Statistical Analysis of Big Data Using R

Speeding up your R code

Introduction

1. What is Rcpp?
2. Essential Rcpp examples
3. Under the hood
4. Practical Rcpp advice
5. Performance considerations when programming in R and Rcpp
6. Dimension reduction example
7. TBD: anything else?

TBD: Delete Me

1. Motivation for Rcpp, from AdvStatComputing intro slides (DONE)
2. What is C++, what is Rcpp? (DONE)
3. Difficulties of Rcpp — image of the bomb (DONE)
4. Some "essential" examples, not worrying about speed improvement (DONE)
5. Under the hood (DONE)
6. Practical advice about print statements, logging, throwing exceptions, modular code practice, measuring elapsed time (DONE).
7. Demonstrate vectorization vs. loops vs. apply vs. C++ loops here, and avoiding repeated cbind/rbind (DONE)
8. OPG application (DONE)
9. Profiling speed (Rprof?) and memory (Valgrind?) ***
10. Maybe a few of the "extras" notes, about packages, etc (DONE)
11. Make sure to add some references to books, websites, etc etc (DONE)

Motivation

- Many of us prefer R for the computing portion of our work.
- R programming is convenient, but resulting code can be slow. This slowness can be quite painful in large simulations or with very large datasets, for example.
- Rcpp allows us to write the slow parts of your code in C++ (fast, efficient memory use) and call it through plain R (friendly & familiar).

C and C++

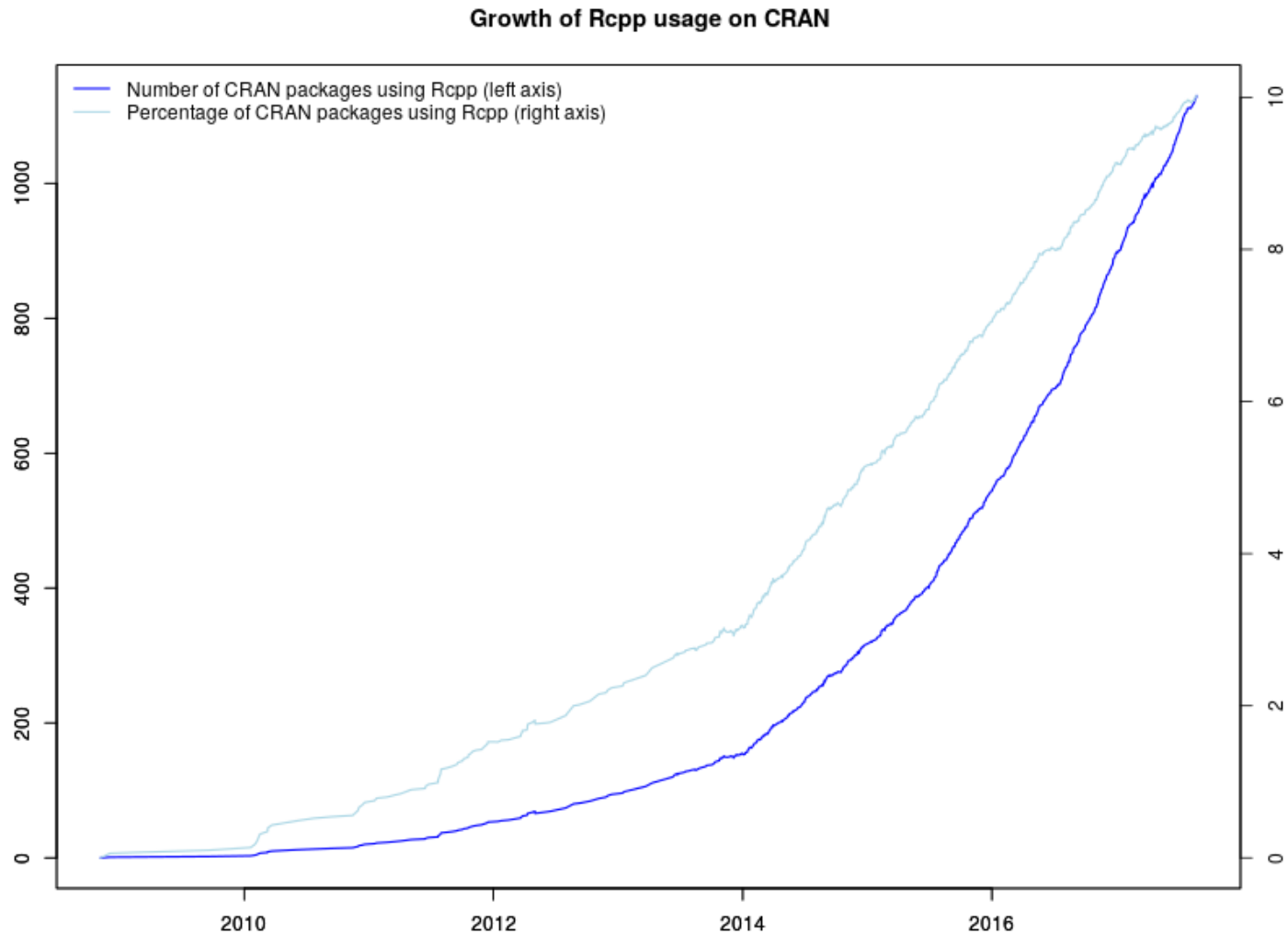
- C is a low-level, imperative procedural language.
  1. It was designed to be compiled.
  2. Provides low-level access to memory.
  3. Provides language constructs that map efficiently to machine instructions.
- C was originally developed by Dennis Ritchie between 1969 and 1973 at Bell Labs, and used to re-implement the UNIX operating system.
- C++ was developed by Bjarne Stroustrup in 1979 at Bell Labs, as an extension of C.
- C++ builds on standard C with features such as function overloading and support for object-oriented programming.
- Libraries such as the Standard Template Library (STL) that provide a variety of data structures and algorithms for C++ programmers.

