we could again rewrite sum() to use the accumulate() function.

Note: Third argument to accumulate gives the initial value determines the data type. (here 'double')

```
1
2 #include <Rcpp.h>
3
4 using namespace Rcpp;
5 // [[Rcpp::export]]
6 double sum4(NumericVector x)
7 {
8   return std::accumulate(x.begin(), x.end(), 0.0);
9 }
10
11
12 > sourceCpp('/home/farheen/Desktop/Rcpp_16_6/Rcpp17.cpp')
13 > sum4(x)
14 [1] 18
```

The STL(cont...)

ALGORITHMS

The <algorithm> header provides large number of algorithms that work with iterators. A good reference is available at http://www.cplusplus.com/reference/algorithm/.

Here we are implementing basic Rcpp version of findInterval()

Where two vector are inputs and

output is there upper bound.

The STL(cont...)

```cpp
#include <Rcpp.h>

using namespace Rcpp;
// [[Rcpp::export]]

IntegerVector findInterval2(NumericVector a, NumericVector y)
{
    IntegerVector out(a.size());
    NumericVector::iterator it, pos;
    IntegerVector::iterator out_it;

    for(it = a.begin(), out_it = out.begin(); it != a.end(); ++it,++out_it)
    {
        pos = std::upper_bound(y.begin(),y.end(),*it);
        *out_it = std::distance(y.begin(),pos);
    }
    return out;
}
```

The STL(cont...)

Look at the output.

Upper bound position of the items in x are searched in the vector y.

It's generally better to use algorithms from the STL than hand rolled loops.

STL algorithms are efficient,well tested and these Standard algorithm makes the code readable and more maintainable.

```
> x <- c(34,56,78)
> findInterval2(x,y)
[1] 4 5 6
> x <- c(34,21,78)
> findInterval2(x,y)
[1] 4 3 6
> y
[1]  4  5  6 34 56 67
> x
[1] 34 21 78
> x <- c(21,34,78)
> findInterval2(x,y)
[1] 3 4 6
```

The STL(cont...)

Data Structures

The STL provides a large set of data structures: array, bitset, list, forward_list, map, multimap, multiset,priority_queue, queue, dequeue, set, stack, unordered_map, unordered_set, unordered_multimap,unordered_multiset, and vector.

The most important of these data structures are the vector, the unordered_set, and the unordered_map.

A good reference for STL data structures is http://www.cplusplus.com/reference/stl/ — I recommend you keep it open while working with the STL.

Rcpp knows how to convert from many STL data structures to their R equivalents

The STL(cont...):Vectors

Data Structures

An STL vector is very similar to an R vector but more efficient.

Vectors are templated i.e you need to specify their type while creating.vector<int>, vector<bool>, vector<double>, vector<String>.

standard [] notation : to access element in the vector

.push_back(): to add new element at the end of the vector.

.reserve() : to allocate sufficient storage.

More methods of a vector are described at http://www.cplusplus.com/reference/vector/vector/.

The STL(cont...) Data Structures

Look at the two vector declared and there type.

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

// [[Rcpp::export]]
List rleC(NumericVector x) {
  std::vector<int> lengths;
  std::vector<double> values;

  // Initialise first value
  int i = 0;
  double prev = x[0];
  values.push_back(prev);
  lengths.push_back(1);

  NumericVector::iterator it;
  for(it = x.begin() + 1; it != x.end(); ++it) {
    if (prev == *it) {
      lengths[i]++;
    } else {
      values.push_back(*it);
      lengths.push_back(1);

      i++;
      prev = *it;
    }
  }

  return List::create(
    _["lengths"] = lengths,
    _["values"] = values
  );
}
```

The STL(cont...):Sets

Set maintain a unique set of values.

C++ provides both ordered (std :: set) and unordered sets (std :: unordered_set).

Unordered set are efficient than ordered sets.

Like vectors sets are also templated ,you need to specify the type. `unordered_set<int>, unordered_set<bool>, etc.`

For more details visit http://www.cplusplus.com/reference/set/set/ and http://www.cplusplus.com/reference/unordered_set/unordered_set/.

Note :The use of `seen.insert(x[i]).second`. `insert()` returns a pair, the `.first` value is an iterator that points to element and the`.second` value is a boolean that's true if the value was a new addition to the set.

The STL(cont...):Sets

Note that the ordered sets are only available in C++ 11.

Therefore you need to use cpp11 plugig.

// [Rcpp::plugins(cpp11)]]

