Hands-On Advanced Rcpp

Dr. Dirk Eddelbuettel
dirk.eddelbuettel@R-Project.org
@eddelbuettel

Invited Lecture
Practical Computing for Economists
Department of Economics
University of Chicago
May 30, 2014

Outline

Basic Agenda

As you already know (some) **Rcpp** and **RcppArmadillo**, we will try to cover the following:

- Simple classes for stateful computation: RcppZiggurat and faster Normal RNGs
- Simple interfaces to external libraries: RcppRedis to access Redis
- Passing compiled objective functions to compiled optimizers
- C / C++ as *glue code*: RcppOctave; embedding Python via Rcpp

We will follow existing packages which will allow you to experiment with this.

Outline

Simulation settting

Rcpp is often used to accelerate simulations, *e.g.* the Gibbs sampler you already coded up.

```
#include <Rcpp.h>                    // load Rcpp
using namespace Rcpp;               // shorthand
// [[Rcpp::export]]
NumericMatrix RcppGibbs(int n, int thn) {
    int i,j;
    NumericMatrix mat(n, 2);
    double x=0, y=0;
    for (i=0; i<n; i++) {
        for (j=0; j<thn; j++) {
            x = R::rgamma(3.0,1.0/(y*y+4));
            y = R::rnorm(1.0/(x+1),1.0/sqrt(2*x+2));
        }
        mat(i,0) = x;
        mat(i,1) = y;
    }
    return mat;             // Return to R

}
```

Simulation settting

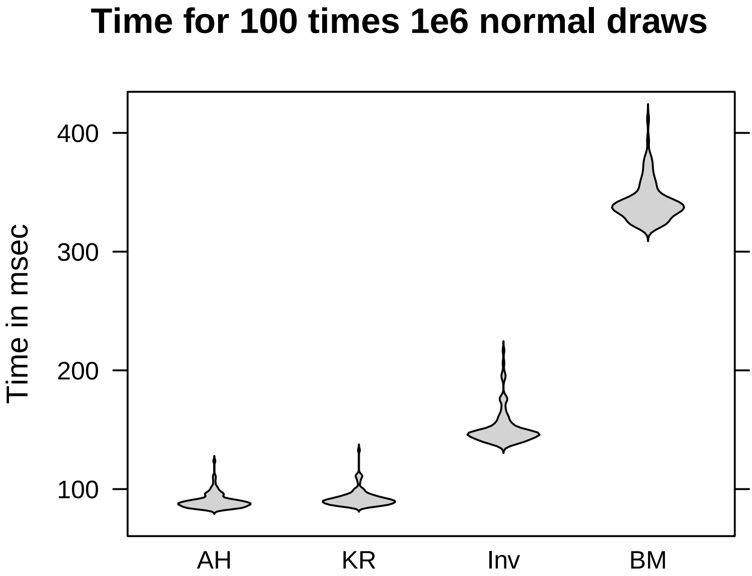
Rcpp helps us to make loops a lot faster, and improves the speed of other operations too.

But some things (such as calls into compiled code) remain unchanged.

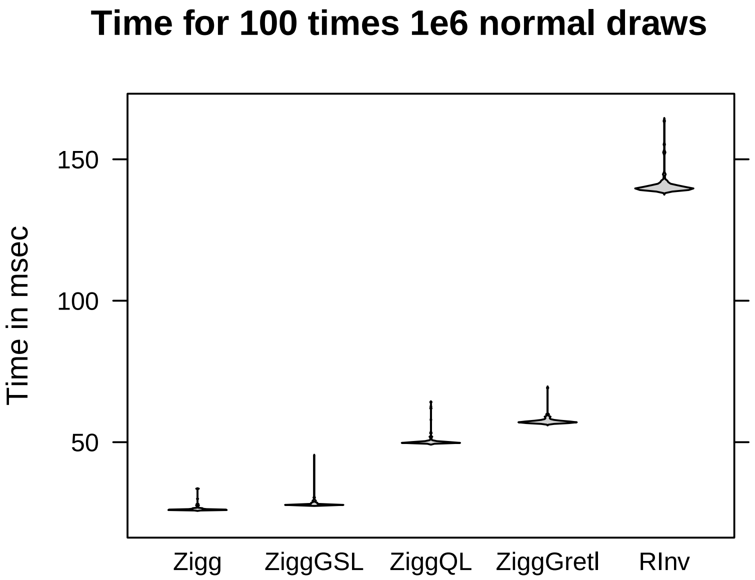
And the RNGs in R (while of excellent statistical quality) are one such item.