What is Rcpp?

- Rcpp is a suite of packages that facilitates interoperability between R and C++ written mainly by Dirk Eddelbuettel and Romain Francois.

- The Rcpp ecosystem includes the packages Rcpp, RcppArmadillo, RcppEigen, and RcppGSL.

Rcpp Usage on CRAN



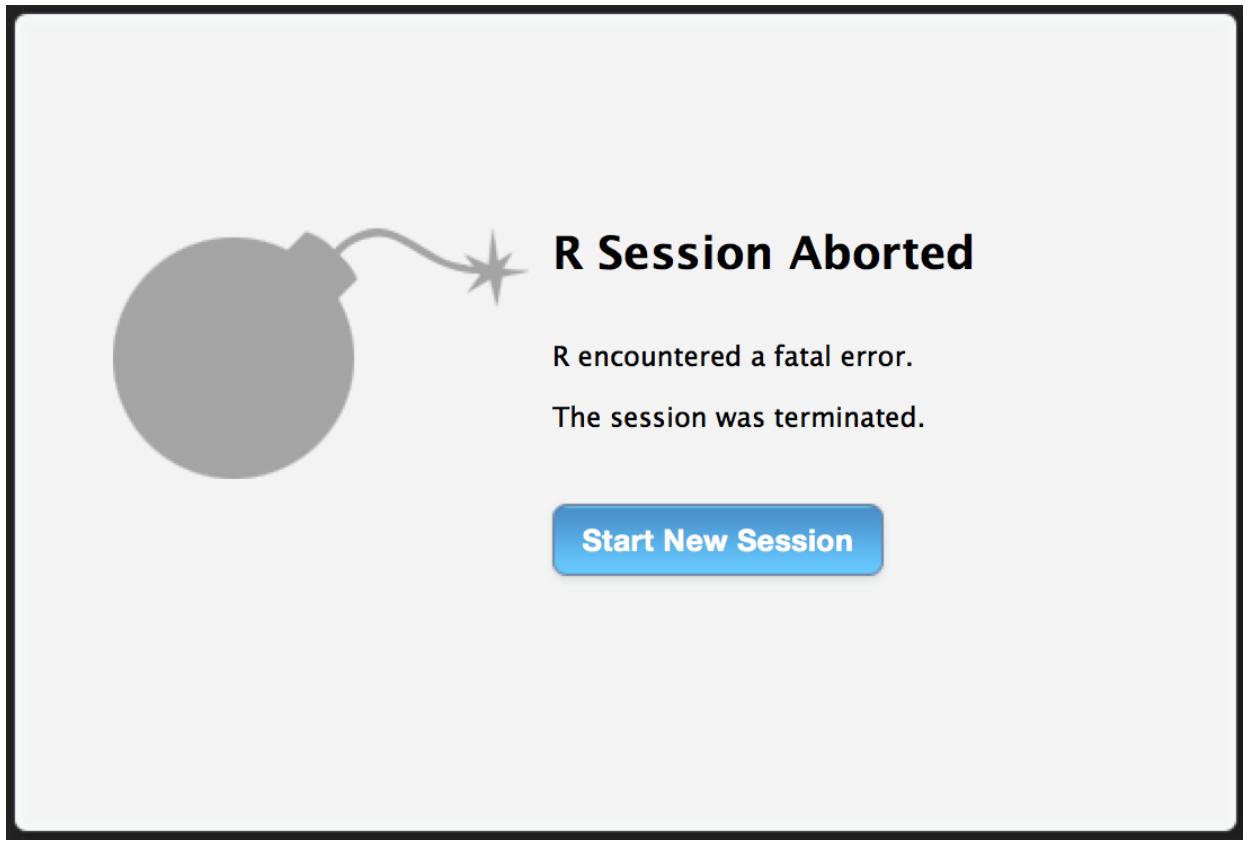
"Rcpp now used by 10 percent of CRAN packages"
Source: `http://dirk.eddelbuettel.com/blog/2017/08/23/`

