Introduction to Rcpp

Drew Schmidt

April 8, 2014

http://r-pbd.org/NIMBioS

Contents

1. Introduction to Rcpp
   - What is Rcpp?
   - Why would we care?
   - Rcpp Pros and Cons

Rcpp

- A package to make utilizing compiled code from R easier.
- A package ecosystem (Rcpp, RcppArmadillo, RcppEigen, . . .).
- Enables R to utilize C++ code.
- Lots of higher-level "magic" to make writing C++ code (for R) easier.

What Rcpp **is**

- A way of interfacing R to compiled code.
- A set of utilities to make writing C++ more convenient for R users.
- A tool which requires C++ knowledge to effectively utilize.

What Rcpp **is not**

- Magic.
- Automatic R-to-C++ converter.
- The only way to bring compiled code to R.
- A way around having to learn C++.
- A tool to make existing R functionality faster (unless you rewrite it).
- As easy to use as R.

Problems with R

- R is slow.
- If you don't know what you're doing it's *REALLY* slow.
- R loops often 100x slower (or worse) than native C/C++.
- R aggressively over-uses memory.
- R is single-threaded.

Performance and Accuracy

Sometimes $\pi = 3.14$ is (a) infinitely faster than the "correct" answer and (b) the difference between the "correct" and the "wrong" answer is meaningless. . . . The thing is, some specious value of "correctness" is often irrelevant because it doesn't matter. While performance almost always matters. And I absolutely detest the fact that people so often dismiss performance concerns so readily.

— Linus Torvalds, August 8, 2008

Why use R at all?

- Most diverse set of statistical methods available.
- Rapid prototyping.
- CRAN packages.
- Syntax is designed for data.

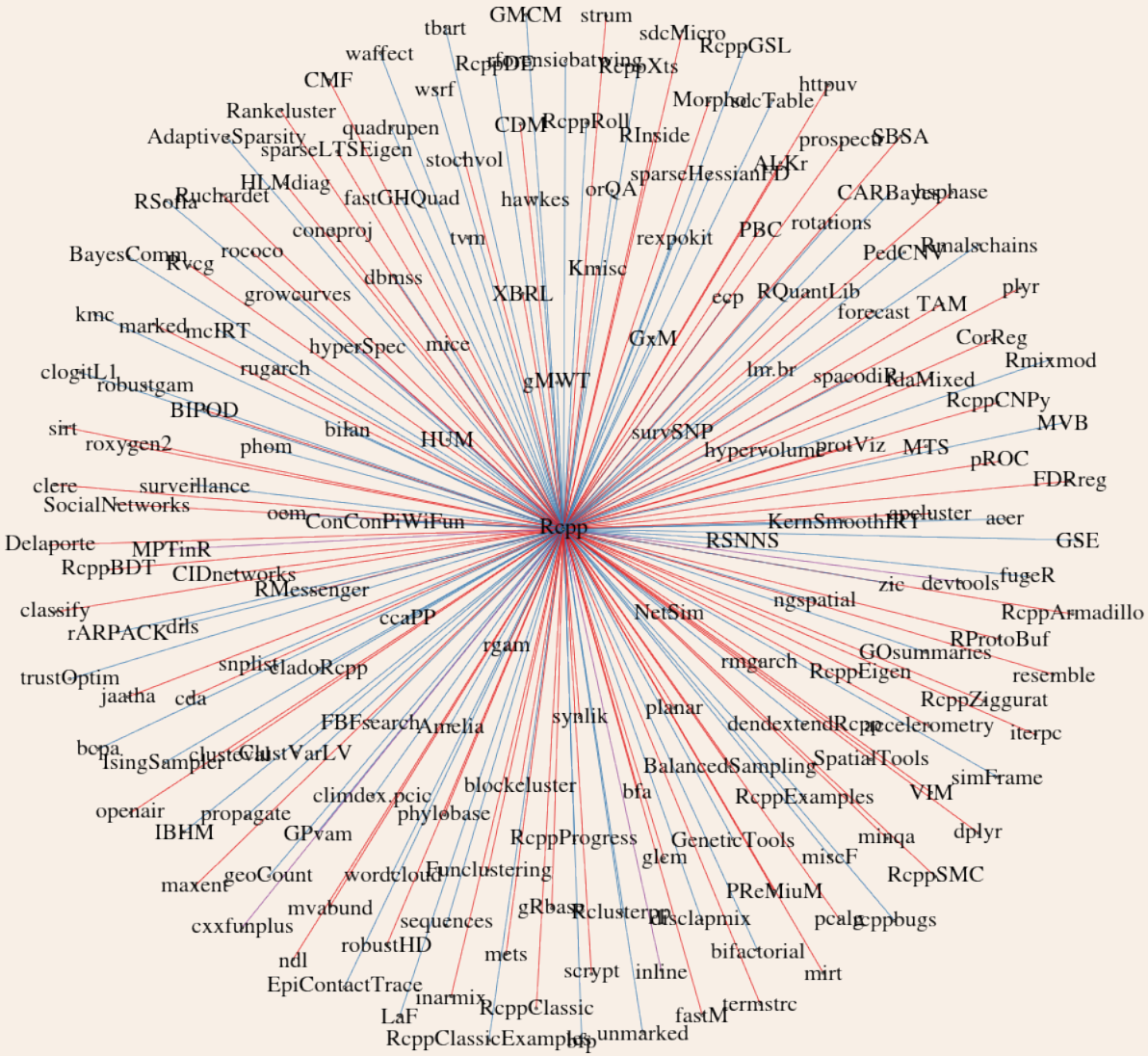
Suggested Model for Developing Efficient R