The next slide shows timing of the Ahrens-Dieter (AH), Kinderman-Ramage (KR), Inversion (Inv) and Box-Muller (BM) generators for N(0,1) draws.

R Normal RNGs



Ziggurat Speeds



Basic Ziggurat

```
#include <math.h>
static unsigned long jz,jsr=123456789;

#define SHR3 (jz=jsr, jsr^=(jsr<<13), jsr^=(jsr>>17), \
              jsr^=(jsr<<5),jz+jsr)
#define UNI (.5 + (signed) SHR3*.2328306e-9)
#define IUNI SHR3

static long hz;
static unsigned long iz, kn[128], ke[256];
static float wn[128],fn[128], we[256],fe[256];

#define RNOR (hz=SHR3, iz=hz&127, \
              (fabs(hz)<kn[iz])? hz*wn[iz] : nfix())
```

A macro (!!) where `nfix()` is a tail correction invoked $<$ 2% of calls.

Base Class

```
#include <cmath>
#include <stdint.h>                 // or cstdint with C++11

namespace Ziggurat {

    class Zigg {
    public:
        virtual ~Zigg() {};
        virtual void setSeed(const uint32_t s) = 0;
        // no getSeed() as GSL has none
        virtual double norm() = 0;
    };
}
```

Used by several implementations in the package.

Parts of original Ziggurat

```
#include <Zigg.h>

#define znew (z = 36969 * (z & 65535) + ( z >> 16 ))
#define wnew (w = 18000 * (w & 65535) + ( w >> 16 ))
// ...

    class Ziggurat : public Zigg {
    public:
        Ziggurat(uint32_t seed=123456789) : jcong(234567891), jsr(123456789),
                                            w(345678912), z(456789123) {
            init();
            setSeed(seed);
        }
        ~Ziggurat() {};
        void setSeed(const uint32_t s) { /* ... */ }

    private:
        float fn[128], wn128];
        int32_t hz;
        uint32_t iz, jcong, uint32_t jsr, uint32_t jz, uint32_t kn[128], w, z;

        void init() { /* ... */ }

        inline float nfix(void) {  /* ... */  }
    };

#undef znew
#undef wnew

// ...
```

Usage is in the src/ directory

```
// Version 1 -- Derived from Marsaglia and Tsang, JSS, 2000
static Ziggurat::MT::ZigguratMT ziggmt;

// Marsaglia and Tsang (JSS,2000)
// [[Rcpp::export]]
Rcpp::NumericVector zrnormMT(int n) {
    Rcpp::NumericVector x(n);
    for (int i=0; i<n; i++) {
        x[i] = ziggmt.norm();
    }
    return x;
}
// [[Rcpp::export]]
void zsetseedMT(int s) {
    ziggmt.setSeed(s);
}
```

Outline

Overview

Why the hype?

- **Simple**: Does one thing, and does it well
- **Fast**: Run `redis-benchmark` to see just how fast
- **Widely used**: Twitter, GitHub, Craigslist, StackOverflow, . . .
- **Multi-language**: Bindings from anything you may use
- **Active**: Well maintained and documented

Write from Python

```
#!/usr/bin/python

import redis

redishost   = "localhost"
redisserver = redis.StrictRedis(redishost)

key = "ex:ascii:simpleString"
val = "abracadabra"
res = redisserver.set(key, val)
```

Read in R

```
library(rredis)

redisConnect()

key <- "ex:ascii:simpleString"
val <- redisGet(key)
cat("Got", val, "from", key, "\n")

## Got abracadabra from ex:ascii:simpleString
```

Or read in Shell

```
$ redis-cli get ex:ascii:simpleString
"abracadabra"
$
```

More generally

We can

- Read
- Write

from just about any programming language or shell.

(So far) all we require is string processing.

Data Structures

Redis supports many relevant data types:

- Strings
- Hashes
- Lists
- Sets
- Sorted Sets

as well as transactions, key management, pub/sub, embedded scripting, connection management and more.

rredis

Wonderful package by Bryan Lewis that covers (all of ?) Redis

Awesome for things like
`redisSet("myModel", lm(someFormula, someData))`

(Mostly) efficient enough.

Uses string format exclusively.

Automagically deploys R serialization.

