ILLINOIS

Introduction to Rcpp:

From Simple Examples to Machine Learning

R/Finance Pre-Conference Tutorial

Dirk Eddelbuettel

17 May 2019

Very Broad Outline

Overview

- Why ?
- How ?

Introduction: Why?

So why Rcpp?

Three Key Reasons

- Speed, Performance, ...
- Do things you could not do before
- Easy to extend R this way

Simple Example

R Version of 'is this number odd or even'

```
isOdd_r <- function(num = 10L) {
   result = (num %% 2L == 1L)
   return(result)
}
isOdd_r(42L)
```

```
## [1] FALSE
```

Simple Example (cont.)

C++ Version of 'is this number odd or even'

```cpp
bool isOdd_cpp(int num = 10) {
    bool result = (num % 2 == 1);
    return result;
}
```

Free-standing code, not yet executable...

Simple Example (cont.)

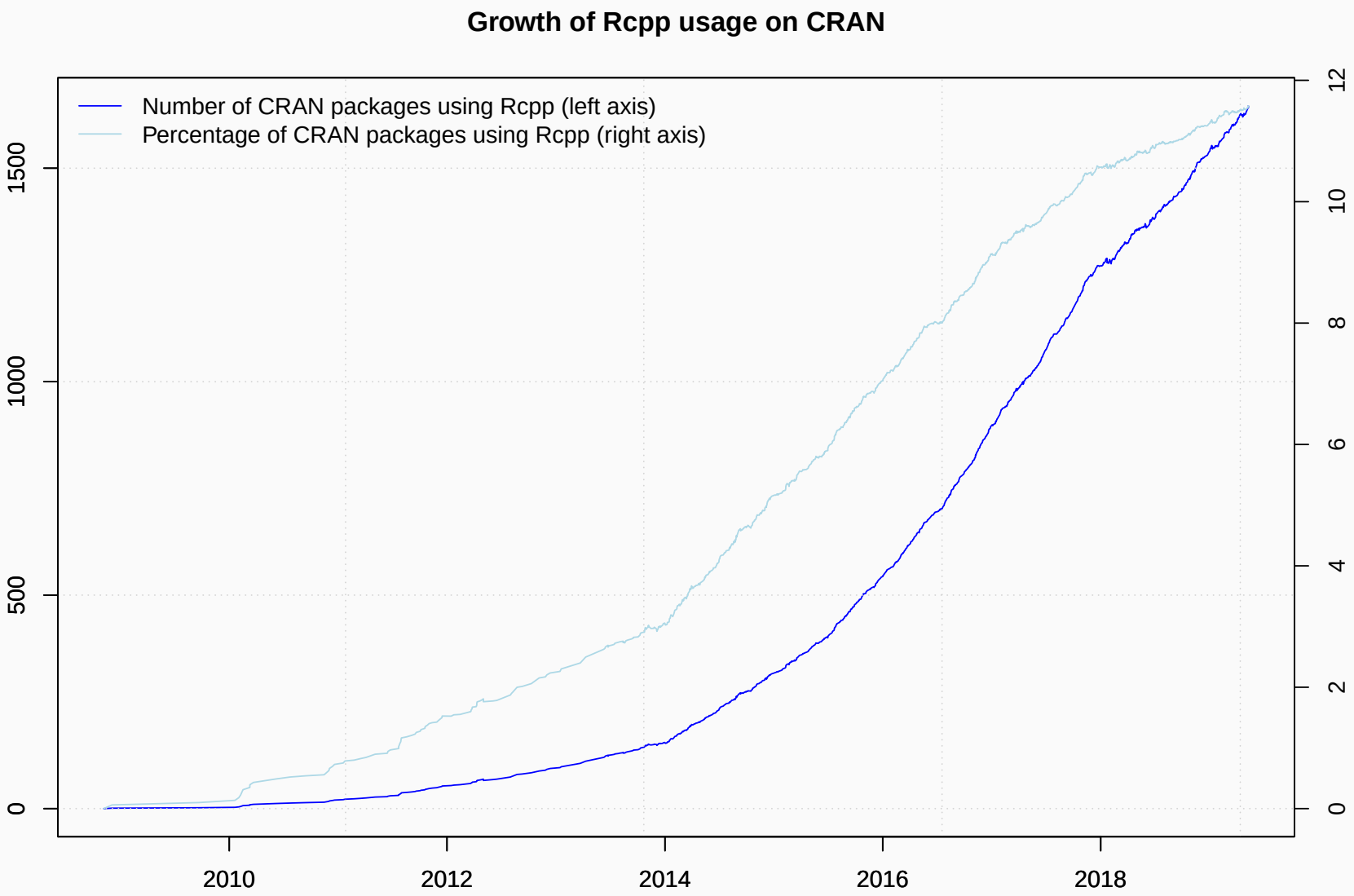
Rcpp Version of 'is this number odd or even'

```
Rcpp::cppFunction("
bool isOdd_cpp(int num = 10) {
   bool result = (num % 2 == 1);
   return result;
}")
isOdd_cpp(42L)
```

```
## [1] FALSE
```

An Aside

Growth



Data current as of May 12, 2019.