1. Prototype in R.
2. *Profile, Profile, Profile.*
3. Move computationally expensive pieces to compiled language (C/C++/Fortran).
4. Use R as high-level interface for low-level code.

Advantages of Rcpp

- Compiled code is *fast*.
- Easy to install.
- Easy to use (comparatively).
- Better documented than alternatives.
- Large, friendly, helpful community.

Rcpp Package Dependencies



Disadvantages

- It's C++ (there be dragons).
- Difficult to debug/profile.
- Rcpp code *must* be GPL licensed.
- Rcpp designed to only work with R.

Inline

For the simplicity and reproducibility, we will be using Rcpp by way of inline throughout the examples.

- Allows you to easily compile Rcpp code from R.
- *Not a permanent solution.*
- Meant for rapid prototyping and demonstration.
- Long term solution: put things in an R package.

2 Installation

- Installation Prerequisites
- Documentation and Help

Package Installation

- To install Rcpp, you need to have a build environment (GNU compilers).
- Some platforms require more work than others...
- RcppGSL requires a system installation of GSL.

Package Installation: Linux

- Use your package manager to install everything.
- Have a beer.

Package Installation: Mac

- Install Xcode from the Appstore: https://itunes.apple.com/us/app/xcode/id497799835?mt=12
- In Xcode, selecte Preferences, then Downloads, and install the Command Line Tools.
- If you will be using fortran, also install gfortran: http://gcc.gnu.org/wiki/GFortranBinaries
- Install Rcpp via `install.packages("Rcpp")`
- Install inline via `install.packages("inline")`

Package Installation: Windows

- Follow the instructions here
  http://cran.rstudio.com/bin/windows/Rtools/
- Install Rcpp via `install.packages("Rcpp")`
- Install inline via `install.packages("inline")`

For help, see:
http://tonybreyal.wordpress.com/2011/12/07/installing-rcpp-on-windows-7-for-r-and-c-integration/
and/or http://www.rstudio.com/ide/docs/packages/prerequisites

Test Code

To ensure that Rcpp and inline are correctly installed, run this sample code in an R session:

```
library(inline)

body <- "std::cout << \"It works\" << std::endl;"
test <- cxxfunction(signature(), body=body,
    plugin="Rcpp")

test()
```

Documentation

- The numerous Rcpp vignettes http://cran.r-project.org/web/packages/Rcpp/index.html (start with Introduction, quickref, and FAQ).
- *High Performance Functions with Rcpp*, Hadley Wickham: http://adv-r.had.co.nz/Rcpp.html
- *Seamless R and C++ Integration with Rcpp* (book), http://www.amazon.com/Seamless-Integration-Rcpp-Dirk-Eddelbuettel/dp/1461468671/ref=sr_1_1?ie=UTF8