Also used as backend for **doRedis**

Simple helper functions

```
redisConnect("someServer.some.net")

rput <- function(X) {
    xstr <- deparse(substitute(X))
    redisSet(xstr, X)
}

rget <- function(key) {
    val <- redisGet(key)      # default instance
    redisDelete(key)
    invisible(val)
}
```

Even nicer: memoise by Michael Kane

```
require(rredis)
redisConnect()

memoize <- function(expr, key=NULL, expire_time=Inf,
                    verbose=FALSE, envir=parent.frame()) {
  if (is.null(key)) {
    key <- paste(substitute(expr), collapse="")
  }
  if (redisExists(key)) {
    ret <- redisGet(key)
  } else {
    ret <- eval(substitute(expr), envir=envir)
    redisSet(key, ret)
  }
  if (expire_time < Inf) {
    redisExpireAt(proj_doc_key,
      as.integer(as.POSIXct(Sys.time())+expire_time))
  }
  ret
}
```

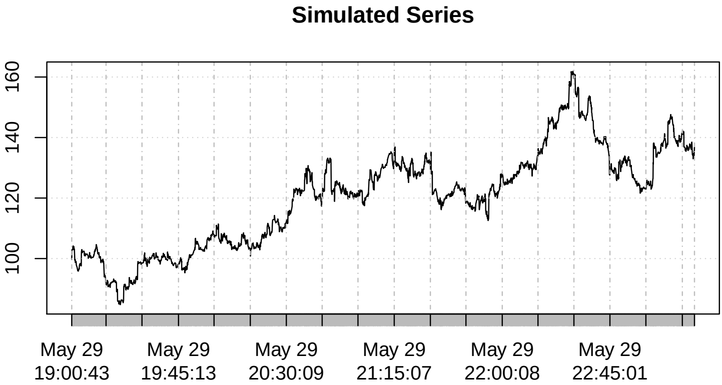
Time series

Our basic premise and idea is to deploy disconnected writers (middleware clients in C, C++, Python, ...) and consumers (R) – by placing Redis in the middle.

But for "longer" time series the combined cost of deserialization and parsing is too high in R.

Example

```
set.seed(123); N <- 2500
x <- xts(100*cumprod(1+rnorm(N)*0.005 +
            (runif(N)>0.95)*rnorm(N)*0.025),
         order.by=Sys.time()+cumsum(exp(3*runif(N))))
plot(x, main="Simulated Series", type='l')
```



Writing and Reading

With **rredis** we set and get the time series as follows:

```
setAsAscii <- function(dat) {
    N <- nrow(dat)
    ## insertion is row by row
    for (i in 1:N) {
        redisZAdd("ex:ascii:series",
                  dat[i,1], dat[i,])
    }
}

## retrieval is by list
getFromAscii <- function() {
    xx <- do.call(rbind,
                  redisZRange("ex:ascii:series", 0, -1))
    xt <- xts(xx[,-1],
              order.by=as.POSIXct(xx[,1], origin="1970-01-01"))
}
```

RApiSerialize

A (fairly new) CRAN package we released recently.

It does just one thing: give us serialization and deserialization from the R API at the C(++) level.

It is used by **RcppRedis**, and provides it with C-level (de-)serialization without having to call “up” to R.

RcppRedis

A (fairly new) (and higly incomplete) CRAN package (as of yesterday).

It covers just a couple of commands, but those run rather fast.

Writing and Reading

```
setAsBinary <- function(dat) {
    redis$zadd("ex:bin:series", as.matrix(dat))
}

getFromBinary <- function() {
    zz <- redis$zrange("ex:bin:series", 0, -1)
    zt <- xts(zz[,-1],
              order.by=as.POSIXct(zz[,1], origin="1970-01-01"))
}
```

Writing and Reading – Part Two

```
// redis "zadd" -- insert score + matrix row (no R serial.)
// by convention, 1st elem of row vector is the score value
double zadd(std::string key, Rcpp::NumericMatrix x) {
  double res = 0;
  for (int i=0; i<x.nrow(); i++) {
    Rcpp::NumericVector y = x.row(i);
    // uses binary protocol, see hiredis doc at github
    redisReply *reply =
       static_cast<redisReply*>(redisCommand(prc_,
                                             "ZADD %s %f %b",
                                             key.c_str(),
                                             y[0],
                                             y.begin(),
                                             y.size()*szdb));
    checkReplyType(reply, replyInteger_t);
    res += static_cast<double>(reply->integer);
    freeReplyObject(reply);
  }
  return(res);
}
```

Net Effect: demo/simDemo.R

```
##
## Writing
##
                 test replications elapsed relative
setAsBinary(dat)            1   0.127    1.000
 setAsAscii(dat)            1 100.001  787.409
##
## Reading
##
               test replications elapsed relative
getFromBinary()           10   0.031    1.000
 getFromAscii()           10   4.792  154.581
```

Mechanics: Link against libhiredis

- For the CRAN package, tiny bit of `configure` logic to find `CFLAGS` and `LIBS`; used in `src/Makevars`
- For local research use, just hardcode it
- Key is to tell compiler about headers, and linker about libraries
- Many powerful C and C++ libraries out there, learning to bind to them is useful
- hiredis is easy to build and a good test case