Users on Core Repositories

Rcpp is currently used by

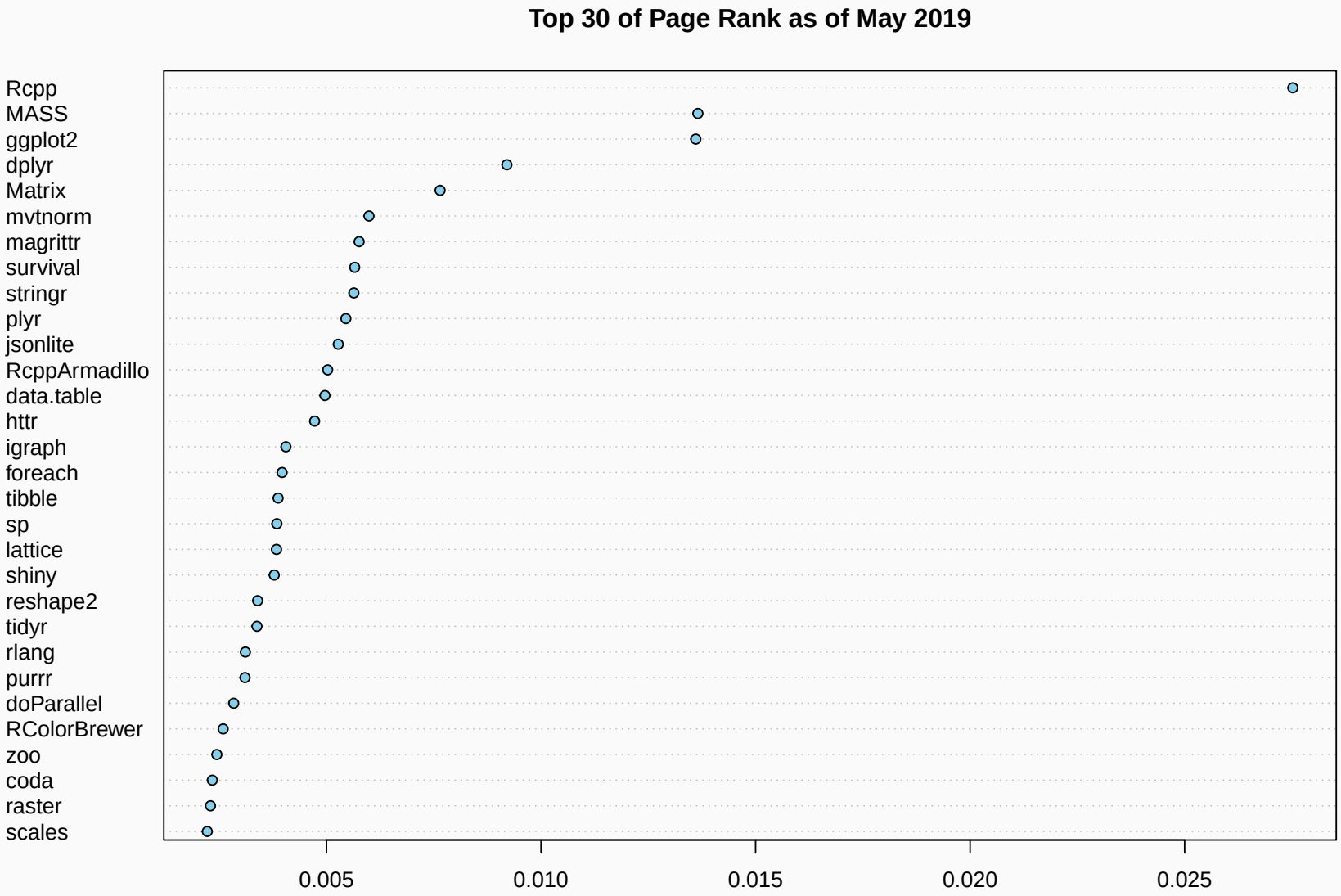
- 1642 CRAN packages (with 278 added since last year)
- 176 BioConductor packages (with 38 added since last year)
- an unknown (but “large”) number of GitHub projects

Pagerank

```
suppressMessages(library(utils))
library(pagerank)   # cf github.com/andrie/pagerank

cran <- "http://cloud.r-project.org"
pr <- compute_pagerank(cran)
round(100*pr[1:5], 3)
```

```
##    Rcpp    MASS ggplot2   dplyr  Matrix
##   2.752   1.365   1.361   0.920   0.765
```