Where to Get Help

- The documentation.
- Stackoverflow: http://stackoverflow.com/questions/tagged/rcpp
- Rcpp-devel list: http://lists.r-forge.r-project.org/mailman/listinfo/rcpp-devel

3 Rcpp Basics

- The Bare Minimum
- Compiling Your Code
- Using Your Compiled Code

Rcpp Basics

Every Rcpp function consists of 2 pieces:

- C++ code
- R-level wrapper code

Rcpp Basics

Rcpp Function myRcpp.cpp

```
#include <Rcpp.h>

RcppExport SEXP my_cxx_fun(SEXP x, SEXP y){
  ...
}
```

Wrapper myR.r

```
1 my_R_fun <- function(x, y){
2   myresult <- .Call('my_cxx_fun', x, y,
        PACKAGE='my_package')
3
4   return(myresult)
5 }
```

Making 'Hello World' Needlessly Complicated!

rcpp_hw.cpp

```
#include <Rcpp.h>

RcppExport SEXP my_hw()
{
  Rcpp::Rcout << "Hello, world!" << std::endl;

  return R_NilValue;
}
```

4 Ways to Compile Our Rcpp Hello World Example

- The really hard way.
- The hard way.
- The package way.
- The inline way.

The really hard way

```
clang++ -I/usr/share/R/include `Rscript -e
    'Rcpp:::CxxFlags()'` -fpic  -O3 -pipe  -g  -c
    rcpp_hw.cpp -o rcpp_hw.o
clang++ -shared -o rcpp_hw.so rcpp_hw.o -L/usr/lib/R/lib
    -lR
```

DO NOT EVER DO THIS

The hard way

```
export PKG_CPPFLAGS=`Rscript -e 'Rcpp:::CxxFlags()'`
export PKG_LIBS=`Rscript -e 'Rcpp:::LdFlags()'`
R CMD SHLIB rcpp_hw.cpp
```

Use this if:

- You don't want to recompile every time you start R (some amount of permanence), AND
- You don't want to set up an R package yet.

The package way

We'll come back to this . . .

The inline way

```
1 library(inline)
2
3 body <- "
4   Rcpp::Rcout << \"It works\" << std::endl;
5
6   return R_NilValue;
7 "
8
9 cxxfunction(signature(), body=body, plugin="Rcpp")
```

Calling your Compiled Code from R

- Use the `.Call()` function.

```
1 .Call("C_fun_name", arg1, arg2, package="mypackage")
```

- Be careful about passing arguments: types must match!

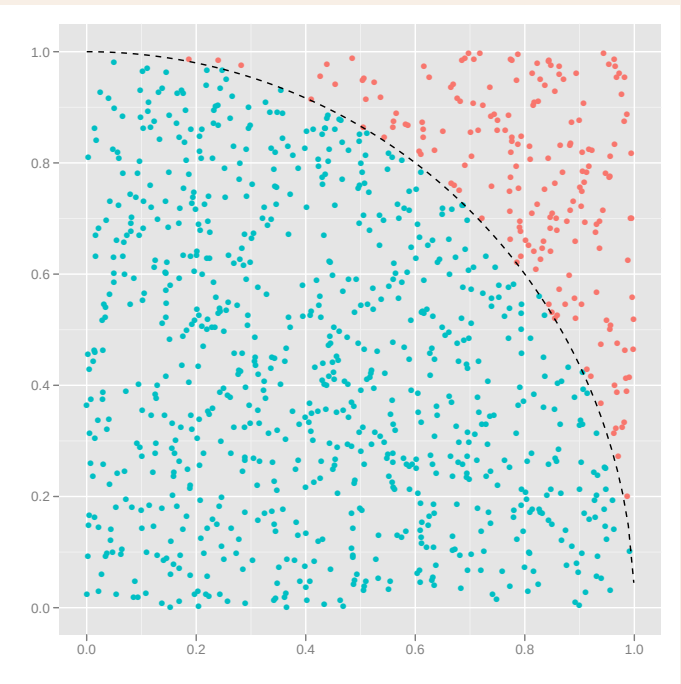
4 The Typical Monte Carlo Simulation for Estimating π