Why use Rcpp?

- Compiled code is fast. More efficient use of processors can reduce the need for parallelization.
- It is well-documented (sometimes). For example:
  1. Rcpp Gallery: articles and code examples for the Rcpp package http://gallery.rcpp.org/
  2. Rcpp vignettes: https://cran.r-project.org/web/packages/Rcpp/vignettes/
- Authors have tried to make it easy to install, learn, and use.
- Rcpp extensions are available to access to external C++ libraries.

Potential Difficulties with Rcpp

- Many R users don't know C++. Programmers who know both will find C++ more time consuming.
- Rcpp code can be difficult to debug and profile.



Essential Rcpp Examples

Essential Rcpp Examples

. . . Demonstration . . .

(See `essential.Rmd`)

Under the Hood

R Application Programming Interface

- R API is based on a set of functions and macros operating on S Expressions (SEXPs).
- An SEXP is a pointer to a C struct which is an internal representation of an R object.
- In C, **all R objects are SEXPs**.
- R provides several calling conventions:
  1. .C() : most basic interface intended for passing standard vectors (not including lists)
  2. .Call(): allows access to R data structures inside of C++
- Every .Call() access uses this interface pattern:

```
SEXP my_function(SEXP x, SEXP y) {
  ...
}
```

.C Interface

Compiling C code into a shared object library usable by R

- Write a function in C that returns void. All of the arguments for the C function should be pointers.

```
void foo(int* nin, double* x)
{
    int n = nin[0];
    for (int i = 0; i < n; i++) {
        x[i] = x[i] * x[i];
    }
}
```

- Compile that function in a UNIX shell, creating a .so file (shared object library). The compiled function will be an object in the .so file that can be accessed from R.

```
R CMD SHLIB foo.c
```

.C Interface

Compiling C code into a shared object library usable by R

- Load the .so in R.

```
dyn.load("foo.so")
```

- Create a wrapper function that allows you to call the C code.

```
foo <- function(x) {
    ret <- .C("foo", length(x), as.double(x))
    ret[[2]]
}
```

```
> foo(1:10)
[1]   1   4   9  16  25  36  49  64  81 100
```

.Call Interface

Compiling C++ code into a shared object library usable by R

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

extern "C" {
    SEXP foo(SEXP sexp_x)
    {
        int n = length(sexp_x);
        double* x = REAL(sexp_x);

        SEXP sexp_y = PROTECT(allocVector(REALSXP, n));
        double* y = REAL(sexp_y);

        for (int i = 0; i < n; i++) {
            y[i] = x[i] * x[i];
        }

        UNPROTECT(1);
        return sexp_y;
    }
}
```

- Compile the function in a UNIX shell.

