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

use **R**!

Outline

It's the data, stupid

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.

Source: Hal Varian, Freakonomics blog, `http://freakonomics.blogs.nytimes.com/2008/02/25/hal-varian-answers-your-questions/`

Statistics as the new in-thing

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.

Source: Hal Varian, The McKinsey Quarterly, `http://www.mckinseyquarterly.com/Hal_Varian_on_how_the_Web_challenges_managers_2286`

Outline

R and S: A really brief Overview

- R is a dialect of S which was started at Bell Labs in 1975
- S won the 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
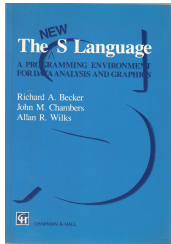 maintained by a core group of academic statisticians /
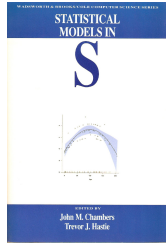 computer scientists

R History by the books
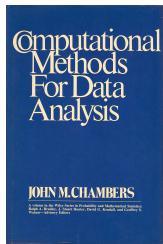
Chambers, *Computational Methods for Data Analysis*. Wiley, 1977.

Earliest publication of what became S.

Becker, Chambers, and Wilks. *The New S Language*. Chapman & Hall, 1988.
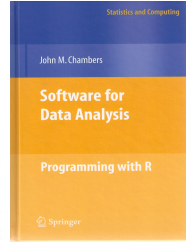
Introduced what is now known as S version 3 (S3)

Chambers and Hastie. *Statistical Models in S*. Chapman & Hall, 1992.

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

Chambers. *Programming with Data*. Springer, 1998.

Version 4 of S, a major revision of S.

Chambers. *Software for Data Analysis: Programming with R*. Springer, 2008

The R version of S4 and other R techniques.

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

R has now become mainstream

The New York Times notices

Dice calls it a job market edge

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

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 Visualizatons**: 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!
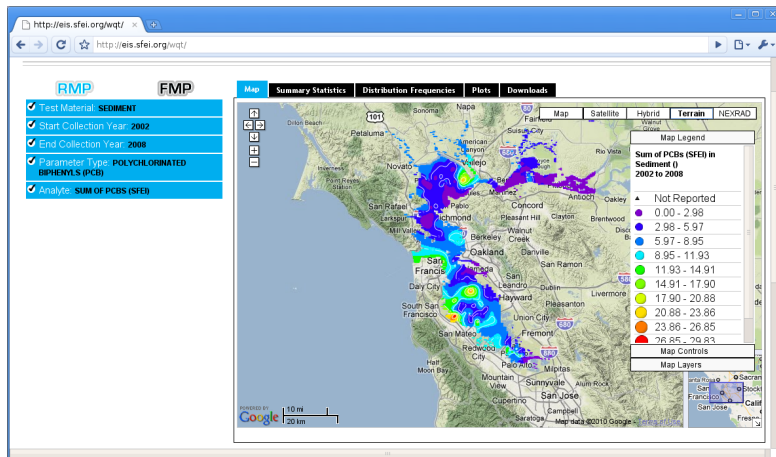
Web Dashboard Examples using RApache

- 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/`
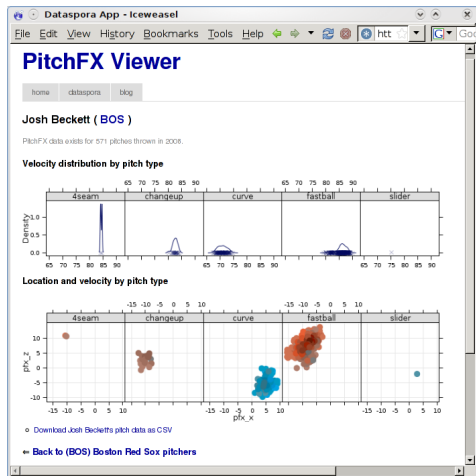
Another Web-based Example: Google/R Mashups

Source: http://eis.sfei.org/wqt/

An Example related to SVN Commits

Number of SVN commits since 2006
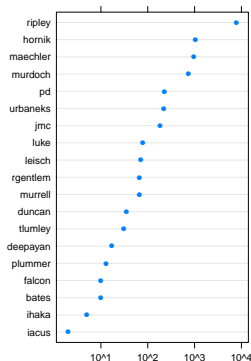
All it takes are these few lines of code:

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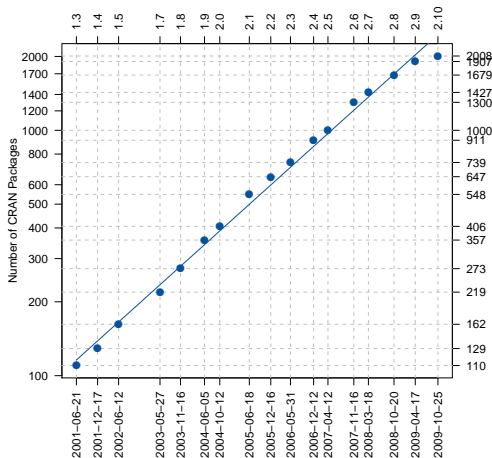
- CRAN archive network growing by 40% p.a., now at around 2150 packages
- John Fox provided this chart in an invited lecture at the *useR! 2008* meetings.
- Details, and more metrics on R and the dynamics of the R Core group, are also in a recent R Journal article.

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

Compiled Code: The Basics

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:

```
1  void convolve(double *a, int *na, double *b,
2                int *nb, double *ab)
3  {
4    int i, j, nab = *na + *nb - 1;
5
6    for(i = 0; i < nab; i++)
7      ab[i] = 0.0;
8    for(i = 0; i < *na; i++)
9      for(j = 0; j < *nb; j++)
10       ab[i + j] += a[i] * b[j];
11 }
```

Compiled Code: The Basics cont.

The convolution function is called from R by

```
1  conv <- function(a, b)
2    .C("convolve",
3      as.double(a),
4      as.integer(length(a)),
5      as.double(b),
6      as.integer(length(b)),
7      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.

Compiled Code: The Basics cont.

Using `.Call`, the example becomes

```
1  #include <R.h>
2  #include <Rdefines.h>
3
4  extern "C" SEXP convolve2(SEXP a, SEXP b)
5  {
6      int i, j, na, nb, nab;
7      double *xa, *xb, *xab;
8      SEXP ab;
9
10     PROTECT(a = AS_NUMERIC(a));
11     PROTECT(b = AS_NUMERIC(b));
12     na = LENGTH(a); nb = LENGTH(b); nab = na + nb - 1;
13     PROTECT(ab = NEW_NUMERIC(nab));
14     xa = NUMERIC_POINTER(a); xb = NUMERIC_POINTER(b);
15     xab = NUMERIC_POINTER(ab);
16     for(i = 0; i < nab; i++) xab[i] = 0.0;
17     for(i = 0; i < na; i++)
18         for(j = 0; j < nb; j++) xab[i + j] += xa[i] * xb[j];
19     UNPROTECT(3);
20     return(ab);
21 }
```

use R!

Compiled Code: The Basics cont.

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:

```
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.

Compiled Code: inline

inline is a package by Oleg Sklyar et al that provides the function cfunction which can wrap Fortran, C or C++ code.

```
1  ## A simple Fortran example
2  code <- "
3          integer i
4          do 1 i=1, n(1)
5      1 x(i) = x(i)**3
6  "
7  cubefn <- cfunction(signature(n="integer", x="numeric"),
8                      code, convention=".Fortran")
9  x <- as.numeric(1:10)
10 n <- as.integer(10)
11 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 .

Outline

Compiled Code: Rcpp

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

Using the `.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 `.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.

Rcpp example

The convolution example can be rewritten as follows in the 'Classic API':

```
1   #include <Rcpp.h>
2
3   RcppExport SEXP convolve_cpp(SEXP a, SEXP b)
4   {
5       RcppVector<double> xa(a);
6       RcppVector<double> xb(b);
7
8       int nab = xa.size() + xb.size() − 1;
9
10      RcppVector<double> xab(nab);
11      for (int i = 0; i < nab; i++) xab(i) = 0.0;
12
13      for (int i = 0; i < xa.size(); i++)
14          for (int j = 0; j < xb.size(); j++)
15              xab(i + j) += xa(i) * xb(j);
16
17      RcppResultSet rs;
18      rs.add("ab", xab);
19      return rs.getReturnList();
20  }
```

Rcpp: The 'New API'

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;
```

Working on almost all datatypes

In R , functional programming easy. Here we are 'applying' a function over columns of a dataset:

```
1  R> data(faithful)
2  R> lapply(faithful, summary)
3  $eruptions
4     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
5     1.60    2.16    4.00    3.49    4.45    5.10
6
7  $waiting
8     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
9     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:

