Programming with Data: Using and extending R

Dirk Eddelbuettel, Ph.D.
edd@debian.org, Dirk.Eddelbuettel@R-Project.org

Invited Guest Lecture
ACM Student Chapter
University of Chicago
18 February 2010
What is a key motivation?

If you are looking for a career where your services will be in high demand, you should find something where you provide a scarce, complementary service to something that is getting ubiquitous and cheap.

So what's getting ubiquitous and cheap? Data.

And what is complementary to data? Analysis.

And more:

I keep saying the sexy job in the next ten years will be statisticians. People think I’m joking, but who would’ve guessed that computer engineers would’ve been the sexy job of the 1990s?

I think statisticians are part of it, but it’s just a part. You also want to be able to visualize the data, communicate the data, and utilize it effectively.

R is a dialect of S which was started at Bell Labs in 1975

S won the 1998 ACM Software Systems award. Citing: *will forever alter the way people analyze, visualize, and manipulate data . . . S is an elegant, widely accepted, and enduring software system, with conceptual integrity, thanks to the insight, taste, and effort of John Chambers.* Other winners: Unix, TeX, TCP/IP, Web, Apache, Make . . .

R was started by R Ihaka and R Gentleman in the early 1990s, has been a GNU project since 1997; and is now maintained by a core group of academic statisticians / computer scientists
R History by the books


Statistical modeling in S; S3 version of classes and methods.


Version 4 of S, a major revision of S. The R version of S4 and other R techniques.


Earliest publication of what became S.

Introduced what is now known as S version 3 (S3).

Thanks to John Chambers for sending me high-resolution scans of the covers of his books.

Dirk Eddelbuettel

*Programming with Data: Using and extending R*
R has now become mainstream

The New York Times notices

Dice calls it a job market edge

Data Analysts Captivated by R’s Power

R first appeared in 2005, when the statistics professors Robert Gentleman, left, and Ross Ihaka released the code as a free software package.

By ASHLEE VANCE

Published January 6, 2009

To some people R is just the 18th letter of the alphabet. To others, it's the rating on racy movies, a measure of an attic's insulation or what pirates in movies say.

R is also the name of a popular programming language used by a growing number of data analysts inside corporations and...
So what is R?

A nice answer was provided in a recent presentation by David Smith of REvolution Computing\(^1\)

- Data analysis software and environment
- A programming language designed by and for statisticians
- An (interactive) environment with a huge library of algorithms for data access, data manipulation, analysis and graphics
- An open-source software project: free, open and active
- A community: thousands of contributors, estimated 2 million users, resources and help in every problem domain

\(^1\) Videos at [http://www.youtube.com/watch?v=M2u7kbcXI_k](http://www.youtube.com/watch?v=M2u7kbcXI_k)
Seven awesome things about R

David Smith expands on his answer with this:

- **Free**: Open Source / GPL; Flexible; Easily integrated; Broad user-base
- **Language**: Programming, not dialogs or cell formulas; Designed for data analysis: vector, matrix, model, ...; built-in library of algorithms; development speed
- **Graphics and Visualizations**: Standard graphs (scatter, smoothing, dot chart, image plot, surfaces, ...); Influences by Tufte and Cleveland; Infinitely customizable
- **Statistics**: All standard methods built in (regression, nonlinear modelling, mixed-effects, gam, trees, ...)
- **Cutting-edge analytics**: Excellent domain-specific support (BioConductor; Rmetrics); 2000+ add-on packages at CRAN
- **Community**: Very active lists; web resources (see below)
- **No limits**: Open, powerful, mashable, flexible, fun!
Web Dashboard Examples using RApache

- Slick RApache demo using public baseball data
- ’Four-dimensional’ visualization of pitch placement, speed and frequency
- Newer examples by Jeroen Ooms at http://www.jeroon.net

Source: http://labs.dataspora.com/gameday/
Another Web-based Example: Google/R Mashups