```
R CMD SHLIB foo.cpp
```

- Load the .so in R.

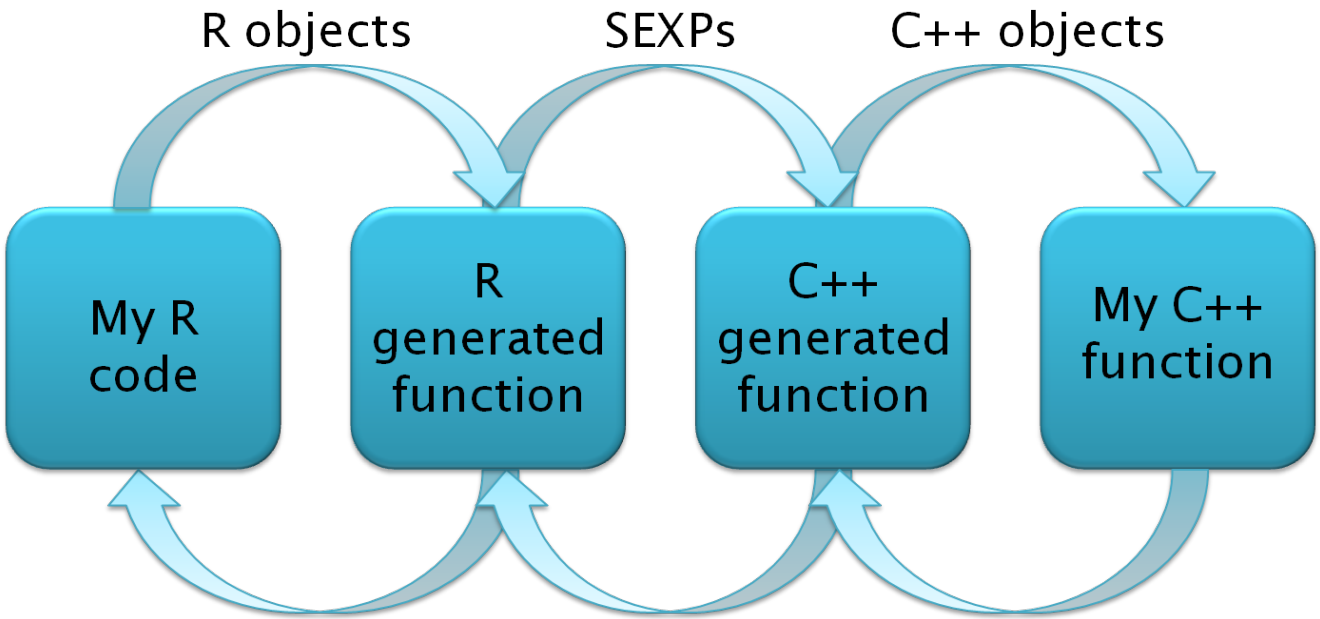
```
> dyn.load("foo.so")
```

- Create a wrapper function that allows you to call the C code.

```
foo <- function(x) {
    ret <- .Call("foo", as.numeric(x))
    return(ret)
}
```

