Using C++ with R for statistical analysis

Assen Tchorbadjieff

Institute of Mathematics and Informatics - BAS

January 28, 2017

Introduction

Rccp

Conclusions

- C - Ken Thompson and Dennis Ritchie 1974
- "C with Classes" - Bjarne Stroustrup 1979
- C++ New features are introduced 1983
  - virtual funct's, function name and operator overloading, etc
- New features are introduced 1989
  - multiple inheritance, abstract classes, static member functions, const member functions, and protected members
- Later STL, MFC, C++ 2011, ...

- OOP
  - abstraction
  - encapsulation
  - inheritance
  - polymorphism
- Compiling to machine code

- S-language, Rick Becker and Allan Wilks, John Chambers (later) of Bell Laboratories.
- First version 1976
- Directly calling Fortran subroutines for statistical computing
- 1979 run on UNIX system
- Now it is split in two versions of S-lnaguage:
  - R: GNU free software project
  - S: PLUS, a commercial product

What R offers us:

- Compiled code in C/Fortran
- Enormous number of very well verified statistical packages with high performance
- Interpreted code with user friendly interface
- An interactive workow for data analysis

WHY to Merge C/C++ with R?

- R is written mainly in C/Fortran. Remarkable improvements in performace are not expected.
- But sometimes loops,function calls, etc. might be improved
- You could develop own R packages.
- You could deploy R statistical functionalities in your C/C++ project.
- Implement core OpenMP functionality in Analysis of Large Data.
- Cluster/Supercomputer computations.

- R CRAN (https://www.r-project.org/)
  Basic installation:
  - Windows: Rtools
  - Mac: Xcode from app store
  - Linux: sudo apt-get install r-base-dev
- C++ compilator
- Eclipse; inline in RStudio, not in MSVS
- Rcpp package - install.packages("Rccp")
- Environment config (example)
  Sys.setenv("*PKG_CXXFLAGS*" = "-std=c++11")

- Library load: dyn.load("libname.so")
- Language dependent calls:
  - .C("*func_name*", params ...) - pure C code. Obsolete and Limited!
  - .Fortran("*func_name*", params ...) - Fortran code. Not good practice!
  - .Call("*func_name*", params ...) - C/C++ code

R uses C data structures:

- Everything is SEXP - a pointer or structure to where it points (SEXPREC).
- SEXPRECs or *VECTOR_SEXPRECs* (R nodes)-C structures with 32-bit header, atributes, data ...
- 32 SEXPTYPEs:
  - NILSXP(NULL ptr)
  - REALSXP (numeric vectors)
  - STRSXP (character vectors)
- Each atomic vector has its special constant for NULL:
  - INTSXP:*NA_INTEGER*
  - STRSXP:*NA_STRING*

Casts:

- Cast functions from/to SEXP:
  - from: 'as()'
  - to: 'wrap()'
  example:
  vector$< double > v = Rcpp :: as < vector < double >> (x)$;
- Wrap pointer as an external:
  $Rcpp :: Xptr < type > ptr =$ ptr(new ...);
- RCpp *sugar* functions:
  - Rcpp::NumericVector(); Rcpp::IntegerVector(); Rcpp::LogicalVector(); etc.
  - Binary arithmetic operators
- R type apply functionality

C++ class Exposing

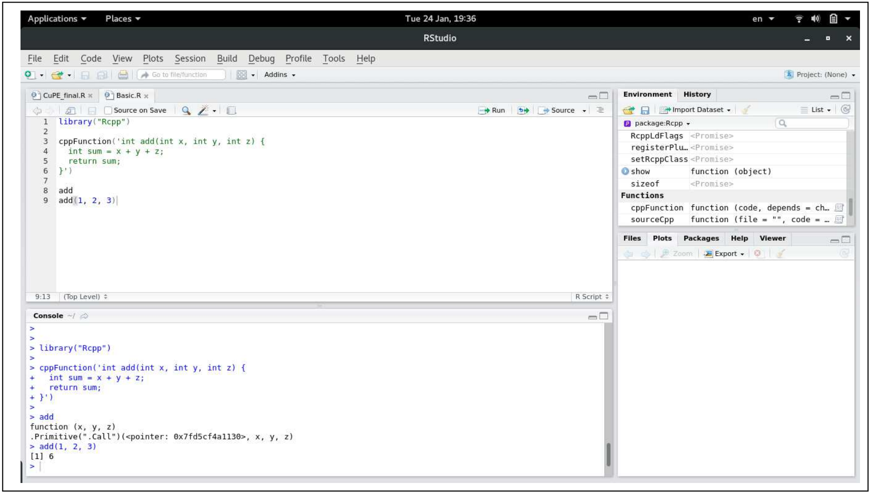
- $RCPP\_MODULE(met\_name)$
  $\{function(''name'', \&func_ref, ...)\}$
- $RCPP\_MODULE(module\_name)\{$
  $Defs : ..\}$
- Definitions:
  $class\_ < Type > (''Name'');$
  $.constructor < types, ... > ()$
  $.field(''name'', \&Type :: field\_ref)$ - variable
  $.field\_readonly(''name'', \&Type :: field\_ref)$ - restricted variable
  $.method(''name'', \&Type :: method\_ref, ..)$ method
  $.property(''name'', \&Type :: method\_ref, ..)$ get/set-ers
- Example:
  $.property(''z'', \&Foo :: get\_z, \&Foo :: set\_z)$