Outline

Differential Evolution: DEoptim and RcppDE

- The **DEoptim** package by Ardia, Mullen et al is a popular and powerful optimiser using the *differential evolution* variant of evolutionary optimization.
- At some point I had set out to port to see if I could go from "easier, shorter, faster: pick any two" to hitting all three
- Code size was reduced from over 700 lines of C to about 400 lines of C++ in package **RcppDE** thanks to **Armadillo**.
- By virtue of diligent code review, I also made it faster.
- Josh Ulrich incorporated those changes so **DEoptim** closed the gap; it has since moved on.
- **RcppDE** pending rework for parallelism via OpenMP.

DEoptim

Popular to optimise classic problems from the literature:

```
Wild <- function(x) {            ## 'Wild' function, global minimum at about -15.81515
    sum(10 * sin(0.3 * x) * sin(1.3 * x^2) + 0.00001 * x^4 + 0.2 * x + 80)/length(x)
}

Rastrigin <- function(x) {
    sum(x+2 - 10 * cos(2*pi*x)) + 20
}

## One generalization of the Rosenbrock banana valley function (n parameters)
Genrose <- function(x) {
    n <- length(x)
    1.0 + sum (100 * (x[-n]^2 - x[-1])^2 + (x[-1] - 1)^2)
}
```

RcppDE allows for compiled objective functions

```
#include <Rcpp.h>
// [[Rcpp::interfaces(r, cpp)]]

double wild(SEXP xs) {
  Rcpp::NumericVector x(xs);
  double sum = 0.0;
  for (int i=0; i<x.size(); i++)
    sum += 10 * sin(0.3 * x[i]) * sin(1.3 * x[i]*x[i]) +
           0.00001 * x[i]*x[i]*x[i]*x[i] + 0.2 * x[i] + 80;
  sum /= x.size();
  return(sum);
}

double rastrigin(SEXP xs) {
  Rcpp::NumericVector x(xs);
  int n = x.size();
  double sum = 20.0;
  for (int i=0; i<n; i++) {
    sum += x[i]+2 - 10*cos(2*M_PI*x[i]);
  }
  return(sum);
}

double genrose(SEXP xs) {
  Rcpp::NumericVector x(xs);
  double sum = 1.0;
  for (int i=1; i<x.size(); i++)
    sum += 100*( pow(x[i-1]*x[i-1] - x[i], 2)) + (x[i] - 1)*(x[i] - 1);
  return(sum);
}
```

RcppDE allows for compiled objective functions

```
// cont. from previous slide

// [[Rcpp::export]]
SEXP create_xptr(std::string fstr) {
  typedef double (*funcPtr)(SEXP);
  if (fstr == "genrose")
    return(Rcpp::XPtr<funcPtr>(new funcPtr(&genrose)));
  else if (fstr == "wild")
    return(Rcpp::XPtr<funcPtr>(new funcPtr(&wild)));
  else
    return(Rcpp::XPtr<funcPtr>(new funcPtr(&rastrigin)));
}
```

RcppDE and compiled objective functions

```
edd@max:~/git/rcppde/demo$ r CompiledBenchmark.R
# At 2014-05-26 18:35:43
              DEoptim RcppDE ratioRcppToBasic pctGainOfRcpp netSpeedUp
Rastrigin50   1.81100  0.671       0.37051353      62.94865   2.698957
Rastrigin100  5.17800  2.171       0.41927385      58.07261   2.385076
Rastrigin200 17.81500  7.917       0.44440079      55.55992   2.250221
Wild50        3.64200  1.033       0.28363537      71.63646   3.525653
Wild100      11.28100  3.572       0.31663860      68.33614   3.158175
Wild200      40.41700 13.478       0.33347354      66.65265   2.998739
Genrose50     2.65800  0.285       0.10722348      89.27765   9.326316
Genrose100    7.66200  0.704       0.09188202      90.81180  10.883523
Genrose200   23.64100  2.110       0.08925172      91.07483  11.204265
MEANS        12.67833  3.549       0.27992638      72.00736   3.572368

# Done 2014-05-26 18:38:10
```

Passing user-defined C++ functions R to C++

http://gallery.rcpp.org/articles/passing-cpp-function-pointers/

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

// [[Rcpp::export]]
arma::vec fun_cpp(const arma::vec& x) { return(10*x); }

typedef arma::vec (*funcPtr)(const arma::vec& x);

// [[Rcpp::export]]
Rcpp::XPtr<funcPtr> putFunPtrInXPtr() {
    return(Rcpp::XPtr<funcPtr>(new funcPtr(&fun_cpp)));
}