- Background and Outline
- Implementation
- Summary and Conclusions

Example 1 : Monte Carlo Simulation to Extimate π

Sample N uniform observations (x_i, y_i) in the unit square $[0, 1] \times [0, 1]$. Then

$$\pi \approx 4 \left(\frac{\#\ Inside\ Circle}{\#\ Total} \right) = 4 \left(\frac{\#\ \text{Blue}}{\#\ \text{Blue} + \#\ \text{Red}} \right)$$



Outline

- Implement in R using loops.
- Implement in R using vectorization.
- Implement in C++ with Rcpp.
- Benchmark.
- Examine other performance considerations.

Example 1: Monte Carlo Simulation Code

R Code (loops)

```
1  mcsim_pi_r <- function(n){
2    r <- 0L
3
4    for (i in 1:n){
5      u <- runif(1)
6      v <- runif(1)
7
8      if (u^2 + v^2 <= 1)
9        r <- r + 1
10   }
11
12   return( 4*r/n )
13 }
```

Example 1: Monte Carlo Simulation Code

R Code (vectorized)

```
1 mcsim_pi_r_vectorized <- function(n){
2   x <- matrix(runif(n * 2), ncol=2)
3   r <- sum(rowSums(x^2) <= 1)
4
5   return( 4*r/n )
6 }
```

Example 1: Monte Carlo Simulation Code

Rcpp Code

```
1  library(inline)
2  cxx_pi <- cxxfunction(signature(n_="int"), body='
3      int i, r = 0;
4      int n = Rcpp::as<int >(n_);
5      double u, v;
6
7      for (i=0; i<n; i++){
8        u = R::runif(0, 1);
9        v = R::runif(0, 1);
10
11       if (u*u + v*v <= 1)
12         r++;
13     }
14
15     return Rcpp::wrap( (double) 4.*r/n );
16
17   ',plugin="Rcpp"
18 )
19
20 mcsim_pi_r_rcpp <- function(n){
21   cxx_pi(as.integer(n))
22 }
```

Example 1: Monte Carlo Simulation Code

Benchmarking the Methods

```
1 library(rbenchmark)
2
3 n <- 50000
4
5 benchmark(R.loops = mcsim_pi_r(n),
6           R.vectorized = mcsim_pi_r_vectorized(n),
7           Rcpp = mcsim_pi_r_rcpp(n),
8           columns=c("test", "replications", "elapsed",
                "relative"))
```

```
          test replications elapsed relative
3         Rcpp          100   0.161    1.000
1      R.loops          100  34.974  213.256
2 R.vectorized          100   0.859    5.238
```

What About the Compiler?

Benchmarking the Methods

```
1  library(rbenchmark)
2  library(compiler)
3
4  mcsim_pi_r <- cmpfun(mcsim_pi_r)
5  mcsim_pi_r_vectorized <- cmpfun(mcsim_pi_r_vectorized)
6  mcsim_pi_r_rcpp <- cmpfun(mcsim_pi_r_rcpp)
7
8  n <- 50000
9
10 benchmark(R.loops = mcsim_pi_r(n),
11           R.vectorized = mcsim_pi_r_vectorized(n),
12           Rcpp = mcsim_pi_r_rcpp(n),
13           columns=c("test", "replications", "elapsed",
                 "relative"))
```

```
           test replications elapsed relative
3          Rcpp          100   0.161    1.000
1       R.loops          100  29.508  181.031
2 R.vectorized          100   0.729    4.472
```

Memory Usage

Loops:

$$\underbrace{4(n+3)}_{\text{Integers}} + \underbrace{8 \cdot 3}_{\text{Doubles}}$$

Vectorized:

$$\underbrace{4n}_{\text{Integers}} + \underbrace{8(2+2n)}_{\text{Doubles}}$$

Rcpp

$$\underbrace{4 \cdot 3}_{\text{Integers}} + \underbrace{8 \cdot 3}_{\text{Doubles}}$$