Attributes

High level syntax for declaring C++ functions callable in R and automatically generate code required for invoke them:

- Rccp::export - to export C++ function to R
- sourceCPP - to source exported function from file
- cppFunction and evalCpp - inline declaration and execution
- Rccp::depends - to specify additional build dependencies for sourceCPP

Example 1. Direct use in RStudio



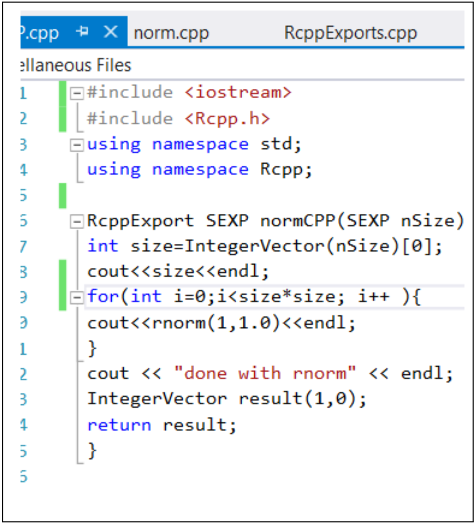
```
library("Rcpp")

cppFunction('int add(int x, int y, int z) {
  int sum = x + y + z;
  return sum;
}')

add
add(1, 2, 3)
```

RcppLdFlags	<Promise>
registerPlu…	<Promise>
setRcppClass	<Promise>
show	function (object)
sizeof	<Promise>
Functions	
cppFunction	function (code, depends = ch…
sourceCpp	function (file = "", code = …

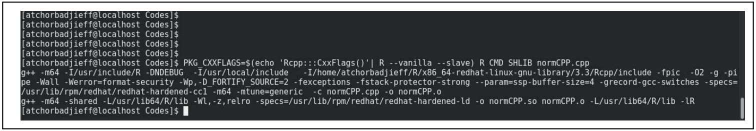
```
>
>
> library("Rcpp")
>
> cppFunction('int add(int x, int y, int z) {
+   int sum = x + y + z;
+   return sum;
+ }')
>
> add
function (x, y, z)
.Primitive(".Call")(<pointer: 0x7fd5cf4a1130>, x, y, z)
> add(1, 2, 3)
[1] 6
>
```

Example 2. C-style function

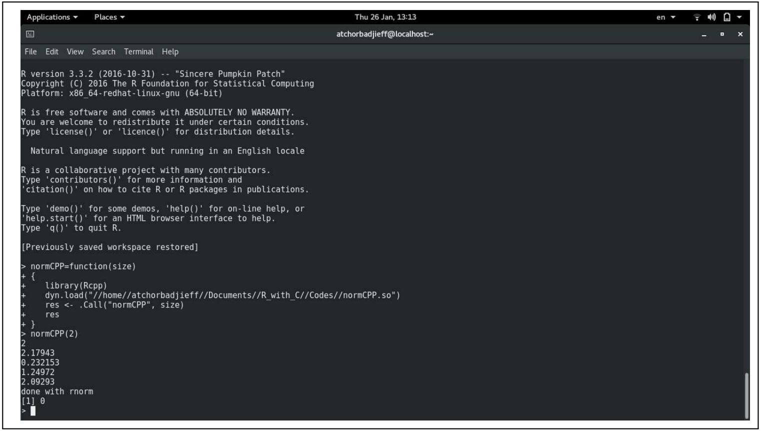
- Generate a matrix[nxn] with randomly generated N(1,1) distributed values
- C++ Function normCPP in file normCPP.cpp
- g++ -m64



