

StatML CDT Computational Training Part 2: Extending R with Rcpp

2020-02-17

Aims

- 1 Cursory overview of Rcpp
- 2 Motivate extending R, and why Rcpp
- 3 Learn through examples
- 4 Provide references to resources for more in depth understanding

Materials available at github.com/jscott6/StatML-Comp-Training-2020

Why Interface R with C++?

C++ is a *compiled* language.

- Safety (type checking as C++ is **statically typed**)
- Performance (compile time optimizations)

Some tasks require advanced algorithms and data structures

- Not available directly in R
- C++ STL and Boost libraries

Advantages of both compiled and interpreted languages.

Rcpp makes it easy...

Rcpp is an extension tool for R

Provides C++ classes which help to interface R and C++ using `.Call()` interface.

An approachable API (unlike the R API).

R Objects C Representation

Everything in R is an object, and has a **base type**

`typeof()` returns the base type of an arbitrary R object

- Examples: “closure”, “integer”, “double”, “list”, “S4”

R is (largely) written in C.

R objects are stored, at the C-level as a SEXP (S-expression). 27 sub-types including

- NILSXP (null)
- REALSXP (double Vector)
- INTSXP (integer Vector)
- LGLSXP (logical Vector)
- CLOSXP (function)
- ENVSXP (environment)
- XPTRSXP (external pointers)

```
.Call("myfunc", arg1, arg2, ...)
```

On the C++ side:

```
#include <R.h>
```

```
#include <Rinternals.h>
```

```
SEXP myfunc(SEXP arg1, SEXP arg 2,...);
```

(For details, see “Writing R extensions”)

Example: convert `dot <- function(a,b) sum(a * b)` into a C++ function

- 1 using R API
- 2 using Rcpp API

Using R API

```
#include <R.h>
#include <Rinternals.h>

SEXP dot(SEXP a, SEXP b) {
    int n;
    SEXP sum;
    a = PROTECT(coerceVector(a, REALSXP));
    b = PROTECT(coerceVector(b, REALSXP));
    n = length(a);
    sum = PROTECT(allocVector(REALSXP, 1));
    REAL(sum)[0] = 0;
    for (int i = 0; i < n; i++) {
        REAL(sum)[0] += REAL(a)[i] * REAL(b)[i]
    }
    UNPROTECT(3);
    return sum;
}
```

Rcpp Equivalent

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

// [[Rcpp::export]]
double dot(const NumericVector& a, const NumericVector& b) {
    return sum(a * b);
}
```


Provides helper classes through `Rcpp.h`, and converters.

Simpler to read and maintain.

- automatic type conversions (most of the time)
- Automatic wrapping for use with `.Call`
- Rcpp sugar

Safer : automatic memory management.

- No manual calls to `PROTECT` and `UNPROTECT`

Rcpp Matching Classes

Rcpp provides matching C++ classes for R data types.

Can easily pass from R to C++, or from C++ to R.

Atomic Vector Type	C Representation	Rcpp Vector	Rcpp Matrix
"double"	REALSXP	NumericVector	NumericMatrix
"integer"	INTSXP	IntegerVector	IntegerMatrix
"logical"	LGLSXP	LogicalVector	LogicalMatrix

Rcpp provides converter functions

- From SEXP to Rcpp type: `Rcpp::as<>()`
 - Example:** `NumericVector b = as<NumericVector>(a);`
- From Rcpp type to SEXP: `Rcpp::wrap()`
 - Example:** `SEXP c = wrap(b);`

Rcpp: A first example

Install using `install.packages("Rcpp")`

C++ source file `timesTwo.cpp` has

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

// [[Rcpp::export]]
NumericVector timesTwo(NumericVector x) {
  return x * 2;
}
```

R script `timesTwo.R` with

```
library(Rcpp)
sourceCpp("examples/timesTwo/timesTwo.cpp")
a <- 1:3
timesTwo(a)

## [1] 2 4 6
```

Rcpp Attributes

Rcpp attributes are annotations to C++ files that provide additional information to the compiler.

Two important attributes:

- `Rcpp::export`: export a C++ function to R
- `Rcpp::depends`: specify build dependencies for `sourceCpp`

For a C++ function to be handled using `Rcpp::export` it must

- Return type either `void` or compatible with `Rcpp::wrap`
- Arguments compatible with `Rcpp::as<>()`
- Global namespace
- Fully qualified type names for arguments and return value (apart from Rcpp types).