Summary

For $n = 50,000$ iterations and 100 replicates:

	Loops	Vectorized	Rcpp
Avg Runtime	0.3497s	0.0086s	0.0016s
Avg Runtime (compiled)	0.2951s	0.0073s	0.0016s
Memory Usage	195.348 KiB	1.144MiB	36 bytes

R Version: 3.0.2 stable
C++ Compiler: clang 3.2-7
CXX Flags: –O3

Conclusions

- Compiled code faster than R code.
- Vectorized code better than loops, but worse than compiled code.
- The bytecode compiler helps with (R) loops, but not much.
- R's memory footprint is terrible.
- *Reality check*: Most speed improvements through Rcpp are often much more modest than this.

5 From the lsa Package: Computing the Cosine Similarity Matrix

- Background and Outline
- Implementation
- Summary and Conclusions

Cosine Similarity

Recall from vector calculus that for vectors x and y

$$cos(x, y) = \|x\| \, \|y\| \, \cos(\theta(x, y))$$

We define

$$\text{cosim}(x, y) := \cos(\theta(x, y)) = \frac{x \cdot y}{\|x\| \, \|y\|}$$

Cosine Similarity Matrix

The cosine similarity matrix of a given (possibly non-square) matrix is the matrix of all pairwise similarities of the columns, i.e., given

$$X_{n,p} = [x_1, \ldots, x_p]$$

We take

$$\text{cosim}(X)_{ij} = \text{cosim}(x_i, x_j)$$

Original implementation from CRAN's lsa package 1

```
1  cosine <- function (x, y = NULL){
2      if (is.matrix(x) && is.null(y)) {
3          co = array(0, c(ncol(x), ncol(x)))
4          f = colnames(x)
5          dimnames(co) = list(f, f)
6          for (i in 2:ncol(x)) {
7              for (j in 1:(i - 1)) {
8                  co[i, j] = cosine(x[, i], x[, j])
9              }
10         }
11         co = co + t(co)
12         diag(co) = 1
13         return(as.matrix(co))
14     }
15     else if (is.vector(x) && is.vector(y)) {
16         return(crossprod(x, y)/sqrt(crossprod(x) *
               crossprod(y)))
17     }
18     else {
```

Original implementation from CRAN's lsa package 2

```
19          stop("argument mismatch. Either one matrix or two
                vectors needed as input.")
20      }
21  }
```

R Improvements 1

```
1  cosine2 <- function(x){
2    cp <- crossprod(x)
3    dg <- diag(cp)
4
5    co <- matrix(0.0, length(dg), length(dg))
6
7    for (j in 2L:length(dg)){
8      for (i in 1L:(j-1L)){
9        co[i, j] <- cp[i, j] / sqrt(dg[i] * dg[j])
10     }
11   }
12
13   co <- co + t(co)
14   diag(co) <- 1.0
15
16   return( co )
17 }
```

Naive Rcpp 1

```
1  library(inline)
2
3  fill_loop <- cxxfunction(
4    signature(cp_="matrix", dg_="numeric"),
5    body='
6      // Shallow copies
7      Rcpp::NumericMatrix cp(cp_);
8      Rcpp::NumericVector dg(dg_);
9
10     // Allocate return
11     Rcpp::NumericMatrix co(cp.nrow(), cp.ncol());
12
13     int i, j;
14
15     for (j=0; j<co.ncol(); j++){
16       for (i=0; i<co.nrow(); i++){
17         if (i == j)
18           co(i, j) = 1.0;
19         else
```

Naive Rcpp 2

```
            co(i, j) = cp(i, j) / std::sqrt(dg[i] * dg[j]);
        }
    }

    return co;
  ',plugin="Rcpp"
)


cosine_Rcpp <- function(x){
  cp <- crossprod(x)
  dg <- diag(cp)

  co <- fill_loop(cp, dg)

  return( co )
}
```