Pagerank

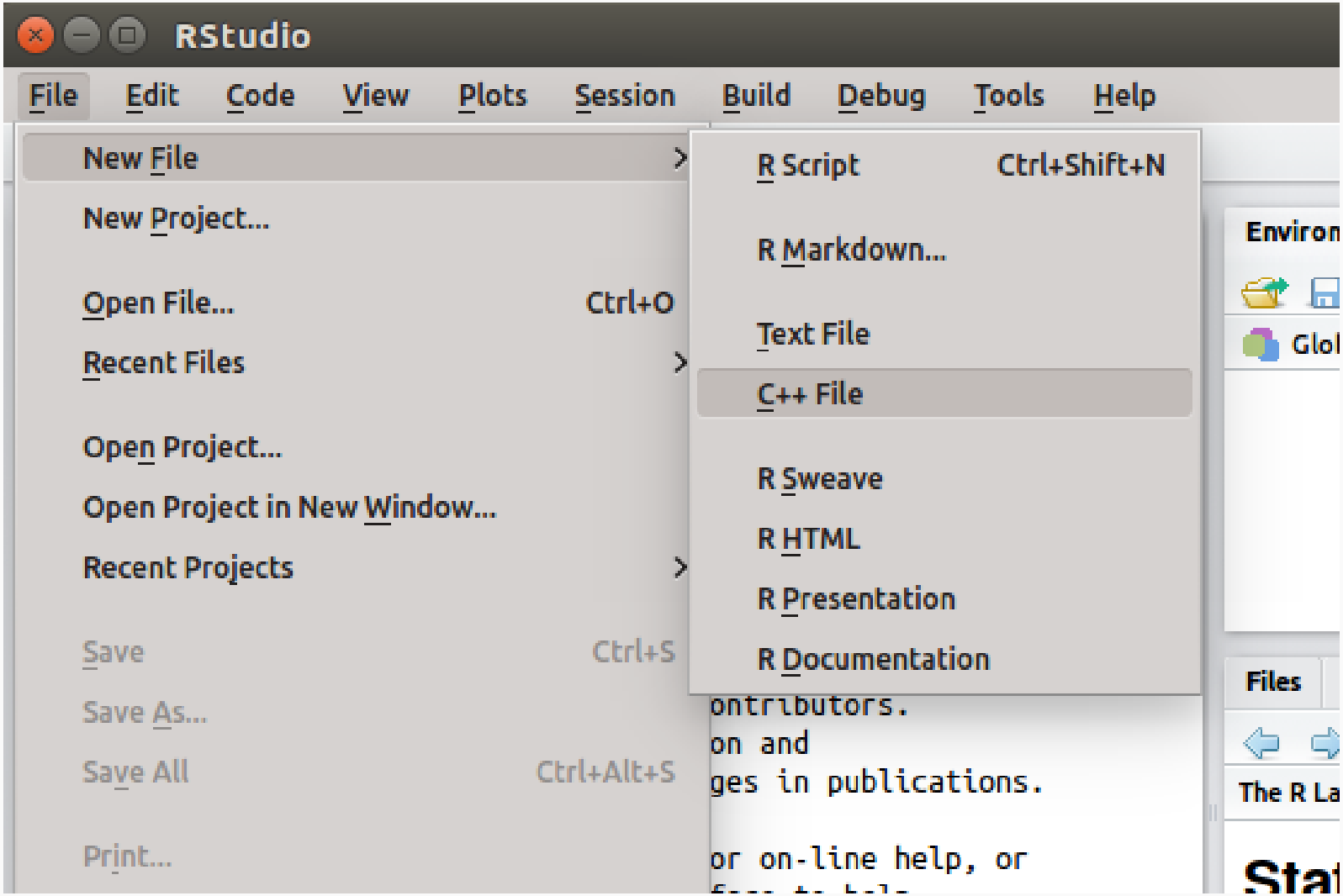

Top 30 of Page Rank as of May 2019
Rcpp
MASS
ggplot2
dplyr
Matrix
mvtnorm
magrittr
survival
stringr
plyr
jsonlite
RcppArmadillo
data.table
httr
igraph
foreach
tibble
sp
lattice
shiny
reshape2
tidyr
rlang
purrr
doParallel
RColorBrewer
zoo
coda
raster
scales
0.005
0.010
0.015
0.020
0.025

Percentage of Compiled Packages

```
db <- tools::CRAN_package_db()   # added in R 3.4.0
## rows: number of pkgs, cols: different attributes
nTot <- nrow(db)
## all direct Rcpp reverse depends, ie packages using Rcpp
nRcpp <- length(tools::dependsOnPkgs("Rcpp", recursive=FALSE,
                                     installed=db))
nCompiled <- table(db[, "NeedsCompilation"])["yes"]
propRcpp <- nRcpp / nCompiled * 100
data.frame(tot=nTot, totRcpp = nRcpp, totCompiled = nCompiled,
           RcppPctOfCompiled = propRcpp)

##     tot totRcpp totCompiled RcppPctOfCompiled
## 1 14208    1642        3573           45.9558
```

Introduction: How?

Jumping Right In: Via RStudio



A First Example: Cont'ed

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

// This is a simple example of exporting a C++ function to R. You can
// source this function into an R session using the Rcpp::sourceCpp
// function (or via the Source button on the editor toolbar). ...

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

// You can include R code blocks in C++ files processed with sourceCpp
// (useful for testing and development). The R code will be automatically
// run after the compilation.

/*** R
timesTwo(42)
*/
```

A First Example: Cont'ed

So what just happened?

- We defined a simple C++ function
- It operates on a numeric vector argument
- We ask Rcpp to 'source it' for us
- Behind the scenes Rcpp creates a wrapper
- Rcpp then compiles, links, and loads the wrapper
- The function is available in R under its C++ name

Another Example: Focus on Speed

Consider a function defined as

$$f(n) \quad \text{such that} \quad \begin{cases} n & \text{when } n < 2 \\ f(n-1) + f(n-2) & \text{when } n \geq 2 \end{cases}$$

An Introductory Example: Simple R Implementation

R implementation and use:

```
f <- function(n) {
    if (n < 2) return(n)
    return(f(n-1) + f(n-2))
}

## Using it on first 11 arguments
sapply(0:10, f)
```

```
##  [1]  0  1  1  2  3  5  8 13 21 34 55
```

An Introductory Example: Timing R Implementation

Timing:

```
library(rbenchmark)
benchmark(f(10), f(15), f(20))[,1:4]
```

```
##    test replications elapsed relative
## 1 f(10)          100   0.008    1.000
## 2 f(15)          100   0.080   10.000
## 3 f(20)          100   0.873  109.125
```

An Introductory Example: C++ Implementation

```
int g(int n) {
    if (n < 2) return(n);
    return(g(n-1) + g(n-2));
}
```

deployed as

```
Rcpp::cppFunction('int g(int n) {
   if (n < 2) return(n);
   return(g(n-1) + g(n-2)); }')
## Using it on first 11 arguments
sapply(0:10, g)
```

```
##  [1]  0  1  1  2  3  5  8 13 21 34 55
```

An Introductory Example: Comparing timing

Timing:

```
library(rbenchmark)
benchmark(f(20), g(20))[,1:4]
```

```
##    test replications elapsed relative
## 1 f(20)          100   0.849    424.5
## 2 g(20)          100   0.002      1.0
```

A nice gain of a few orders of magnitude.

Some Background

R Type mapping

Standard R types (integer, numeric, list, function, ... and compound objects) are mapped to corresponding C++ types using extensive template meta-programming – it just works:

```
library(Rcpp)
cppFunction("NumericVector la(NumericVector x){
  return log(abs(x));
}")
la(seq(-5, 5, by=2))
```

Also note: vectorized C++! `log(abs())` on vectors as R would.

STL Type mapping

Use of `std::vector<double>` and STL algorithms:

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

inline double f(double x) { return ::log(::fabs(x)); }

// [[Rcpp::export]]
std::vector<double> logabs2(std::vector<double> x) {
  std::transform(x.begin(), x.end(), x.begin(),  f);
  return x;
}
```

Not vectorized but `std::transform()` 'sweeps' `f()` across.