```cpp
// [[Rcpp::plugins(cpp11)]]
#include <Rcpp.h>
#include <unordered_set>
using namespace Rcpp;

// [[Rcpp::export]]
LogicalVector duplicatedC(IntegerVector x) {
  std::unordered_set<int> seen;
  int n = x.size();
  LogicalVector out(n);

  for (int i = 0; i < n; ++i) {
    out[i] = !seen.insert(x[i]).second;
  }

  return out;
}
```

The STL(cont...):Map

It is useful for functions like table() or match() that need to look up the value ,instead of storing presence and absence it can store the additional data.

There are ordered (std::map) and unordered (std::unordered_map).

Map is templated ,you need to specify the type like map<int, double> ,unordered_map<double, int>.

The STL(cont...):Sets

Unordered maps are only available in C++ 11

```cpp
#include <Rcpp.h>

using namespace Rcpp;
// [[Rcpp::export]]
std::map<double, int> tableC(NumericVector x)
{
    std::map<double, int> counts;
    int n = x.size();

    for(int i = 0 ; i < n; i++)
    {
        counts[x[i]]++;
    }
    return counts;
}
```

The STL(cont...):Sets

output table:

```
> sourceCpp('/home/farheen/Desktop/Rcpp_16_6/Rcpp21.cpp')
> tableC(x)
1 2 3 4 5 6 7
3 2 3 3 2 4 1
> x
 [1] 1 1 1 2 2 3 3 3 4 4 4 5 5 6 6 6 6 7
>
> y
[1]  4  5  6 34 56 67
> tableC(y)
 4  5  6 34 56 67
 1  1  1  1  1  1
```

7.Case Studies

Two case-studies to give you a reason to replace your R code with Rcpp.

1.Gibbs Sampler

2.R vectorisation vs C++ vectorisation

GIBBS SAMPLER:

The R and C++ code shown below is very similar (it only took a few minutes to convert the R version to the C++ version), but runs about 20 times faster on my computer.

Case Studies(cont...):GIBBS SAMPLER

R CODE:

```r
gibbs_r <- function(N, thin) {
  mat <- matrix(nrow = N, ncol = 2)
  x <- y <- 0

  for (i in 1:N) {
    for (j in 1:thin) {
      x <- rgamma(1, 3, y * y + 4)
      y <- rnorm(1, 1 / (x + 1), 1 / sqrt(2 * (x + 1)))
    }
    mat[i, ] <- c(x, y)
  }
  mat
}
```

```
> gibbs_r(5,5)
          [,1]       [,2]
[1,] 0.4385219 -0.2793711
[2,] 0.2961570  1.4575477
[3,] 1.3773151  0.2997856
[4,] 0.9273027  0.1845081
[5,] 0.2673503  1.2651604
```

Case Studies(cont...):GIBBS SAMPLER

RCPP CODE:

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

// [[Rcpp::export]]
NumericMatrix gibbs_cpp(int N, int thin)
{
    NumericMatrix mat(N,2);
    double x = 0,y=0;
    for(int i = 0;i <N;i++)
    {
        for(int j = 0;j<thin;j++)
        {
            x = rgamma(1,3,1/(y * y + 4))[0];
            y = rnorm(1,1/(x+1),1 / sqrt(2 * (x + 1)))[0];
        }
        mat(i,0) = x;
        mat(i, 1)= y;
    }
    return(mat);
}
```

```
> gibbs_cpp(5,5)
          [,1]       [,2]
[1,] 1.1301489  0.1162286
[2,] 1.4336608 -0.0830889
[3,] 1.1651273  0.3415322
[4,] 0.5975518  0.1269640
[5,] 1.1521344  0.2927905
```

Case Studies(cont...):GIBBS SAMPLER

Rcpp program is 20 times faster than the R code.

Check this-

min: 298/15 ~20

max:702.240/40.028 ~17.54~``20

```
> library("microbenchmark", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.2")
> microbenchmark(gibbs_r(5,5),gibbs_cpp(5,5))
Unit: microseconds
           expr     min      lq      mean  median       uq     max neval cld
   gibbs_r(5, 5) 298.761 340.136 366.61687 365.643 377.5000 702.240   100   b
 gibbs_cpp(5, 5)  15.252  18.705  21.85202  20.429  22.8265  40.028   100  a
> 
```

Case Studies: R VECTORIZATION VS C++ VECTORIZATION

-adapted from “Rcpp is smoking fast for agent-based models in data frames”.

-predict the model response from three different inputs.