Source: http://eis.sfei.org/wqt/
An Example related to SVN Commits

All it takes are these few lines of code:

```r
commits <- lapply(2006:2009, function(y) {
    u <- paste("http://developer.r-project.org/R.svnlog.",
                as.character(y), sep="")
    x <- readLines(u)
    rx <- x[,grep("^[0-9]{5} \|",x)]
    who <- gsub(" \|", sapply(strsplit(rx,"\|"),"\[",2))
})
ctab <- table(do.call(c, commits))
library(lattice)
dotplot(ctab[order(ctab)], scales=list(x=list(log = TRUE)),
       xlab="", main="Number of SVN commits since 2006")
```
More about R – and R repositories

A few key points:

- **R** is now a de-facto standard for statistical applications and research
- “Success has many fathers”: several key drivers can be identified as to why R has done so well
- We would like to stress *repositories* and available packages here: CRAN, as well as BioConductor and Omegahat.
- CRAN has been a key driver: an open yet rigorously QA’ed repository which has experienced tremendous growth
- CRAN Task View, CRANberries, CRANtastic can help navigate CRAN.
Illustration: Growth of CRAN packages

- CRAN archive network growing by 40% p.a., now at around 2150 packages
- John Fox provided this chart in an invited lecture at the useR! 2008 meetings.
- Details, and more metrics on R and the dynamics of the R Core group, are also in a recent R Journal article.

Source: Fox (2008, 2010), our calculations
Getting started with R

- `sudo apt-get install r-base` on Debian / Ubuntu
- `run apt-cache search r- | grep ^r-c | sort`
- Visit the R website, the CRAN and R-Forge sites for packages
- Use CRANberries and CRANtastic for package information
- Visit the R Graph Gallery and the R Graphical Manual for visual inspiration
- Use RSeek for searches and the R Wiki
- Browse PlanetR, R Bloggers and the REvo blog
Outline

1. Intro
2. S/R
3. Extending
4. Rcpp
5. Rcpp Usage Examples
6. Summary
R offers several functions to access compiled code: we focus on `.C` and `.Call` here. (R Extensions, sections 5.2 and 5.9; Software for Data Analysis).

The canonical example is the convolution function:

```c
void convolve(double *a, int *na, double *b,
               int *nb, double *ab)
{
  int i, j, nab = *na + *nb - 1;

  for(i = 0; i < nab; i++)
    ab[i] = 0.0;
  for(i = 0; i < *na; i++)
    for(j = 0; j < *nb; j++)
      ab[i + j] += a[i] * b[j];
}
```
The convolution function is called from R by

```r
conv <- function(a, b)
  .C("convolve",
     as.double(a),
     as.integer(length(a)),
     as.double(b),
     as.integer(length(b)),
     ab = double(length(a) + length(b) - 1))$ab
```

As stated in the manual, one must take care to coerce all the arguments to the correct R storage mode before calling `.C` as mistakes in matching the types can lead to wrong results or hard-to-catch errors.
Using `.Call`, the example becomes

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

extern "C" SEXP convolve2(SEXP a, SEXP b)
{
  int i, j, na, nb, nab;
  double *xa, *xb, *xab;
  SEXP ab;

  PROTECT(a = AS_NUMERIC(a));
  PROTECT(b = AS_NUMERIC(b));
  na = LENGTH(a); nb = LENGTH(b); nab = na + nb - 1;
  PROTECT(ab = NEW_NUMERIC(nab));
  xa = NUMERIC_POINTER(a); xb = NUMERIC_POINTER(b);
  xab = NUMERIC_POINTER(ab);

  for(i = 0; i < nab; i++) xab[i] = 0.0;
  for(i = 0; i < na; i++)
    for(j = 0; j < nb; j++) xab[i + j] += xa[i] * xb[j];

  UNPROTECT(3);
  return(ab);
}
```

Dirk Eddelbuettel
Programming with Data: Using and extending R
Now the call becomes easier by just using the function name and the vector arguments—all other handling is done at the C/C++ level:

```r
conv <- function(a, b) .Call("convolve2", a, b)
```

In summary, we see that
- there are different entry points
- using different calling conventions
- leading to code that may need to do more work at the lower level.
inline is a package by Oleg Sklyar et al that provides the function cfunction which can wrap Fortran, C or C++ code.