Rcpp improved 1

```
1  fill_loop2 <- cxxfunction(
2    signature(cp_="matrix", dg_="numeric"),
3    body='
4
5      // Shallow copies
6      Rcpp::NumericMatrix cp(cp_);
7      Rcpp::NumericVector dg(dg_);
8
9      const unsigned int n = cp.nrow();
10
11     // Allocate return
12     Rcpp::NumericMatrix co(n, n);
13
14     int i, j;
15
16     // Fill diagonal
17     for (j=0; j<n; j++)
18       co(j, j) = 1.0;
19
```

Rcpp improved 2

```
20      // Fill lower triangle
21      for (j=0; j<n; j++){
22        for (i=0; i<j; i++)
23          co(i, j) = cp(i, j) / std::sqrt(dg[i] * dg[j]);
24      }
25
26      // Copy lower triangle to upper
27      for (j=0; j<n; j++){
28        for (i=j+1; i<n; i++)
29          co(i, j) = co(j, i);
30      }
31
32      return co;
33    ',plugin="Rcpp"
34  )
35
36  cosine_Rcpp2 <- function(x){
37    cp <- crossprod(x)
38    dg <- diag(cp)
39
```

Rcpp improved 3

```
40    co <- fill_loop2(cp, dg)
41
42    return( co )
43  }
```

Relative Performance

Dimension	cosine()	cosine2()	`cosine3()`	`cosine4()`
100x100	112.695	25.763	1.119	1
200x200	69.511	14.101	1.056	1
300x300	55.878	10.609	1.027	1
400x400	50.977	8.345	1.029	1
500x500	44.342	7.298	1.038	1
600x600	40.804	6.136	1.022	1
700x700	38.685	5.488	1.020	1
800x800	35.565	4.647	1.014	1
900x900	33.680	4.331	1.010	1
1000x1000	31.413	3.928	1.010	1

Relative Performance with Bytecode Compilation

Dimension	cosine()	cosine2()	`cosine3()`	`cosine4()`
100x100	81.000	7.712	1.119	1
200x200	52.508	4.710	1.009	1
300x300	43.931	4.131	1.041	1
400x400	37.100	3.136	1.030	1
500x500	35.757	2.678	1.029	1
600x600	31.840	2.374	1.015	1
700x700	31.453	2.409	1.013	1
800x800	29.440	2.165	1.012	1
900x900	29.315	2.009	1.012	1
1000x1000	27.744	1.824	1.007	1

6 The Rcpp Ecosystem: RcppArmadillo, RcppGSL, ...

- RcppArmadillo
- RcppGSL

RcppArmadillo

- Armadillo: high-level C++ interface for BLAS and LAPACK
- RcppArmadillo: access to Armadillo syntax for R objects (at C level).

```
1  f <- function(x) list(outer=x %*% t(x), inner=t(x) %*% x)
2
3
4  body <- '
5    arma::mat v = Rcpp::as<arma::mat>(vs);
6    arma::mat op = v * v.t();
7    arma::mat ip = v.t()*v;
8
9    return Rcpp::List::create(
10     Rcpp::Named("outer")=op, Rcpp::Named("inner") = ip);
11 '
12
13 library(inline)
14 g <- cxxfunction(signature(vs="matrix"),
       plugin="RcppArmadillo", body=body)
15
16
17
18 x <- matrix(1:30, 10)
19 all.equal(f(x), g(x))
```

RcppGSL

- GSL: very comprehensive set of numerical routines.
- Thousands of functions: numerical integration, polynomials, FFT, RNG's, ...
- RcppGSL: high-level access to GSL (through R).
- Plays nice with Rcpp ecosystem.