R code:

```r
vaccR1 <- function(age, female, ily) {
  p <- 0.25 + 0.3 * 1 / (1 - exp(0.04 * age)) + 0.1 * ily
  p <- p * if (female) 1.25 else 0.75
  p <- max(0, p)
  p <- min(1, p)
  p
}

vaccR2 <- function(age, female, ily) {
  n <- length(age)
  out <- numeric(n)
  for (i in seq_len(n)) {
    out[i] <- vaccR1(age[i], female[i], ily[i])
  }
  out
}
```

Case Studies:R VECTORIZATION VS C++ VECTORIZATION(cont...)

R Code:(no loop in it)

```
vaccR <- function(age,female,ily) {
  p <- 0.25 + 0.3* 1/(1 - exp(0.04 * age)) +0.1*ily
  p <- p* ifelse(female,1.25,0.75)
  p<- pmax(0, p)
  p <= pmin(0, p)
  p
}
```

Case Studies: R VECTORIZATION VS C++ VECTORIZATION(cont...)

Rcpp Code:

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

double vaccCpp(double age,bool female, bool ily)
{
    double p = .25 + .3 * 1/(1 - exp(0.04 * age)) + 0.1 * ily;
    p = p * (female ? 1.25 : 0.75);
    p = std::max(p,0.0);
    p = std::min(p,1.0);
    return p;
}


// [[Rcpp::export]]
 NumericVector vaccRcpp(NumericVector age,LogicalVector female,LogicalVector ily) {
     int n = age.size();
     NumericVector out(n);
    for(int i = 0; i < n; i++)
    {
        out[i] = vaccCpp(age[i],female[i],ily[i]);
    }
    return out;

}
```

Case Studies: R VECTORIZATION VS C++ VECTORIZATION(cont...)

Vectorising in R gives a huge speedup.

```
> stopifnot(
+     all.equal(vaccR2(age, female, ily), vaccR(age, female, ily)),
+     all.equal(vaccR2(age, female, ily), vaccRcpp(age, female, ily))
+ )
>
```

create 11 vectors

Performance (~10x) with the C++ loop

create only 1 vector.

```
> microbenchmark(
+     vacc1 = vaccR2(age, female, ily),
+     vacc2 = vaccR(age, female, ily),
+     vacc3 = vaccRcpp(age, female, ily)
+ )
Unit: microseconds
  expr      min       lq       mean    median        uq       max neval cld
 vacc1 7185.416 7381.747 9321.43999 7803.5265 8481.8995 21924.882   100   b
 vacc2  448.927  461.346  496.60289  490.2125  514.2535   722.571   100  a
 vacc3   57.047   58.542   67.37255   63.7660   69.1335   110.152   100  a
>
```

8.USING RCPP IN PACKAGE

C++ code can also be bundles in packages instead of using sourceCpp().

BENEFITS:

- user can use C++ code without development tools
- R package build system automatically handle multiple source file and their dependencies.
- provide additional infrastructure

To include Rcpp to the existing package you put your C++ files in the `src/` directory and modify/create the following configuration files:

- In `DESCRIPTION` add:

```
LinkingTo: Rcpp
Imports: Rcpp
```

- Make sure your `NAMESPACE` includes:

```
useDynLib(mypackage)
importFrom(Rcpp, sourceCpp)
```

USING RCPP IN PACKAGE(cont...)

- To generate a new Rcpp package that includes a simple “hello world” function you can use Rcpp.package.skeleton():

```
Rcpp.package.skeleton("NewPackage", attributes = TRUE)
```

- To generate a package based on C++ files that you’ve been using with sourceCpp(), use the cpp_files parameter:

```
Rcpp.package.skeleton("NewPackage", example_code = FALSE,
                      cpp_files = c("convolve.cpp"))
```

Before building package you need to run Rcpp::compileAttributes()

- scans the C++ files forRcpp::export attributes
- generates the code required to make the functions available in R

Re-run Rcpp::compileAttributes() whenever function is added.

For more details see vignette("Rcpp-package")

LEARNING MORE

Rcpp book

vignette("Rcpp-package")

vignette("Rcpp-modules")

vignette("Rcpp-quickref")

Rcpp homepage

Dirk's Rcpp page

Learning C++:

Effective C++ and *Effective STL* by Scott Meyers

C++ Annotations

Algorithm Libraries

Algorithm Design Manual

Introduction to Algorithms

online textbook

coursera course

Thank You

You can get the whole code :https://github.com/Farheen2302/RCPP_Code

Any questions?

Contact us at

info@decisionstats.org

farheenfnilofer@gmail.com