STL Type mapping

Used via

```
library(Rcpp)
sourceCpp("code/logabs2.cpp")
logabs2(seq(-5, 5, by=2))
```

Type mapping is seamless

Simple outer product of a col. vector (using RcppArmadillo):

```
library(Rcpp)
cppFunction("arma::mat v(arma::colvec a) {
              return a*a.t();}",
            depends="RcppArmadillo")
v(1:3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    2    4    6
## [3,]    3    6    9
```

Uses implicit conversion via `as<>` and `wrap` – cf vignette Rcpp-extending.

C++11: lambdas, auto, and much more

We can simplify the `log(abs(...))` example further:

```
#include <Rcpp.h>
// [[Rcpp::plugins(cpp11)]]

using namespace Rcpp;

// [[Rcpp::export]]
std::vector<double> logabs3(std::vector<double> x) {
   std::transform(x.begin(), x.end(), x.begin(),
                  [](double x) {
                    return ::log(::fabs(x));
                  } );
   return x;
}
```

How to: Main Usage Patterns

Basic Usage: evalCpp()

`evalCpp()` evaluates a single C++ expression. Includes and dependencies can be declared.

This allows us to quickly check C++ constructs.

```
library(Rcpp)
evalCpp("2 + 2")      # simple test
```

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

```
evalCpp("std::numeric_limits<double>::max()")
```

```
## [1] 1.79769e+308
```

Basic Usage: cppFunction()

`cppFunction()` creates, compiles and links a C++ file, and creates an R function to access it.

```
cppFunction("
    int exampleCpp11() {
        auto x = 10;
        return x;
}", plugins=c("cpp11"))
exampleCpp11()  # same identifier as C++ function
```

Basic Usage: sourceCpp()

`sourceCpp()` is the actual workhorse behind `evalCpp()` and `cppFunction()`. It is described in more detail in the package vignette Rcpp-attributes.

`sourceCpp()` builds on and extends `cxxfunction()` from package inline, but provides even more ease-of-use, control and helpers – freeing us from boilerplate scaffolding.

A key feature are the plugins and dependency options: other packages can provide a plugin to supply require compile-time parameters (cf RcppArmadillo, RcppEigen, RcppGSL).

Basic Usage: Packages

Package are *the* standard unit of R code organization.

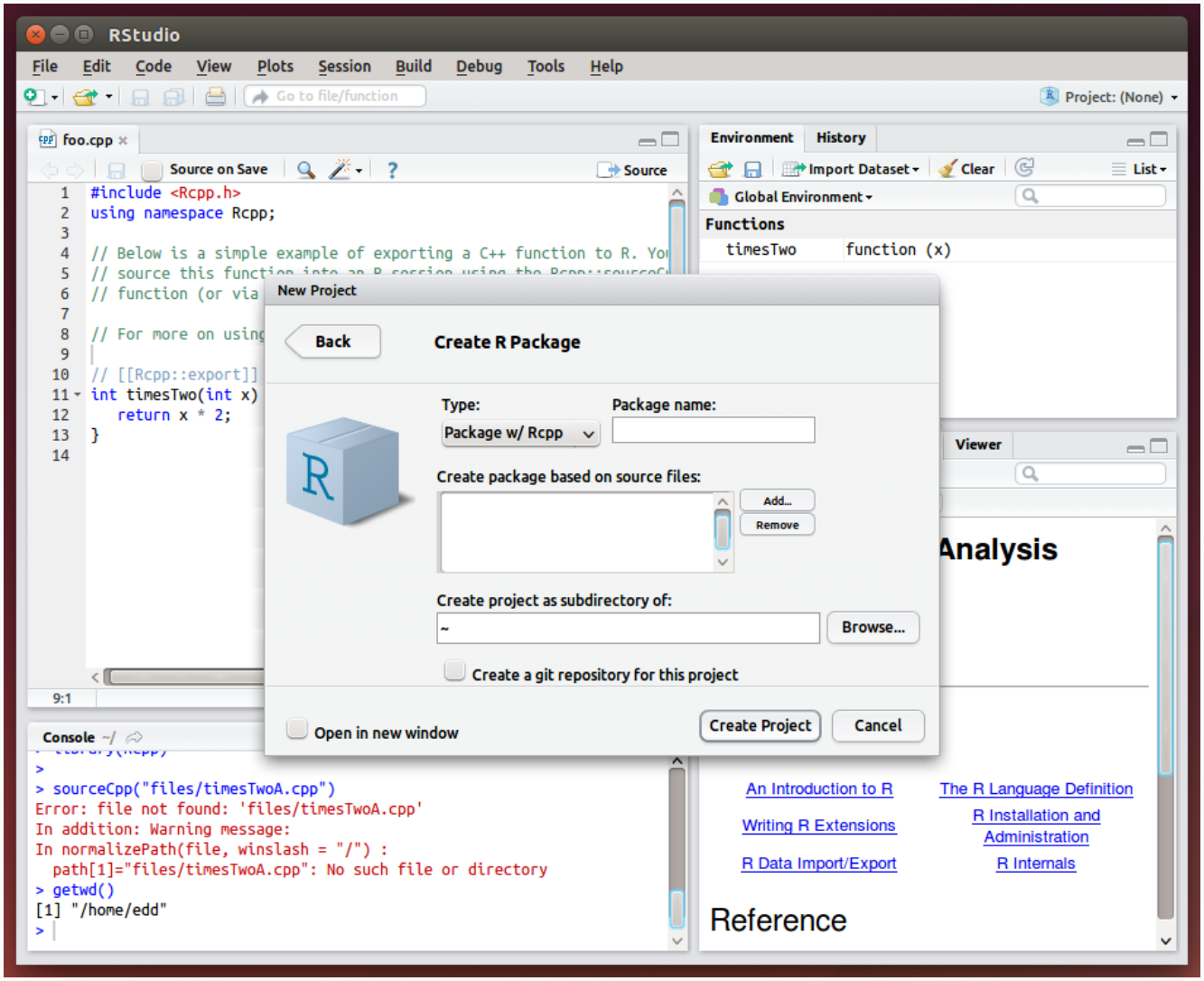
Creating packages with Rcpp is easy; an empty one to work from can be created by `Rcpp.package.skeleton()`