```
1  src <- 'Rcpp::List input(data);
2          Rcpp::Function f(fun) ;
3          Rcpp::List output(input.size());
4          std::transform(input.begin(), input.end(), output.begin(), f);
5          output.names() = input.names();
6          return output; '
   cpp_lapply <- cfunction(signature(data="list", fun = "function"), src, Rcpp = TRUE )
```

Rcpp example

The convolution example can be rewritten in the new API:

```
1  #include <Rcpp.h>
2
3  RcppExport SEXP convolve_cpp(SEXP a, SEXP b){
4      Rcpp::NumericVector xa(a); // automatic conversion from SEXP
5      Rcpp::NumericVector xb(b);
6
7      int n_xa = xa.size();
8      int n_xb = xb.size();
9      int nab = n_xa + n_xb - 1;
10
11     Rcpp::NumericVector xab(nab);
12
13     for (int i = 0; i < n_xa; i++)
14         for (int j = 0; j < n_xb; j++)
15             xab[i + j] += xa[i] * xb[j];
16
17     return xab;  // automatic conversion to SEXP
18 }
```

Speed comparison

In a paper we are about to submit, the following table summarises the convolution performance:

Implementation	Time in millisec	Relative to R API
R API (as benchmark)	32	
`RcppVector<double>`	354	11.1
`NumericVector::operator[]`	52	1.6
`NumericVector::begin`	33	1.0

Table 1: Performance for convolution example

We used 1000 replications with two 100-element vectors.

Another Speed Comparison Example

Regression is a key compoent 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

```
1   lmGSL <- function() {
2     src <- '
3
4     RcppVectorView<double> Yr(Ysexp);
5     RcppMatrixView<double> Xr(Xsexp);
6
7     int i,j,n = Xr.dim1(), k = Xr.dim2();
8     double chi2;
9
10    gsl_matrix *X = gsl_matrix_alloc(n,k);
11    gsl_vector *y = gsl_vector_alloc(n);
12    gsl_vector *c = gsl_vector_alloc(k);
13    gsl_matrix *cov = gsl_matrix_alloc(k,k);
14
15    for (i = 0; i < n; i++) {
16      for (j = 0; j < k; j++) {
17        gsl_matrix_set (X, i, j, Xr(i,j));
18      }
19      gsl_vector_set (y, i, Yr(i));
20    }
21
22    gsl_multifit_linear_workspace *wk =
23           gsl_multifit_linear_alloc(n,k);
24    gsl_multifit_linear(X,y,c,cov,&chi2,wk);
25    gsl_multifit_linear_free (wk);
26    RcppVector<double> StdErr(k);
27    RcppVector<double> Coef(k);
```

```
28    for (i = 0; i < k; i++) {
29      Coef(i) = gsl_vector_get(c,i);
30      StdErr(i) =
31           sqrt(gsl_matrix_get(cov,i,i));
32    }
33
34    gsl_matrix_free (X);
35    gsl_vector_free (y);
36    gsl_vector_free (c);
37    gsl_matrix_free (cov);
38
39    RcppResultSet rs;
40    rs.add("coef", Coef);
41    rs.add("stderr", StdErr);
42
43    return = rs.getReturnList();
44    '
45    ## turn into a function that R can call
46    ## args redundant on Debian/Ubuntu
47    fun <-
48      cfunction(signature(Ysexp="numeric",
49        Xsexp="numeric"), src,
50        includes=
51           "#include <gsl/gsl_multifit.h>",
52        Rcpp=TRUE,
53        cppargs="-I/usr/include",
54        libargs="-lgsl -lgslcblas")
55  }
```

Linear regression via Armadillo