```r
## A simple Fortran example
code <- "
    integer i
    do 1 i=1, n(1)
    1 x(i) = x(i)**3
"
cubefn <- cfunction(signature(n="integer", x="numeric"),
                    code, convention=".Fortran")
x <- as.numeric(1:10)
n <- as.integer(10)
cubefn(n, x)$x
```

cfunction takes care of compiling, linking, loading, ... by placing the resulting dynamically-loadable object code in the per-session temporary directory used by R.

Rcpp makes it easier to interface C++ and R code.

Using the \texttt{.Call} interface, we can use features of the C++ language to automate the tedious bits of the macro-based C-level interface to R.

One major advantage of using \texttt{.Call} is that richer R objects (vectors, matrices, lists, ...) can be passed directly between R and C++ without the need for explicit passing of dimension arguments. And by using the C++ class layers, we do not need to directly manipulate the SEXP objects.
The convolution example can be rewritten as follows in the 'Classic API':

```cpp
#include <Rcpp.h>

RcppExport SEXP convolve_cpp(SEXP a, SEXP b) {
  RcppVector<double> xa(a);
  RcppVector<double> xb(b);

  int nab = xa.size() + xb.size() - 1;

  RcppVector<double> xab(nab);
  for (int i = 0; i < nab; i++) xab(i) = 0.0;

  for (int i = 0; i < xa.size(); i++)
    for (int j = 0; j < xb.size(); j++)
      xab(i + j) += xa(i) * xb(j);

  RcppResultSet rs;
  rs.add("ab", xab);
  return rs.getReturnList();
}
```
Rcpp was significantly extended over the last few months to permit more natural expressions. Consider this comparison between the R API and the new Rcpp API:

```
1 SEXP ab;
2 PROTECT(ab = allocVector(STRSXP, 2));
3 SET_STRING_ELT(ab, 0, mkChar("foo"));
4 SET_STRING_ELT(ab, 1, mkChar("bar"));
5 UNPROTECT(1);
```

```
1 CharacterVector ab(2);
2 ab[0] = "foo";
3 ab[1] = "bar";
```

Data types, including STL containers and iterators, can be nested and other niceties. Implicit converters allow us to combine types:

```
1 std::vector<double> vec;
2 [...]  
3 List x(3);
4 x[0] = vec;
5 x[1] = "some text";
6 x[2] = 42;
```
In R, functional programming is easy. Here we are 'applying' a function over columns of a dataset:

```r
R> data(faithful)
R> lapply(faithful, summary)

$eruptions
  Min. 1st Qu. Median Mean 3rd Qu. Max.
  1.60  2.16  4.00  3.49  4.45  5.10

$waiting
  Min. 1st Qu. Median Mean 3rd Qu. Max.
  43.0  58.0  76.0  70.9  82.0  96.0
```

We can do that in C++ as well and pass the R function down to the data elements we let the STL iterate over:

```cpp
src ← 'Rcpp::List input(data);
    Rcpp::Function f(fun);
    Rcpp::List output(input.size());
    std::transform(input.begin(), input.end(), output.begin(), f);
    output.names() = input.names();
    return output;

cpp_lapply ← cfunction(signature(data="list", fun = "function"), src, Rcpp = TRUE )
```
The convolution example can be rewritten in the new API:

```cpp
#include <Rcpp.h>

RcppExport SEXP convolve_cpp(SEXP a, SEXP b) {
    Rcpp::NumericVector xa(a); // automatic conversion from SEXP
    Rcpp::NumericVector xb(b);

    int n_xa = xa.size();
    int n_xb = xb.size();
    int nab = n_xa + n_xb - 1;