RcppGSL

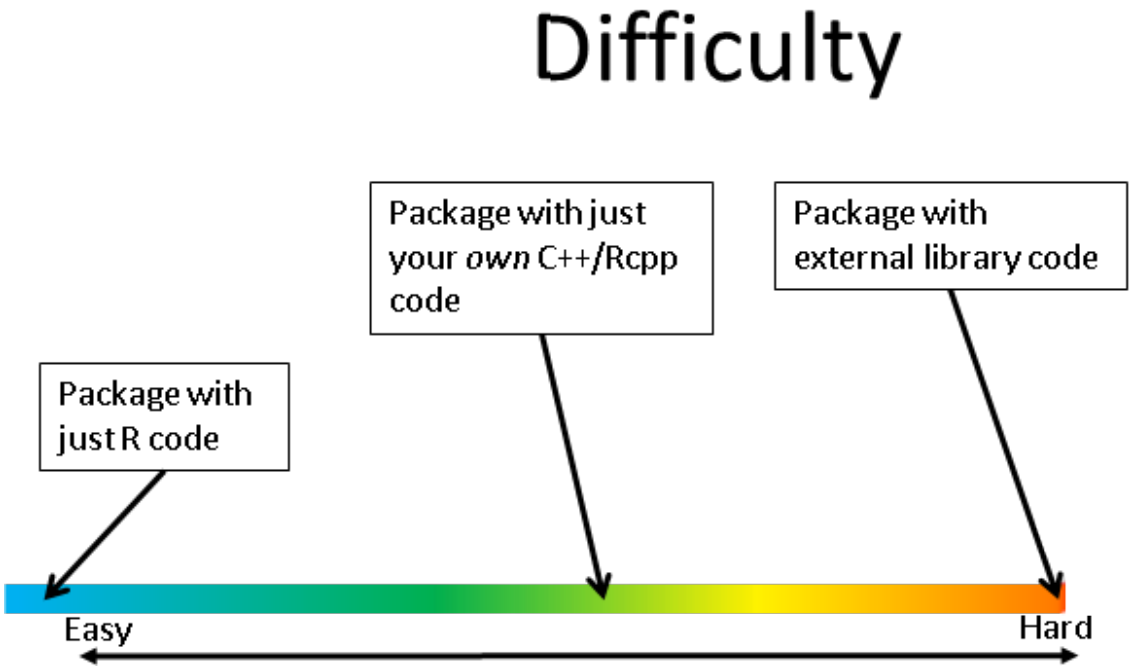
```
includes <- '
  #include <gsl/gsl_matrix.h>
  #include <gsl/gsl_blas.h>
'

body <- '
  RcppGSL::matrix<double> M = sM;
  int k = M.ncol();
  Rcpp::NumericVector n(k);

  for (int j = 0; j < k; j++) {
      RcppGSL::vector_view<double> colview =
          gsl_matrix_column(M, j);
      n[j] = gsl_blas_dnrm2(colview);
  }

  M.free() ;
  return n;
'

library(inline)
g <- cxxfunction(signature(sM="matrix"),
    plugin="RcppGSL", body=body, inc=includes)
```