```
 1  lmArmadillo <- function() {
 2      src <- '
 3      Rcpp::NumericVector yr(Ysexp);
 4      Rcpp::NumericVector Xr(Xsexp);          // actually an n x k matrix
 5      std::vector<int> dims = Xr.attr("dim");
 6      int n = dims[0], k = dims[1];
 7      arma::mat X(Xr.begin(), n, k, false);   // use advanced armadillo constructors
 8      arma::colvec y(yr.begin(), yr.size());
 9      arma::colvec coef = solve(X, y);         // model fit
10      arma::colvec resid = y - X*coef;         // to comp. std.errr of the coefficients
11      arma::mat covmat = trans(resid)*resid/(n-k) * arma::inv(arma::trans(X)*X);
12
13      Rcpp::NumericVector coefr(k), stderrestr(k);
14      for (int i=0; i<k; i++) {                // with RcppArmadillo template converters
15          coefr[i]     = coef[i];              // this would not be needed but we only
16          stderrestr[i] = sqrt(covmat(i,i));   // have Rcpp.h here
17      }
18
19
20      Rcpp::Pairlist res(Rcpp::Named( "coef", coefr),
21                         Rcpp::Named( "stderr", stderrestr));
22      return res;
23      '
24
25      ## turn into a function that R can call
26      fun <- cfunction(signature(Ysexp="numeric", Xsexp="numeric"),
27                       src, includes="#include <armadillo>", Rcpp=TRUE,
28                       cppargs="-I/usr/include", libargs="-larmadillo")
29  }
```

Rcpp Example: Regression timings

Comparison of R and linear model fit routines
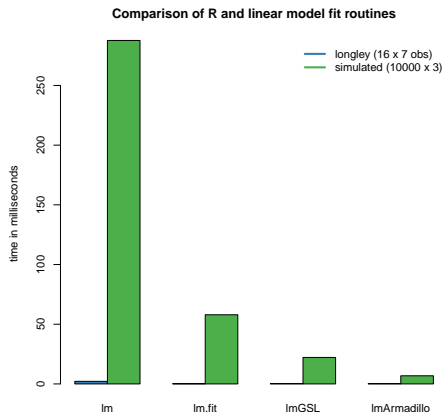
The small `longley` example exhibits less variability between methods, but the larger data set shows the gains more clearly.

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

longley (16 x 7 obs)
simulated (10000 x 3)

ratio to lm() baseline

lm   lm.fit   lmGSL   lmArmadillo

Source: Our calculations
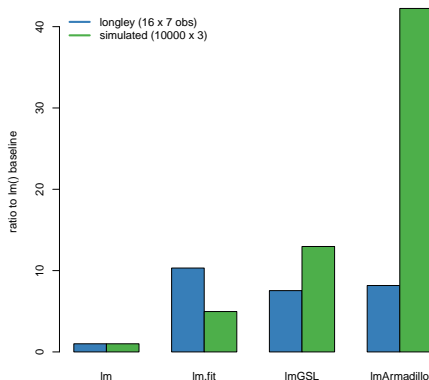
By dividing the `lm` time by the respective times, we obtain the 'possible gains' from switching.

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

Outline

From RApache to littler to RInside

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!\n";// 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:

```
#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);
}
```

RInside workflow

- 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`

About Google ProtoBuf

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.

Google ProtoBuf

```
1  R> library( RProtoBuf )                      ## load the package
2  R> readProtoFiles( "addressbook.proto" )     ## acquire protobuf information
3  R> bob <- new( tutorial.Person,              ## create new object
4  +    email = "bob@example.com",
5  +    name = "Bob",
6  +    id = 123 )
7  R> writeLines( bob$toString() )              ## serialize to stdout
8  name: "Bob"
9  id: 123
10 email: "bob@example.com"
11
12 R> bob$email                                 ## access and/or override
13 [1] "bob@example.com"
14 R> bob$id <- 5
15 R> bob$id
16 [1] 5
17
18 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.

Outline

Wrapping up

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++
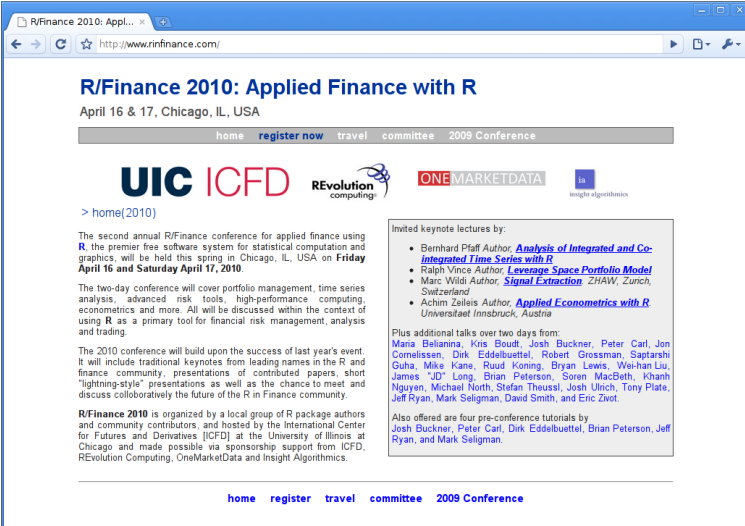
  all while retaining 'high-level' STL-alike semantics
- R, as a first class Open Source citizen with a wonderful community, is a joy to work with.

And a short commercial
A steal at $25 for a student registration