    Rcpp::NumericVector xab(nab);  

    for (int i = 0; i < n_xa; i++)
        for (int j = 0; j < n_xb; j++)
            xab[i + j] += xa[i] * xb[j];

    return xab;  // automatic conversion to SEXP
}
```
In a paper we are about to submit, the following table summarises the convolution performance:

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Time in millisec</th>
<th>Relative to R API</th>
</tr>
</thead>
<tbody>
<tr>
<td>R API (as benchmark)</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>RcppVector&lt;double&gt;</td>
<td>354</td>
<td>11.1</td>
</tr>
<tr>
<td>NumericVector::operator[]</td>
<td>52</td>
<td>1.6</td>
</tr>
<tr>
<td>NumericVector::begin</td>
<td>33</td>
<td>1.0</td>
</tr>
</tbody>
</table>

**Table 1: Performance for convolution example**

We used 1000 replications with two 100-element vectors.
Another Speed Comparison Example

Regression is a key component of many studies. In simulations, we often want to run a very large number of regressions.

R has `lm()` as the general purposes function. It is very powerful and returns a rich object—but it is not lightweight.

For this purpose, R has `lm.fit()`. But, this does not provide all relevant auxiliary data as e.g. the standard error of the estimate.

For one of the *Intro to High-Performance Computing with R* tutorials, I had created a hybrid R/C/C++ solution using GSL.

We complement this with a new C++ implementation around the Armadillo linear algebra classes.
Linear regression via GSL

```r
lmGSL <- function() {
  src <- 

  RcppVectorView<double> Yr(Ysexp);
  RcppMatrixView<double> Xr(Xsexp);

  int i, j, n = Xr.dim1(), k = Xr.dim2();
  double chi2;

  gsl_matrix *X = gsl_matrix_alloc(n,k);
  gsl_vector *y = gsl_vector_alloc(n);
  gsl_vector *c = gsl_vector_alloc(k);
  gsl_matrix *cov = gsl_matrix_alloc(k,k);

  for (i = 0; i < n; i++) {
    for (j = 0; j < k; j++) {
      gsl_matrix_set (X, i, j, Xr(i,j));
    }
    gsl_vector_set (y, i, Yr(i));
  }

  gsl_multifit_linear_workspace *wk =
    gsl_multifit_linear_alloc(n,k);
  gsl_multifit_linear (X, y, c, cov, &chi2, wk);
  gsl_multifit_linear_free (wk);
  RcppVector<double> StdErr(k);
  RcppVector<double> Coef(k);

  for (i = 0; i < k; i++) {
    Coef(i) = gsl_vector_get (c, i);
    StdErr(i) = sqrt(gsl_matrix_get(cov, i, i));
  }

  gsl_matrix_free (X);
  gsl_vector_free (y);
  gsl_vector_free (c);
  gsl_matrix_free (cov);

  RcppResultSet rs;
  rs.add("coef", Coef);
  rs.add("stderr", StdErr);

  return = rs.getReturnList();
}
```

for (i = 0; i < k; i++) {
  Coef(i) = gsl_vector_get (c, i);
  StdErr(i) = sqrt(gsl_matrix_get(cov, i, i));
}

gsl_matrix_free (X);
gsl_vector_free (y);
gsl_vector_free (c);
gsl_matrix_free (cov);
RcppResultSet rs;
rs.add("coef", Coef);
r

return = rs.getReturnList();
```
Linear regression via Armadillo

```r
lmArmadillo <- function() {
  src <- 'Rcpp::NumericVector yr(Ysexp);
  Rcpp::NumericVector Xr(Xsexp); // actually an n x k matrix
  std::vector<int> dims = Xr.attr("dim");
  int n = dims[0], k = dims[1];
  arma::mat X(Xr.begin(), n, k, false); // use advanced armadillo constructors
  arma::colvec y(yr.begin(), yr.size());
  arma::colvec coef = solve(X, y); // model fit
  arma::colvec resid = y - X*coef; // to comp. std.err of the coefficients
  arma::mat covmat = trans(resid)*resid/(n-k) * arma::inv(arma::trans(X)*X);

  Rcpp::NumericVector coefr(k), stderrestr(k);
  for (int i=0; i<k; i++) {
    // with RcppArmadillo template converters
    coefr[i] = coef[i]; // this would not be needed but we only
    stderrestr[i] = sqrt(covmat(i, i)); // have Rcpp.h here
  }

  Rcpp::Pairlist res(Rcpp::Named("coef", coefr),
                     Rcpp::Named("stderr", stderrestr));
  return res;
}

## turn into a function that R can call
fun <- cfunction(signature(Ysexp="numeric", Xsexp="numeric"),
  src, includes="#include <armadillo>", Rcpp=TRUE,
  cppargs="-I/usr/include", libargs="-larmadillo")
```

Dirk Eddelbuettel
Programming with Data: Using and extending R
Rcpp Example: Regression timings

The small *longley* example exhibits less variability between methods, but the larger data set shows the gains more clearly.