```
> foo(1:10)
[1]   1   4   9  16  25  36  49  64  81 100
```

A Function Call in Rcpp



A Function Call in Rcpp

. . . Demonstration . . .

(See `underthehood.Rmd`)

Practical Rcpp

Practical Rcpp

. . . Demonstration . . .

(See `practical.Rmd`)

Performance Considerations

Performance Considerations

- To be an efficient R/Rcpp programmer, it helps to understand tradeoffs between:
  1. loops,
  2. the apply family (apply, lapply, sapply, mapply, vapply, etc), and
  3. matrix algebra.
- There is an art to writing good code. It should have good performance, while being as simple and readable as possible (which can be subjective).
- Loops and apply statements are very slow when computations inside are small. Either option can be more readable, depending on the situation.
- Matrix algebra in R directly uses compiled libraries (BLAS, LAPACK, etc), so it is generally very fast.
- Sometimes, loops and apply statements can be recast as matrix algebra to improve performance of R code. This technique is referred to as **vectorization**. If overused, it can lead to cryptic code, and sometimes requires much more memory than a loop/apply.
- Loops in Rcpp are fast and do not need to be avoided. But matrix libraries can still be much faster when applicable.

Toeplitz Matrix Example

- A Toeplitz covariance matrix has the form

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_0 & \sigma_1 & \sigma_2 & \cdots & \sigma_{p-1} \\ \sigma_1 & \sigma_0 & \sigma_1 & \cdots & \sigma_{p-2} \\ \sigma_2 & \sigma_1 & \sigma_0 & \cdots & \sigma_{p-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma_{p-1} & \sigma_{p-2} & \sigma_{p-3} & \cdots & \sigma_0 \end{pmatrix}.$$

- In other words, the (i, j)th element of $\boldsymbol{\Sigma}$ is $\sigma_{|i-j|}$ for $i, j \in \{1, \ldots, p\}$, so that the covariance depends only on the lag.
- A special case is the autoregressive covariance AR(1)

$$\text{Cov}(Y_s, Y_t) = \sigma^2 \rho^{|t-s|}, \quad \text{Var}(Y_t) = \sigma^2$$

  for $s, t \in \{1, 2, \ldots\}$.
- Let us consider some R and Rcpp functions to form a symmetric Toeplitz matrix given its first row

$$\boldsymbol{\sigma} = (\sigma_0, \ldots, \sigma_{p-1}).$$

Toeplitz Matrix Example

. . . Demonstration . . .

(See `toeplitz.Rmd`)

Dimension Reduction Application

Dimension Reduction Application

- Recall the OPG algorithm.

Outer Product of Gradients (OPG) Algorithm

Inputs: dimension d and bandwidth h.

for $j = 1$ to n **do**

Compute weights $w_{ij} = \frac{K_h(\boldsymbol{x}_i - \boldsymbol{x}_0)}{\sum_{\ell=1}^{n} K_h(\boldsymbol{x}_\ell - \boldsymbol{x}_j)}$ for $i = 1, \ldots, n$.

Compute $(\hat{\alpha}_j, \hat{\gamma}_j)$ by minimizing $Q_{\boldsymbol{B}}(\alpha_j, \gamma_j)$ via WLS.

end for

return $\hat{\boldsymbol{B}}$ as matrix of the first d eigenvectors of $\hat{\boldsymbol{\Sigma}} = \frac{1}{n} \sum_{j=1}^{n} \hat{\gamma}_j \hat{\gamma}_j^\top$.

- **Question**: Can we drastically improve the performance with Rcpp over a plain R implementation?

- Let us implement the computation of weight and local regression coefficients in Rcpp.

Dimension Reduction Application

. . . Demonstration . . .

(See `opg.Rmd`)

Conclusions

Other Rcpp Topics

- Rcpp can be used within R packages. Support for package development is built into Rstudio.
- Rcpp Sugar provides "syntactic sugar" to make C++ programming feel more like R programming.
- Rcpp Attributes provide a way to "tag" C++ code to inject it with certain features. For example, we have seen Rcpp::export and Rcpp::depends.
- Rcpp Modules allow the programmer to expose C++ classes to R. This provides another level of interoperability in object-oriented programs.
- RInside allows R code to be embedded in C++ programs.

Where to go from here?

Rcpp books

- Eddelbuettel (2013, *Seamless R and C++ Integration with Rcpp*)
- Wickham (2014, *Advanced R*)

Rcpp online

- Rcpp website: `http://www.rcpp.org`
- Armadillo website: `http://arma.sourceforge.net`
- Ask `http://www.google.com`. E.g. "Rcpp how to make a 3d array?"
- StackOverflow: `http://stackoverflow.com`

C++ books

- Stroustrup (2013, *The C++ Programming Language*)
- Eubank and Kupresanin (2011, *Statistical Computing in C++ and R*)

C++ online

- `http://www.cplusplus.com`
- Standard Template Library: `http://www.sgi.com/tech/stl`
- Boost: `http://www.boost.org`

References I

Dirk Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. Springer, 2013.

Randall L. Eubank and Ana Kupresanin. *Statistical Computing in C++ and R*. Chapman and Hall/CRC, 2011.

Bjarne Stroustrup. *The C++ Programming Language,*. Addison-Wesley Professiona, 4th edition, 2013.

Hadley Wickham. *Advanced R*. Chapman and Hall/CRC, 2014.