The vignette Rcpp-packages has fuller details.

As of May 2019, there are 1642 CRAN and 176 BioConductor packages which use Rcpp all offering working, tested, and reviewed examples.

Packages and Rcpp

Best way to organize R code with Rcpp is via a package:



Packages and Rcpp

`Rcpp.package.skeleton()` and its derivatives. e.g.

`RcppArmadillo.package.skeleton()` create working packages.

```
// another simple example: outer product of a vector,
// returning a matrix
//
// [[Rcpp::export]]
arma::mat rcpparma_outerproduct(const arma::colvec & x) {
    arma::mat m = x * x.t();
    return m;
}

// and the inner product returns a scalar
//
// [[Rcpp::export]]
double rcpparma_innerproduct(const arma::colvec & x) {
    double v = arma::as_scalar(x.t() * x);
    return v;
}
```

Two (or three) ways to link to external libraries

- *Full copies:* Do what RcppMLPACK (v1) does and embed a full copy; larger build time, harder to update, self-contained
- *With linking of libraries:* Do what RcppGSL or RcppMLPACK (v2) do and use hooks in the package startup to store compiler and linker flags which are passed to environment variables
- *With C++ template headers only:* Do what RcppArmadillo and other do and just point to the headers

More details in extra vignettes.

Machine Learning

Overview

Among the 1600+ CRAN packages using Rcpp, several wrap Machine Learning libraries.

Here are three:

- RcppShark based on Shark (but archived in March 2018)
- RcppMLPACK based on MLPACK (using older code base embedded)
- dlib based on DLib

MLPACK

High-level:

- Written by Ryan Curtin et al, Georgia Tech
- Uses Armadillo, and like Armadillo, “feels right”
- Qiang Kou created ‘RcppMLPACK v1’, it is on CRAN

MLPACK ‘v2’

High-level:

- A few of us are trying to update RcppMLPACK to ‘v2’
- Instead of embedding, an external library is used
- This makes deployment a little tricker on Windows and macOS
- We are still waiting on macOS installation of libraries

List of Algorithms:

- Collaborative filtering (with many decomposition techniques)
- Decision stumps (one-level decision trees)
- Density estimation trees
- Euclidean minimum spanning tree calculation
- Gaussian mixture models
- Hidden Markov models
- Kernel Principal Components Analysis (optionally with sampling)
- k-Means clustering (with several accelerated algorithms)
- Least-angle regression (LARS/LASSO)
- Linear regression (simple least-squares)
- Local coordinate coding
- Locality-sensitive hashing for approximate nearest neighbor search
- Logistic regression
- Max-kernel search
- Naive Bayes classifier
- Nearest neighbor search with dual-tree algorithms
- Neighborhood components analysis
- Non-negative matrix factorization
- Perceptrons
- Principal components analysis (PCA)
- RADICAL (independent components analysis)
- Range search with dual-tree algorithms
- Rank-approximate nearest neighbor search
- Sparse coding with dictionary learning

RcppMLPACK: K-Means Example

```
#include "RcppMLPACK.h"

using namespace mlpack::kmeans;
using namespace Rcpp;

// [[Rcpp::depends(RcppMLPACK)]]

// [[Rcpp::export]]
List cppKmeans(const arma::mat& data, const int& clusters) {

    arma::Col<size_t> assignments;
    KMeans<> k;     // Initialize with the default arguments.
    k.Cluster(data, clusters, assignments);

    return List::create(Named("clusters") = clusters,
                        Named("result")   = assignments);
}
```

Timing

Table 1: Benchmarking result

test	replications	elapsed	relative	user.self	sys.self
mlKmeans(t(wine), 3)	100	0.028	1.000	0.028	0.000
kmeans(wine, 3)	100	0.947	33.821	0.484	0.424

Table taken 'as is' from RcppMLPACK vignette.

RcppMLPACK: Linear Regression Example

```
// [[Rcpp::depends(RcppMLPACK)]]
// [[Rcpp::plugins(openmp)]]
#include <RcppMLPACK.h>                    // MLPACK, Rcpp and RcppArmadillo

// particular algorithm used here
#include <mlpack/methods/linear_regression/linear_regression.hpp>

// [[Rcpp::export]]
arma::vec linearRegression(arma::mat& matX,
                           arma::vec& vecY,
                           const double lambda = 0.0,
                           const bool intercept = true) {

    matX = matX.t();
    mlpack::regression::LinearRegression lr(matX, vecY.t(), lambda, intercept);
    arma::rowvec fittedValues(vecY.n_elem);
    lr.Predict(matX,  fittedValues);
    return fittedValues.t();
}
```