For the small data set, all three appear to improve similarly on *lm*.

Source: Our calculations
Another Rcpp example (cont.)

Comparison of R and linear model fit routines

By dividing the `lm` time by the respective times, we obtain the ’possible gains’ from switching.

One caveat, measurements depends critically on the size of the data as well as the cpu and libraries that are used.

Source: Our calculations
Outline

1. Intro
2. S/R
3. Extending
4. Rcpp
5. Rcpp Usage Examples
6. Summary
Jeff Horner’s work on **RApache** lead to joint work in **littler**, a scripting / cmdline front-end. As it embeds **R** and simply ’feeds’ the REPL loop, the next step was to embed **R** in proper **C++** classes: **RInside**.

```cpp
#include "RInside.h"  // for the embedded R via RInside

int main(int argc, char *argv[]) {

    RInside R(argc, argv);  // create an embedded R instance

    std::string txt = "Hello, world!
"; // assign a standard C++ string to 'txt'
    R["txt"] = txt;  // assign C++ string var to R variable 'txt'

    std::string evalstr = "cat(txt)";
    R.parseEvalQ(evalstr);  // eval the init string, ignoring any returns

    exit(0);
}
```
Another simple example

This example shows some of the new assignment and converter code:

```cpp
#include "RInside.h" // for the embedded R via RInside

int main(int argc, char *argv[]) {
    RInside R(argc, argv); // create an embedded R instance

    R["x"] = 10;
    R["y"] = 20;

    R.parseEvalQ("z <- x + y");
    int sum = Rcpp::as<int>(R["z"]);

    std::cout << "10 + 20 = " << sum << std::endl;
    exit(0);
}
```
C++ programs compute, gather or aggregate raw data.
Data is saved and analysed before a new ’run’ is launched.
With RInside we now skip a step:
- collect data in a vector or matrix
- pass data to R — easy thanks to Rcpp wrappers
- pass one or more short ’scripts’ as strings to R to evaluate
- pass data back to C++ programm — easy thanks to Rcpp converters
- resume main execution based on new results
A number of simple examples ship with RInside
Quoting from the page at Google Code:

*Protocol buffers are a flexible, efficient, automated mechanism for serializing structured data—think XML, but smaller, faster, and simpler.*

You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages.

You can even update your data structure without breaking deployed programs that are compiled against the "old" format.

Google provides native bindings for C++, Java and Python.
R> library( RProtoBuf ) ## load the package
R> readProtoFiles( "addressbook.proto" ) ## acquire protobuf information
R> bob <- new( tutorial.Person, ## create new object
  +   email = "bob@example.com",
  +   name = "Bob",
  +   id = 123 )
R> writeLines( bob$toString() ) ## serialize to stdout
name: "Bob"
id: 123
e-mail: "bob@example.com"

R> bob$email ## access and/or override
[1] "bob@example.com"
R> bob$id <- 5
R> bob$id
[1] 5

R> serialize( bob, "person.pb" ) ## serialize to compact binary format

Under the hood, **Rcpp** is used extensively and works very well in conjunction with the rich C++ API provided by Google.
This presentation has tried to convince you that

- Data matters, and data skills matter more and more
- R is designed for *Programming with Data*
- R is being applied to do just about any field
- R can be extended in many ways; we focussed on
  - extensions reasonably close to the wire using C++
  - allowing us to extend R with C++ and
  - allowing us to embed R inside C++
- all while retaining ’high-level’ STL-alike semantics
- R, as a first class Open Source citizen with a wonderful community, is a joy to work with.
And a short commercial
A steal at $25 for a student registration