RcppOctave: Running Octave from R

by Dirk Eddelbuettel and Renaud Gaujoux

Abstract The **RcppOctave** package connects Octave to R, allowing R user to access another numerical computing language from within R. TODO: Expand

Introduction

Octave (Eaton et al., 2008) is an interactive language that is primarily intended for numerical computations. It is mostly compatible with Matlab (MATLAB, 2010) and has found widespread adoption across different disciplines. This include domains in which Matlab has historically been pre-eminent such as applied mathematics, electrical engineering and signal processing, but also in other fields such as machine learning, bioinformatics, and finance. Consequently, a large corpus of application programs are available as `.m`-files (named after the commonly-chose file extension) for which Octave provides an open source engine.

R (R Core Team, 2012), a language and environment for statistical computing and graphics, has become the dominant language for statistical research, and a widely-used environment for empirical work in a variety of fields.

While both languages share commonalities, their respective focus is different making a combination of both environments an even more compelling choice. This short paper illustrates the **RcppOctave** package by Gaujoux (2012) which implements an interface between both these environments.

Octave

Octave (Eaton et al., 2008) is a powerful interactive language "not unlike Matlab". Octave can do arithmetic for real, complex or integer-valued scalars and matrices, solve sets of nonlinear algebraic equations, integrate functions over finite and infinite intervals, and integrate systems of ordinary differential and differential-algebraic equations. Given the popularity of Matlab, Octave has always had a lot of appeal in fields in which Matlab is popular.

As a simple illustration, consider the function below which converts two arguments 'distance' (in miles) and 'time' (in minutes and seconds) into a pace expressed as minutes-per-mile:

```
## usage:   p = pace (dist, time)
function p = pace (dist, time)
  if (nargin < 2)
    usage("Call as pace (dist, time)")
  end

  ## total seconds
  a = floor(time)*60 + (time-floor(time))*100;
  ## seconds per miles
  b = a / dist;
  ## minutes per mile
  c = floor(b/60);
  ## seconds per mile
  d = b - c*60;

  p = c + d/100;
  printf("%.2f over %.2f is %d:%d aka %.2f\n",
         time, dist, c, d, p);
endfunction
```

The example shows a few subtle difference between Octave and Matlab. Comments starts with hash sign. Default function help is provided before the function definition. The `endfunction` keyword ends a function.

It also shows how the semicolon at the end of lines is used to suppress output; expression not ending in a semi-colon print their final result.

Octave is particularly useful for linear algebra and calculations involving matrices and vectors. For multiplication, the `*` symbol is overloaded. For example, for a row-vector a, the expressions `a * a'` amd `a' * a` compute, respectively the (scalar) inner product and matrix outer product as the apostrophe invokes a tranposition. Common matrix functions such as `eig` or `det` are available as well. For more complete example in provided in the next section.

Example: Kalman Filter

Eddelbuettel and Sanderson (2013) introduce the **RcppArmadillo** package and illustrate it via an example comparing a Kalman filter implementation in both R and C++. As the code underlying this example was initially published for Matlab[1], it can of course also be used with RcppOctave.

TODO: Few words about the example

```
function Y = kalmanM(pos)
  dt=1;
  %% Initialize state transition matrix
  A=[ 1 0 dt 0 0 0;...      % [x  ]
      0 1 0 dt 0 0;...      % [y  ]
      0 0 1 0 dt 0;...      % [Vx]
      0 0 0 1 0 dt;...      % [Vy]
      0 0 0 0 1 0 ;...      % [Ax]
      0 0 0 0 0 1 ];        % [Ay]
```

[1]See `http://www.mathworks.com/products/matlab-coder/demos.html?file=/products/demos/shipping/coder/coderdemo_kalman_filter.html`.

```
% Initialize measurement matrix
H = [ 1 0 0 0 0 0; 0 1 0 0 0 0 ];

Q = eye(6);
R = 1000 * eye(2);

x_est = zeros(6, 1);
p_est = zeros(6, 6);

numPts = size(pos,1);
Y = zeros(numPts, 2);

for idx = 1:numPts
  z = pos(idx, :)';

  %% Predicted state and covariance
  x_prd = A * x_est;
  p_prd = A * p_est * A' + Q;
  %% Estimation
  S = H * p_prd' * H' + R;
  B = H * p_prd';
  klm_gain = (S \ B)';
  %% Estimated state and covariance
  x_est = x_prd + klm_gain * (z - H * x_prd);
  p_est = p_prd - klm_gain * H * p_prd;
  %% Compute the estimated measurements
  Y(idx, :) = H * x_est;
end                 % of the function
end    % of the function
```

This function, along with several R implementations, is provided in the **RcppOctave** package as `demo(Kalman)`. Table 1 summaries the performance.

Implementation	Time in sec.	Rel. to best
KalmanOctave	1.93	1.00
KalmanRfunC	4.99	2.59
KalmanRC	5.13	2.66
KalmanRfun	5.45	2.83
KalmanR	5.46	2.83
FirstKalmanRC	6.41	3.32
FirstKalmanR	6.84	3.54

Table 1: Performance comparison of various implementations of a Kalman filter. KalmanOctave is the **RcppOctave** based implementation shown above. KalmanR is the Rimplementation using an environment; KalmanRfun use a function; FirstKalmanR is a direct translation of the original Matlab implementation; versions ending in C are the byte-compiled variants of the respective version. Timings are averaged over 100 replications. The comparison was made using R version 2.15.2, Rcpp version 0.10.2 and RcppOctave version 0.8.14 on Ubuntu 12.10 running in 64-bit mode on a 2.67 GHz Intel i7 processor.