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

RcppExport SEXP normCPP(SEXP nSize)
int size=IntegerVector(nSize)[0];
cout<<size<<endl;
for(int i=0;i<size*size; i++ ){
cout<<rnorm(1,1.0)<<endl;
}
cout << "done with rnorm" << endl;
IntegerVector result(1,0);
return result;
}
```



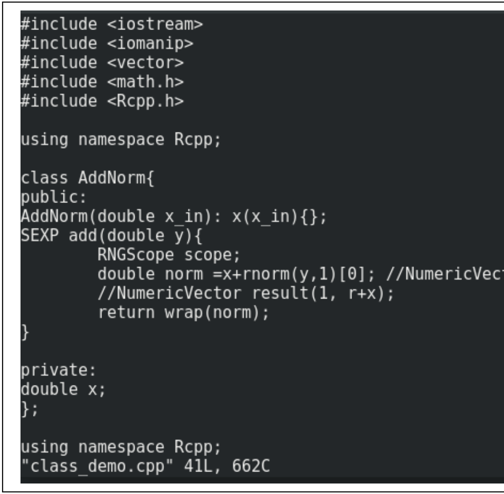
```
[atchorbadjieff@localhost Codes]$
[atchorbadjieff@localhost Codes]$
[atchorbadjieff@localhost Codes]$
[atchorbadjieff@localhost Codes]$
[atchorbadjieff@localhost Codes]$
[atchorbadjieff@localhost Codes]$ PKG_CXXFLAGS=$(echo 'Rcpp:::CxxFlags()'| R --vanilla --slave) R CMD SHLIB normCPP.cpp
g++ -m64 -I/usr/include/R -DNDEBUG  -I/usr/local/include   -I/home/atchorbadjieff/R/x86_64-redhat-linux-gnu-library/3.3/Rcpp/include -fpic  -O2 -g -pi
pe -Wall -Werror=format-security -Wp,-D_FORTIFY_SOURCE=2 -fexceptions -fstack-protector-strong --param=ssp-buffer-size=4 -grecord-gcc-switches -specs=
/usr/lib/rpm/redhat/redhat-hardened-cc1 -m64 -mtune=generic  -c normCPP.cpp -o normCPP.o
g++ -m64 -shared -L/usr/lib64/R/lib -Wl,-z,relro -specs=/usr/lib/rpm/redhat/redhat-hardened-ld -o normCPP.so normCPP.o -L/usr/lib64/R/lib -lR
[atchorbadjieff@localhost Codes]$
```



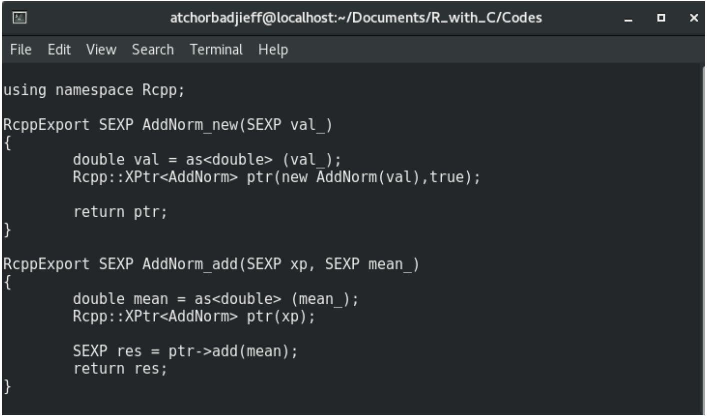
```
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-redhat-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> normCPP=function(size)
+ {
+    library(Rcpp)
+    dyn.load("//home//atchorbadjieff//Documents//R_with_C//Codes//normCPP.so")
+    res <- .Call("normCPP", size)
+    res
+ }
> normCPP(2)
2
2.17943
0.232153
1.24972
2.09293
done with rnorm
[1] 0
>
```

Example 3. Class definition

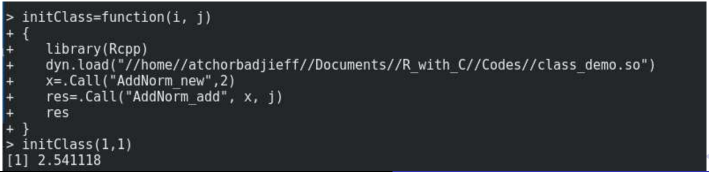
- OOP class definition
- C++ class AddNorm
- class function SEXP Add(double y);
- Class is initiated in C-class function