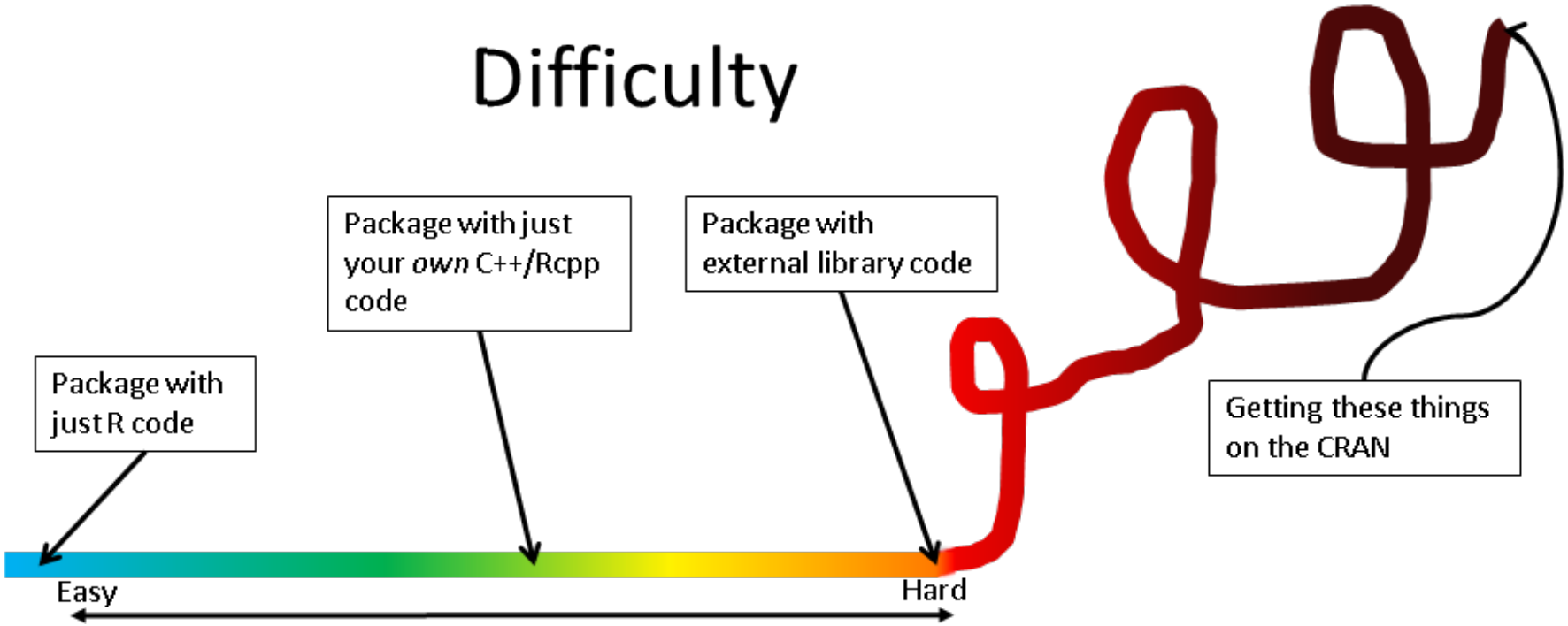
7 Putting Your Rcpp Code in a Package

- General Information
- Package Basics
- Makevars

The CRAN

- We won't be going into CRAN specific issues.
- Getting on the CRAN *can be* annoying.
- Getting on the CRAN with compiled code *is* annoying.
- Getting on the CRAN with Rcpp is somehow even worse.


Difficulty
Package with just your *own* C++/Rcpp code
Package with external library code
Package with just R code
Easy
Hard


Difficulty
Package with just R code
Package with just your *own* C++/Rcpp code
Package with external library code
Getting these things on the CRAN
Easy
Hard


You
Brian Ripley
Your Package

R Packages

- `package.skeleton()` and `Rcpp.package.skeleton()`
- R code goes in `R/` subdirectory
- Compiled code goes in `src/` subdirectory
- Help files go in `man/` subdirectory
- Others: `inst/`, `vignettes/`, `data/`, ...

R Packages

- A dense, but *invaluable* resource:
  http://cran.r-project.org/doc/manuals/R-exts.html
- The **devtools** package has its uses.

Putting Code into a Package

- Put R code in `R/`
- Put compiled code in `src/` and create a Makevars file (and god help you, a configure.ac).
- You can use Cmake, but CRAN offers limited support.

Makevars

- R's strange version of a Makefile
- If you know GNU autotools, you can probably figure things out.
- Generally not too complicated unless you link to external libraries.
- Don't be afraid to look at what other package developers are doing — or even ask them.
- The R-devel list is available, but usual warnings apply.

Makevars or Makevars.in with Rcpp

Makevars and Rcpp

Put these at the top of your Makevars when using Rcpp

```
1 PKG_CXXFLAGS = `$(R_HOME)/bin/Rscript -e
      "Rcpp:::CxxFlags()"`
2 PKG_LIBS = `$(R_HOME)/bin/Rscript -e "Rcpp:::LdFlags()"`
```

8 Wrapup

Other Important Topics Not Discussed Here

- Getting an Rcpp-using package onto the CRAN...
- Embedding R in C (e.g., RInside).
- Bringing Fortran into the mix.
- Rcpp + threads/MPI.
- Choice of compiler.
- If you already know C++, you may be interested in RcppAttributes http://cran.rstudio.com/web/packages/Rcpp/vignettes/Rcpp-attributes.pdf
- Tabs vs Spaces (spaces)

Thanks for coming!

Questions?