TODO: More about the example, performance comparison, knocking socks off R

Example: Gibbs Sampler

Wilkinson (TODO: Reference from his repeated blog posts) used a simple bivariate Gibbs Sampler as a basis for comparisons between different programming languages such as C, Java, Python and R. His example has been re-used in number of other presentations.

We can adapt this example here as it provides a suitable framework for showing how **RcppOctave** can interact with the random number generators in R.

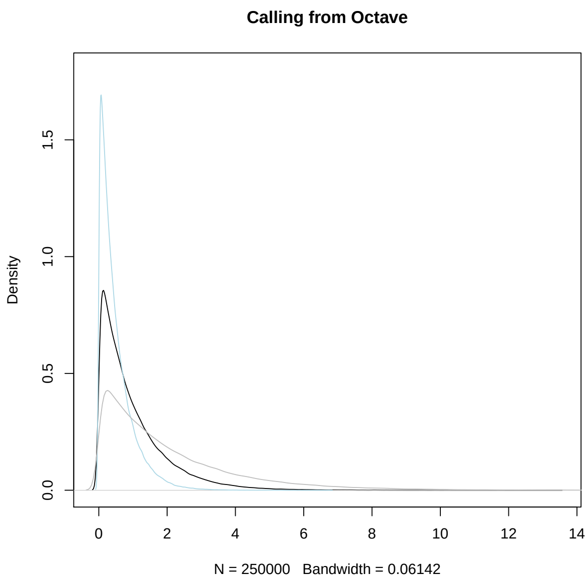
TODO: Make true Gibbs sampler example

```
library(RcppOctave)
set.seed(1)      # reset the R RNG
a <- runif(10)

# define octave function
o_source(text="
   function [x] = orng(n)
      x = rand(1, n);
   end
")
set.seed(1)      # reset the R RNG
identical(.O$orng(10), a)

## [1] TRUE
```

In this example, we first draw ten random number distributed $U(0,1)$ using the default generator in R. We then define an Octave function from R. Similar to how the **inline** package (Sklyar et al., 2012) handles C or C++ code, we simply define a character string which then passed the Octave interpreter via the `o_source()` function. We re-set the seed of the underlying RNG in R and call our Octave function. As it actually calls back into the RNG from R, our two vectors are the same. This is a critically important feature to ensure reproducibility.

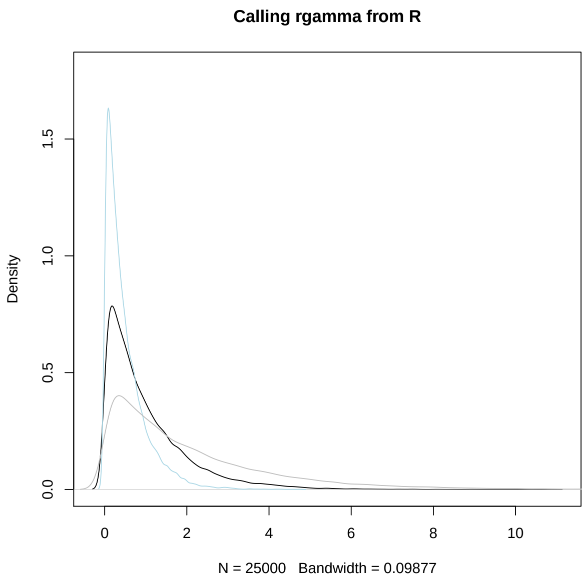
We can also define functions which draws from the Gamma distribution:

```
## define a Gamma RNG draw function
o_source(text="
  function x = orngg(n)
    x = randg(1, n);
  end
")
set.seed(42)
## 'flatten' 500x500 matrix into vector
## we have  'randg(a, N)/b' == rgamma(a,b)
x1 <- as.vector(matrix(.O$orngg(500), ncol=1))
x2 <- as.vector(matrix(.O$orngg(500), ncol=1))/2
x3 <- as.vector(matrix(.O$orngg(500), ncol=1))*2
plot(density(x1),
     ylim=c(0,1.8), main="Calling from Octave")
lines(density(x2), col='lightblue')
lines(density(x3), col='gray')
```



As **RcppOctave** ensures that Octave calls back into the R random-number generator, we can generate the same chart directly from R:

```
set.seed(42)  # reset RNG
N <- 25000
plot(density(rgamma(N,1,1)),
     ylim=c(0,1.8), main="Calling rgamma from R")
lines(density(rgamma(N,1,2)), col='lightblue')
lines(density(rgamma(N,1,1/2)), col='gray')
```



Bibliography

J. W. Eaton, D. Bateman, and S. Hauberg. *GNU Octave Manual Version 3*. Network Theory Limited, 2008. ISBN 0-9546120-6-X.

D. Eddelbuettel and C. Sanderson. RcppArmadillo: Easily extending R with high-performance C++ code. Forthcoming in *Computational Statistics and Data Analysis*, 2013.

R. Gaujoux. *RcppOctave: Seamless Interface to Octave – and Matlab*, 2012. URL http://CRAN.R-Project.org/package=RcppOctave. R package version 0.8.5.

MATLAB. *Version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL http://www.R-project.org/. ISBN 3-900051-07-0.

O. Sklyar, D. Murdoch, M. Smith, D. Eddelbuettel, and R. François. *inline: Inline C, C++, Fortran function calls from R*, 2012. URL http://CRAN.R-Project.org/package=inline. R package version 0.3.10.

Dirk Eddelbuettel
Debian Project
River Forest, IL
USA
edd@debian.org

Renaud Gaujoux
Computational Biology
University of Cape Town
Cape Town
South Africa
renaud@cbio.uct.ac.za