```
#include <iostream>
#include <iomanip>
#include <vector>
#include <math.h>
#include <Rcpp.h>

using namespace Rcpp;

class AddNorm{
public:
AddNorm(double x_in): x(x_in){};
SEXP add(double y){
        RNGScope scope;
        double norm =x+rnorm(y,1)[0]; //NumericVec
        //NumericVector result(1, r+x);
        return wrap(norm);
}

private:
double x;
};

using namespace Rcpp;
"class_demo.cpp" 41L, 662C
```


atchorbadjieff@localhost:~/Documents/R_with_C/Codes
File Edit View Search Terminal Help

```
using namespace Rcpp;

RcppExport SEXP AddNorm_new(SEXP val_)
{
        double val = as<double> (val_);
        Rcpp::XPtr<AddNorm> ptr(new AddNorm(val),true);

        return ptr;
}

RcppExport SEXP AddNorm_add(SEXP xp, SEXP mean_)
{
        double mean = as<double> (mean_);
        Rcpp::XPtr<AddNorm> ptr(xp);

        SEXP res = ptr->add(mean);
        return res;
}
```



```
> initClass=function(i, j)
+ {
+    library(Rcpp)
+    dyn.load("//home//atchorbadjieff//Documents//R_with_C//Codes//class_demo.so")
+    x=.Call("AddNorm_new",2)
+    res=.Call("AddNorm_add", x, j)
+    res
+ }
> initClass(1,1)
[1] 2.541118
```

Example 4. Using OpenMP

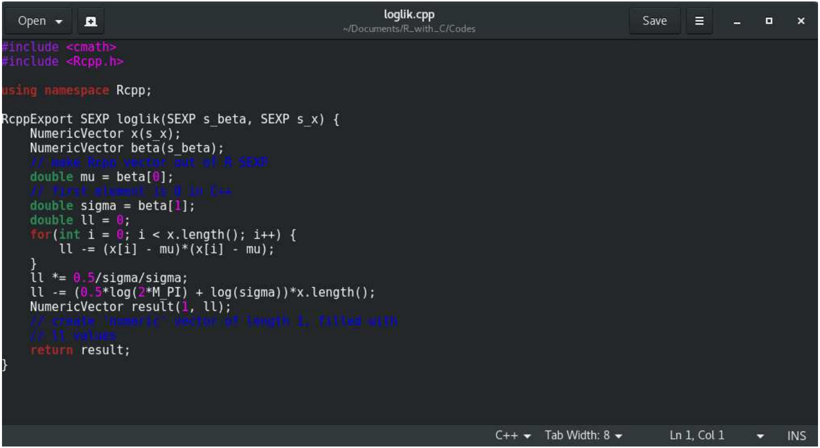
- Sample of random normals
- log-likelihood function

$$lnL = -\frac{1}{2}nln(2\pi) - nln\sigma - \frac{\sum(x_i - \mu)^2}{2\sigma^2} \tag{1}$$

- Optimize lnL
- R version:

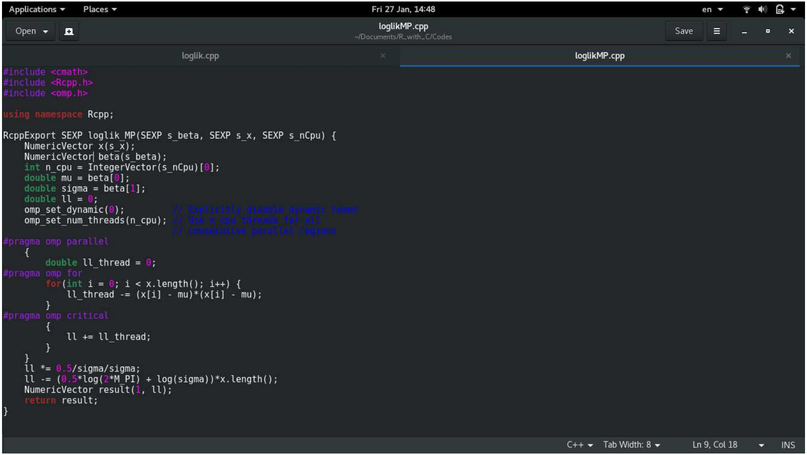
```
llR = function(par, x) {
mu = par[1]
sigma = par[2]
sum(-1/2*log(2*pi) - log(sigma) - 1/2*((x - mu)^2)/sigma^2) }
```

C++ code



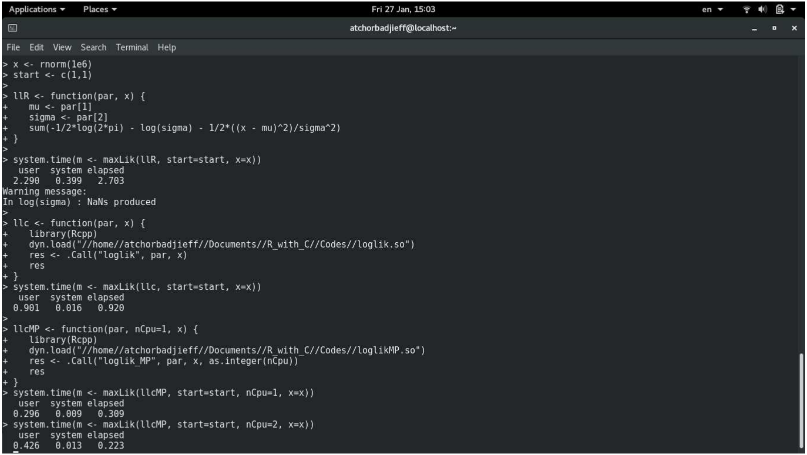
```cpp
#include <cmath>
#include <Rcpp.h>