// [[Rcpp::export]]
arma::vec callViaXPtr(const arma::vec x, SEXP xpsexp) {
    Rcpp::XPtr<funcPtr> xpfun(xpsexp);
    funcPtr fun = *xpfun;
    arma::vec y = fun(x);
    return(y);
}
```

Passing user-defined C++ functions R to C++

http://gallery.rcpp.org/articles/passing-cpp-function-pointers/

Quick illustration:

```
fun <- putFunPtrInXPtr()
callViaXPtr(1:4, fun)

##      [,1]
## [1,]   10
## [2,]   20
## [3,]   30
## [4,]   40
```

Outline

RcppOctave embeds Octave

- Package by Renaud Gaujoux
- Embeds Octave using Rcpp
- Permits use of many Matlab and Octave scripts from R
- Package on CRAN, builds on all major OSs
- Package has a few demos, including the Kalman filtering example from the **RcppArmadillo** vignette / paper

RcppOctave example: The Gibbs Sampler

```
library(RcppOctave)

Mgibbs <- OctaveFunction('
  function mat = Mgibbs(N, thin)
    mat = zeros(N, 2);
    x = 0;
    y = 0;
    for i = 1:N
      for j = 1:thin
        x = randg(3) / (y*y+4);
        y = randn(1)*1/sqrt(2*(x+1)) + 1/(x+1);
      end
      mat(i,:) = [ x, y ];
    end
  end
')
```

Use Boost Python to embed Python in C++

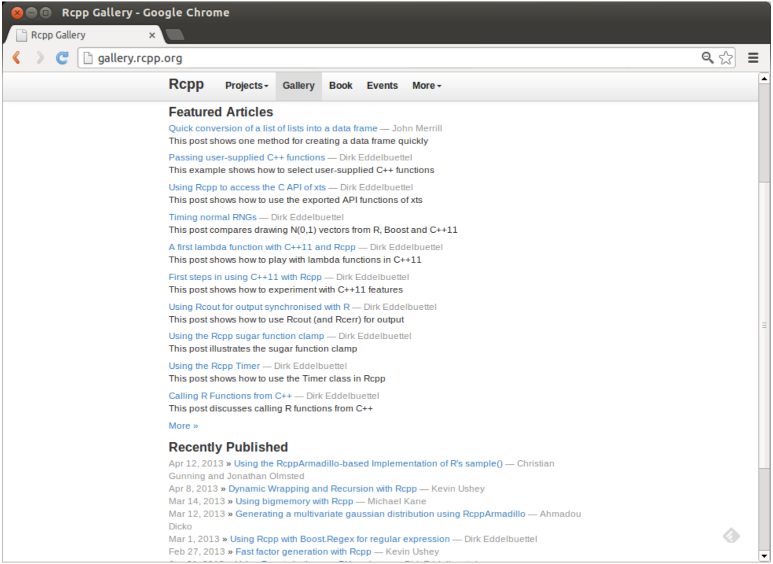
- Contribution by Wush Wu to the Rcpp Gallery
- Uses Rcpp to call C++, and Boost Python to embed Python
- Builds fine on Ubuntu, some porting work may be needed for other platforms but *should* work on other Linux variants and OS X.
- See Gallery article for details.
- At present a powerful proof-of-concept, could be generally useful.

Outline

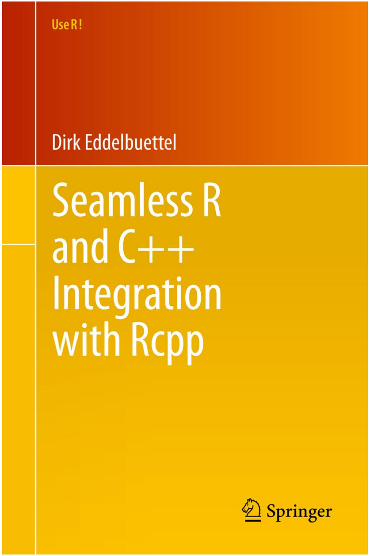
Documentation

- The **Rcpp** package comes with **eight pdf vignettes**, and numerous help pages.
- The introductory vignettes are now **published** (Rcpp and RcppEigen in *J Stat Software*, RcppArmadillo in *Comp. Stat.& Data Anal.*).
- The **rcpp-devel** list is *the* recommended resource, generally very helpful, and fairly low volume.
- **StackOverflow** is at almost 500 **Rcpp** posts.
- And a number of **blog posts** introduce/discuss features.
- Plus ...

Rcpp Gallery



The Rcpp book



Available since June 2013