RcppMLPACK: Linear Regression Example

```
suppressMessages(library(utils))
library(RcppMLPACK)
data("trees", package="datasets")
X <- with(trees, cbind(log(Girth), log(Height)))
y <- with(trees, log(Volume))
lmfit <- lm(y ~ X)
# summary(fitted(lmfit))

mlfit <- linearRegression(X, y)
# summary(mlfit)

all.equal(unname(fitted(lmfit)),  as.vector(mlfit))
```

```
## [1] TRUE
```

RcppMLPACK: Logistic Regression Example

```cpp
#include <RcppMLPACK.h>             // MLPACK, Rcpp and RcppArmadillo
#include <mlpack/methods/logistic_regression/logistic_regression.hpp>   // algo use here

// [[Rcpp::export]]
Rcpp::List logisticRegression(const arma::mat& train, const arma::irowvec& labels,
                              const Rcpp::Nullable<Rcpp::NumericMatrix>& test = R_NilValue) {

    // MLPACK wants Row<size_t> which is an unsigned representation that R does not have
    arma::Row<size_t> labelsur, resultsur;

    // TODO: check that all values are non-negative
    labelsur = arma::conv_to<arma::Row<size_t>>::from(labels);

    // Initialize with the default arguments. TODO: support more arguments>
    mlpack::regression::LogisticRegression<> lrc(train, labelsur);
    arma::rowvec parameters = lrc.Parameters();

    Rcpp::List return_val;
    if (test.isNotNull()) {
        arma::mat test2 = Rcpp::as<arma::mat>(test);
        lrc.Classify(test2, resultsur);
        arma::vec results = arma::conv_to<arma::vec>::from(resultsur);
        return_val = Rcpp::List::create(Rcpp::Named("parameters") = parameters,
                                        Rcpp::Named("results") = results);
    } else {
        return_val = Rcpp::List::create(Rcpp::Named("parameters") = parameters);
    }
    return return_val;
}
```

RcppMLPACK: Linear Regression Example

```
suppressMessages(library(utils))
library(RcppMLPACK)
example(logisticRegression)
```

```
##
## lgstcRR> data(trainSet)
##
## lgstcRR> mat <- t(trainSet[, -5])     ## train data, transpose and removing class labels
##
## lgstcRR> lab <- trainSet[, 5]         ## class labels for train set
##
## lgstcRR> logisticRegression(mat, lab)
## $parameters
## [1] -11.081991  13.902248   0.803497  -9.348522 -13.086997
##
##
## lgstcRR> testMat <- t(testSet[, -5])  ## test data
##
## lgstcRR> logisticRegression(mat, lab, testMat)
## $parameters
## [1] -11.081991  13.902248   0.803497  -9.348522 -13.086997
##
## $results
## [1] 0 0 0 1 1 1 1
```

RcppMLPACK: Nearest Neighbors Example

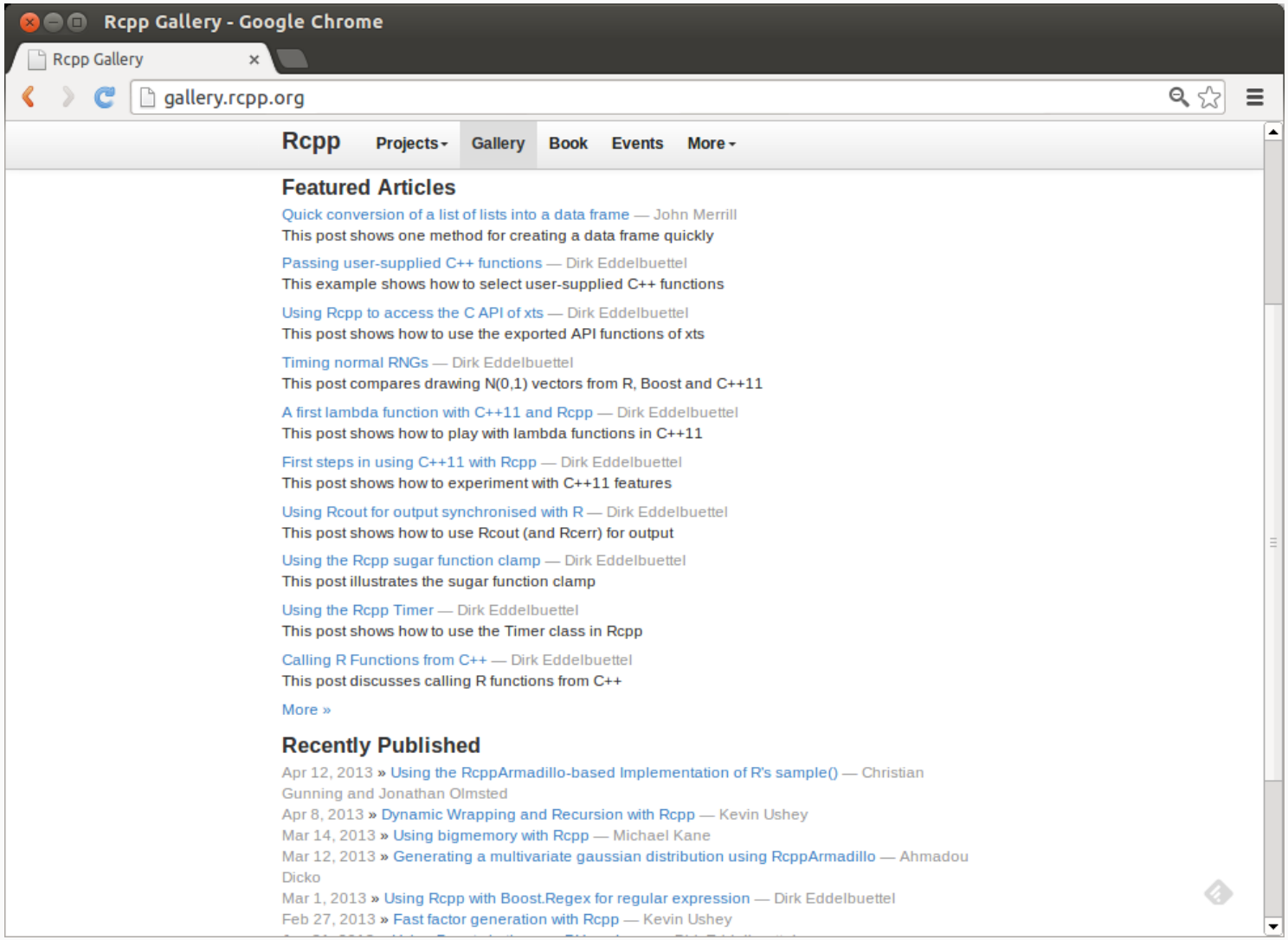
```
#include "RcppMLPACK.h"

using namespace Rcpp;
using namespace mlpack;           using namespace mlpack::neighbor;
using namespace mlpack::metric;   using namespace mlpack::tree;

// [[Rcpp::depends(RcppMLPACK)]]
// [[Rcpp::export]]
List nn(const arma::mat& data, const int k) {
    // using a test from MLPACK 1.0.10 file src/mlpack/tests/allknn_test.cpp
    CoverTree<LMetric<2>, FirstPointIsRoot,
              NeighborSearchStat<NearestNeighborSort> > tree =
        CoverTree<LMetric<2>, FirstPointIsRoot,
                  NeighborSearchStat<NearestNeighborSort> >(data);

    NeighborSearch<NearestNeighborSort, LMetric<2>,
                   CoverTree<LMetric<2>, FirstPointIsRoot,
                             NeighborSearchStat<NearestNeighborSort> > >
        coverTreeSearch(&tree, data, true);

    arma::Mat<size_t> coverTreeNeighbors;
    arma::mat coverTreeDistances;
    coverTreeSearch.Search(k, coverTreeNeighbors, coverTreeDistances);

    return List::create(Named("clusters") = coverTreeNeighbors,
                        Named("result")   = coverTreeDistances);
}
```