using namespace Rcpp;

RcppExport SEXP loglik(SEXP s_beta, SEXP s_x) {
    NumericVector x(s_x);
    NumericVector beta(s_beta);
    // make Rcpp vector out of R SEXP
    double mu = beta[0];
    // first element is 0 in C++
    double sigma = beta[1];
    double ll = 0;
    for(int i = 0; i < x.length(); i++) {
        ll -= (x[i] - mu)*(x[i] - mu);
    }
    ll *= 0.5/sigma/sigma;
    ll -= (0.5*log(2*M_PI) + log(sigma))*x.length();
    NumericVector result(1, ll);
    // create 'numeric' vector of length 1, filled with
    // ll values
    return result;
}
```

openMP C++ code



```cpp
#include <cmath>
#include <Rcpp.h>
#include <omp.h>

using namespace Rcpp;

RcppExport SEXP loglik_MP(SEXP s_beta, SEXP s_x, SEXP s_nCpu) {
    NumericVector x(s_x);
    NumericVector beta(s_beta);
    int n_cpu = IntegerVector(s_nCpu)[0];
    double mu = beta[0];
    double sigma = beta[1];
    double ll = 0;
    omp_set_dynamic(0);         // Explicitly disable dynamic teams
    omp_set_num_threads(n_cpu); // Use n_cpu threads for all
                                // consecutive parallel regions
#pragma omp parallel
    {
        double ll_thread = 0;
#pragma omp for
        for(int i = 0; i < x.length(); i++) {
            ll_thread -= (x[i] - mu)*(x[i] - mu);
        }
#pragma omp critical
        {
            ll += ll_thread;
        }
    }
    ll *= 0.5/sigma/sigma;
    ll -= (0.5*log(2*M_PI) + log(sigma))*x.length();
    NumericVector result(1, ll);
    return result;
}
```

Execution and Elapsed Time



```
> x <- rnorm(1e6)
> start <- c(1,1)
>
> llR <- function(par, x) {
+    mu <- par[1]
+    sigma <- par[2]
+    sum(-1/2*log(2*pi) - log(sigma) - 1/2*((x - mu)^2)/sigma^2)
+ }
>
> system.time(m <- maxLik(llR, start=start, x=x))
   user  system elapsed
  2.290   0.399   2.703
Warning message:
In log(sigma) : NaNs produced
>
> llc <- function(par, x) {
+    library(Rcpp)
+    dyn.load("//home//atchorbadjieff//Documents//R_with_C//Codes//loglik.so")
+    res <- .Call("loglik", par, x)
+    res
+ }
> system.time(m <- maxLik(llc, start=start, x=x))
   user  system elapsed
  0.901   0.016   0.920
>
> llcMP <- function(par, nCpu=1, x) {
+    library(Rcpp)
+    dyn.load("//home//atchorbadjieff//Documents//R_with_C//Codes//loglikMP.so")
+    res <- .Call("loglik_MP", par, x, as.integer(nCpu))
+    res
+ }
> system.time(m <- maxLik(llcMP, start=start, nCpu=1, x=x))
   user  system elapsed
  0.296   0.009   0.309
> system.time(m <- maxLik(llcMP, start=start, nCpu=2, x=x))
   user  system elapsed
  0.426   0.013   0.223
```

Not included in the lecture, but must be considered:

- Computational Algebra:
  - Armadillo (C++ linear algebra library; signal processing)
  - RcppArmadillo extends Rccp
- C++ template implementation

References:

- Dirk Eddelbuettel: dirk.eddelbuettel.com/code/rcpp.html
- OpenMP examples are copied from http://www.parallelr.com/

- Thank you!