Making Available in R

Use `sourceCpp("path/to/foo.cpp")`:

- parses c++ file "foo.cpp"
- looks for `Rcpp::export` attributes to determine exported C++ functions
- creates wrappers for exported functions (check using `verbose = TRUE`)
- compiles, links and loads wrapper into R under the C++ name

Related functions include

```
library(Rcpp)
evalCpp("2+2")
```

```
## [1] 4
```

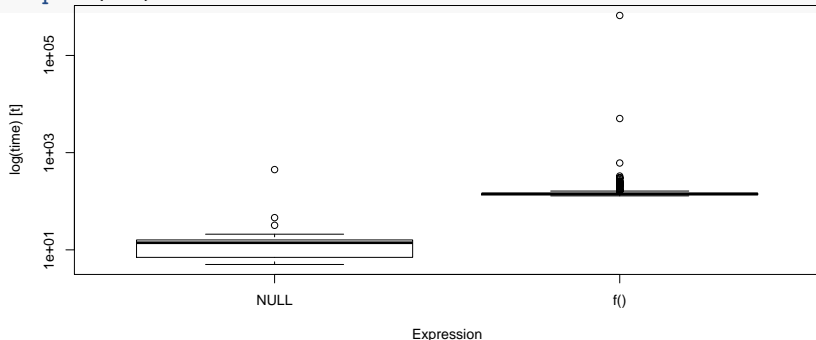
```
cppFunction("double add(double a, double b) { return a + b; } ")
add(3,2)
```

```
## [1] 5
```

Slow R

Function calls have a **high overhead** in R.

```
f <- function() NULL  
res <- microbenchmark::microbenchmark(NULL, f(), times=1000L)  
boxplot(res)
```



Why? code translated by interpreter, variables scoped, type checked dynamically, methods dispatched etc. . .

Avoid large numbers of function calls

Loops which cannot be vectorised, i.e. each iteration not independent of others. Think MCMC

Recursive functions, i.e. naive implementation of Fibonacci sequence

Second Example: Fibonacci Sequence

$$F_n = F_{n-1} + F_{n-2}$$

with initial conditions $F_1 = 1$, $F_2 = 2$.

A recursive R implementation

```
fibR <- function(n) {  
  if (n < 3) n  
  else fibR(n-1) + fibR(n-2)  
}
```

How many calls are made to fib for a given n?

- fib(n) has $2F_n - 1$ function calls (check). Grows exponentially.

Intermission: how to improve without C++? *Hint: avoid repeated work*

Fibonacci Sequence: C++

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

// [[Rcpp::export]]
int fibCpp(int n) {
    if (n < 3) return n;
    return fibCpp(n-1) + fibCpp(n-2);
}
```

Benchmark

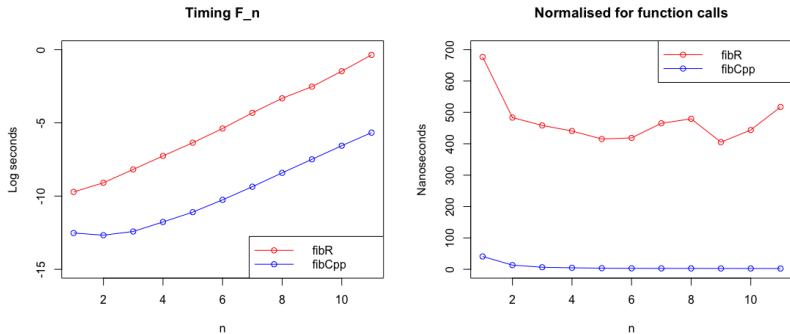


Figure 1: Benchmarking recursive Fibonacci programs in R and C++

Exercise 1

Create a dataframe in R

```
col1 <- runif (12^5, 0, 2)
col2 <- rnorm (12^5, 0, 2)
col3 <- rpois (12^5, 3)
col4 <- rchisq (12^5, 2)
df <- data.frame (col1, col2, col3, col4)
```

Consider a function intended to append a column to df, where entries take value "greater_than_4" if other four columns sum > 4 , otherwise "lesser_than_4".

- 1 Write an R function for this which is not vectorised.
- 2 Write a vectorised R version
- 3 Write a C++ implementation using Rcpp
- 4 Profile code using `system.time()`

Reference: <https://www.r-bloggers.com/strategies-to-speedup-r-code/>

Rcpp modules are based on Boost.Python modules.

Particularly useful for exposing C++ classes to R.

Why Expose C++ Classes?

Manipulate C++ objects interactively in R.

Can be used to implement advanced data structures and efficient methods to operate on them

Leverage C++'s strong OOP system.

A Dummy Class

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

class Rectangle {
public:
    Rectangle(double width, double height):
        width_(width), height_(height) { }
    double area() { return width_ * height_; }
private:
    double width_, height_;
};

RCPP_MODULE(Rectangle_Module) {
    class_<Rectangle>("Rectangle")

        .constructor<double, double>()
        .method("area", &Rectangle::area)
        ;
}
```