More

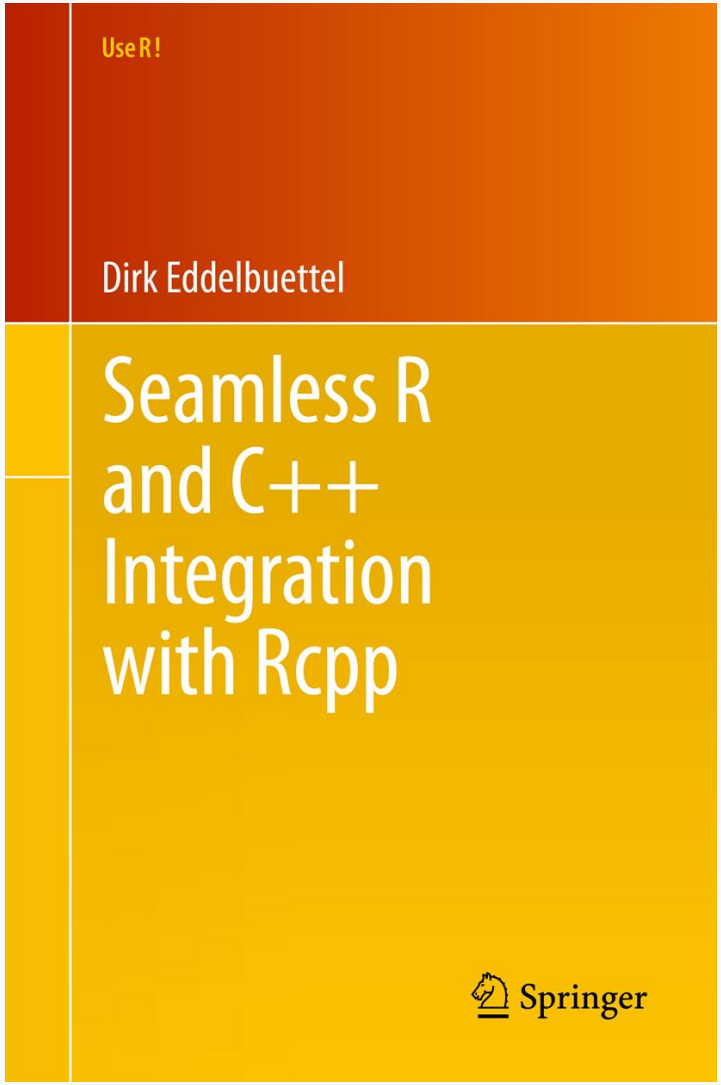
Documentation

- The package comes with nine pdf vignettes, and help pages.
- The introductory vignettes are now published (Rcpp and RcppEigen in *J Stat Software*, RcppArmadillo in *Comp Stat & Data Anlys*, Rcpp again in *TAS*)
- The rcpp-devel list is *the* recommended resource, generally very helpful, and fairly low volume.
- StackOverflow has a fair number of posts too.
- And a number of blog posts introduce/discuss features.

Rcpp Gallery



The Rcpp book



On sale since June 2013.

Thank you!

slides http://dirk.eddelbuettel.com/presentations/

web http://dirk.eddelbuettel.com/

mail dirk@eddelbuettel.com

github @eddelbuettel

twitter @eddelbuettel

Appendix: More Rcpp Examples

Cumulative Sum: vector-cumulative-sum

A basic looped version:

```
#include <Rcpp.h>
#include <numeric>      // for std::partial_sum
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector cumsum1(NumericVector x){
    double acc = 0;     // init an accumulator variable

    NumericVector res(x.size());  // init result vector

    for(int i = 0; i < x.size(); i++){
         acc += x[i];
         res[i] = acc;
    }
    return res;
}
```

Cumulative Sum: vector-cumulative-sum

An STL variant:

```
// [[Rcpp::export]]
NumericVector cumsum2(NumericVector x){
    // initialize the result vector
    NumericVector res(x.size());
    std::partial_sum(x.begin(), x.end(), res.begin());
    return res;
}
```

Cumulative Sum: vector-cumulative-sum

Or just Rcpp sugar:

```
// [[Rcpp::export]]
NumericVector cumsum_sug(NumericVector x){
    return cumsum(x);  // compute + return result vector
}
```

Of course, all results are the same.

R Function Call from C++: r-function-from-c++

```
#include <Rcpp.h>

using namespace Rcpp;

// [[Rcpp::export]]
NumericVector callFunction(NumericVector x,
                           Function f) {
    NumericVector res = f(x);
    return res;
}

/*** R
callFunction(x, fivenum)
*/
```

Using Boost via BH: using-boost-with-bh

```
// [[Rcpp::depends(BH)]]
#include <Rcpp.h>

// One include file from Boost
#include <boost/date_time/gregorian/gregorian_types.hpp>

using namespace boost::gregorian;

// [[Rcpp::export]]
Rcpp::Date getIMMDate(int mon, int year) {
    // compute third Wednesday of given month / year
    date d = nth_day_of_the_week_in_month(
                        nth_day_of_the_week_in_month::third,
                        Wednesday, mon).get_date(year);
    date::ymd_type ymd = d.year_month_day();
    return Rcpp::wrap(Rcpp::Date(ymd.year, ymd.month, ymd.day));
}
```

Using Boost via BH: using-boost-with-bh

```
#include <Rcpp.h>
#include <boost/foreach.hpp>
using namespace Rcpp;
// [[Rcpp::depends(BH)]]

// the C-style upper-case macro name is a bit ugly
#define foreach BOOST_FOREACH

// [[Rcpp::export]]
NumericVector square( NumericVector x ) {

  // elem is a reference to each element in x
  // we can re-assign to these elements as well
  foreach( double& elem, x ) {
    elem = elem*elem;
  }
  return x;
}
```

C++11 now has something similar in a smarter `for` loop.

Vector Subsetting: subsetting

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

// [[Rcpp::export]]
NumericVector positives(NumericVector x) {
    return x[x > 0];
}

// [[Rcpp::export]]
List first_three(List x) {
    IntegerVector idx = IntegerVector::create(0, 1, 2);
    return x[idx];
}

// [[Rcpp::export]]
List with_names(List x, CharacterVector y) {
    return x[y];
}
```

Armadillo Eigenvalues: armadillo-eigenvalues

```
#include <RcppArmadillo.h>

// [[Rcpp::depends(RcppArmadillo)]]

// [[Rcpp::export]]
arma::vec getEigenValues(arma::mat M) {
    return arma::eig_sym(M);
}
```

Armadillo Eigenvalues: armadillo-eigenvalues

```
sourceCpp("code/armaeigen.cpp")

set.seed(42)
X <- matrix(rnorm(4*4), 4, 4)
Z <- X %*% t(X)
getEigenValues(Z)
```

```
##           [,1]
## [1,]  0.331887
## [2,]  1.685588
## [3,]  2.409920
## [4,] 14.210011
```

```
# R gets the same results (in reverse)
# and also returns the eigenvectors.
```

Create xts from in C++: creating-xts-from-c++

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

NumericVector createXts(int sv, int ev) {
    IntegerVector ind = seq(sv, ev);     // values

    NumericVector dv(ind);               // date(time)s == reals
    dv = dv * 86400;                     // scaled to days
    dv.attr("tzone")    = "UTC";         // index has attributes
    dv.attr("tclass")   = "Date";

    NumericVector xv(ind);               // data has same index
    xv.attr("dim")         = IntegerVector::create(ev-sv+1,1);
    xv.attr("index")       = dv;
    CharacterVector cls = CharacterVector::create("xts","zoo");
    xv.attr("class")       = cls;
    xv.attr(".indexCLASS") = "Date";
    // ... some more attributes ...

    return xv;
}
```

RcppParallel 1/3: parallel-matrix-transform

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

#include <cmath>
#include <algorithm>

// [[Rcpp::export]]
NumericMatrix matrixSqrt(NumericMatrix orig) {

  // allocate the matrix we will return
  NumericMatrix mat(orig.nrow(), orig.ncol());

  // transform it
  std::transform(orig.begin(), orig.end(), mat.begin(), ::sqrt);

  // return the new matrix
  return mat;
}
```

RcppParallel 2/3: parallel-matrix-transform

```
// [[Rcpp::depends(RcppParallel)]]
#include <RcppParallel.h>
using namespace RcppParallel;

struct SquareRoot : public Worker {

   const RMatrix<double> input;   // source matrix
   RMatrix<double> output;        // destination matrix

   // initialize with source and destination
   SquareRoot(const NumericMatrix input, NumericMatrix output)
      : input(input), output(output) {}

   // take the square root of the range of elements requested
   void operator()(std::size_t begin, std::size_t end) {
      std::transform(input.begin() + begin,
                     input.begin() + end,
                     output.begin() + begin,
                     ::sqrt);
   }
};
```

RcppParallel 3/3: parallel-matrix-transform

```
// [[Rcpp::export]]
NumericMatrix parallelMatrixSqrt(NumericMatrix x) {

  // allocate the output matrix
  NumericMatrix output(x.nrow(), x.ncol());

  // SquareRoot functor (pass input and output matrixes)
  SquareRoot squareRoot(x, output);

  // call parallelFor to do the work
  parallelFor(0, x.length(), squareRoot);

  // return the output matrix
  return output;
}
```