**RcppOctave:** Seamless Interface to Octave – And Matlab

Renaud Gaujoux

*RcppOctave* package – Version 0.18.1 [October 6, 2015]*

**Abstract**

The *RcppOctave* package provides a direct interface to *Octave* from *R*. It allows octave functions to be called from an *R* session, in a similar way C/C++ or *Fortran* functions are called using the base function `.Call`. Since *Octave* uses a language that is mostly compatible with Matlab®, *RcppOctave* may also be used to run Matlab m-files. This package was originally developed to facilitate the port and comparison of *R* and Matlab code. In particular, it provides *Octave* modules that redefine *Octave* default random number generator functions, so that they call *R* own dedicated functions. This enables to also reproduce and compare stochastic computations.

**Contents**

1 Introduction ........................ 1
2 Objectives & Features .................. 2
2.1 Linux ............................. 3
2.2 Windows ............................ 3
2.3 Mac OS ............................. 3
4 Accessing Octave from *R* .............. 4
4.1 Core interface: `.CallOctave` ...... 4
4.1.1 Overview .......................... 4
4.1.2 Controlling output values ........ 4
4.1.3 Examples ........................ 5
4.2 Direct interface: the `.O` object ... 8
4.2.1 Manipulating variables .......... 9
4.2.2 Calling functions ............... 9
4.2.3 Auto-completion ............... 10
4.3 Utility functions .................. 10
5 Calling *R* functions from *Octave* ... 15
6 Examples ............................ 15
6.1 Comparing implementations ......... 15
6.2 Random computations ............. 16
7 Known issues ........................ 17
8 News and changes ................... 17
9 References .......................... 19

1 Introduction

In many research fields, source code of algorithms and statistical methods are published as Matlab files (the so called m-files). While such code is generally released under public Open Source licenses like the GNU Public Licenses (GPLs) [3], effectively running or using it require

*This vignette was built using *Octave* 3.8.1*
either to have Matlab®, which is a nice but expensive proprietary software¹, or to be/get – at least – a bit familiar with Octave [1], which is free and open source, and is able to read and execute m-files, as long as they do not require Matlab-specific functions. However, R users may have neither Matlab license, nor the time/will to become Octave-skilled, and yet want to use algorithms written in Matlab/Octave for their analyses and research.

Being able to run m-files or selectively use Octave functionalities directly from R can greatly alleviate a process that otherwise typically implies exporting/importing data between the two environments via files on disk, as well as dealing with a variety of issues including rounding errors, format compatibility or subtle implementation differences, that all may lead to intricate hard-to-debug situations. Even if one eventually wants to rewrite or optimise a given algorithm in plain R or in C/C++, and therefore remove any dependency to Octave, it is important to test the correctness of the port by comparing its results with the original implementation. Also, a direct interface allows users to stick to their preferred computing environment, in which they are more comfortable and productive.

An R package called ROctave ² does exist, and intends to provide an interface between R and Octave, but appears to be outdated (2002), and does not work out of the box with recent version of Octave. A more recent forum post³ brought back some interest on binding these two environments, but apparently without any following.

The RcppOctave package⁴ described in this vignette aims at filling the gap and facilitating the usage of Octave/Matlab code from R, by providing a lean interface that enables direct and easy interaction with an embedded Octave session. The package’s name was chosen both to differentiate it from the existing ROctave package, and to reflect its use and integration of the C++ framework defined by the Rcpp package⁵.

2 Objectives & Features

The ultimate objective of RcppOctave is to provide a two-way interface between R and Octave, i.e. that allows calling Octave from R and vice-versa. The interface intends to be lean and as transparent as possible, as well as providing convenient utilities to perform commonly needed tasks (e.g. source files, browse documentation).

Currently, the package focuses on accessing Octave functionalities from R with:

- An out-of-the-box-working embedded Octave session;
- Ability to run/source m-files from R;
- Ability to evaluate Octave statements and function calls from R;
- Ability to call R functions in Octave code⁶;
- Transparent passage of variables between R and Octave;
- Reproducibility of computations, including stochastic computations, in both environment;

Future development should provide similar reverse capabilities, i.e. an out of the box embedded R session, typically via the RInside package⁷.

¹http://www.mathworks.com
²http://www.omegahat.org/ROctave
⁴http://cran.r-project.org/package=RcppOctave
⁵http://cran.r-project.org/package=Rcpp
⁶Currently only when run from R through RcppOctave.
⁷http://cran.r-project.org/package=RInside
3 OS requirements

The package has been developed and tested under Linux (Ubuntu), and has notably been reported to work fine on other Linux distributions. Developments to make it run on Windows and Mac recently started, and has been so far relatively successful.

3.1 Linux

The only requirement on Linux machines is to have \textit{Octave} $\geq 3.2.4$ and its development files installed, although a more recent version ($\geq 3.6$) is recommended to get full functionalities.

On Debian/Ubuntu this amounts to:

\textbf{Octave $\geq 3.6$:} (works out of the box):

\begin{verbatim}
# install octave and development files
sudo apt-get install octave liboctave-dev
# install as usual in R
Rscript -e "install.packages('RcppOctave')"
\end{verbatim}

\textbf{Octave 3.2.4:} (might require extra command)

\begin{verbatim}
# install octave and development files
sudo apt-get install octave3.2 octave3.2-headers
# requires to explicitly export Octave lib directory
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:`octave-config -p OCTLIBDIR`
# install as usual in R
Rscript -e "install.packages('RcppOctave')"
\end{verbatim}

3.2 Windows

Support for Windows started with version 0.11. Developments and tests are performed on Windows 7 using the following settings:

\textbf{Rtools:} the package contains C++ source files that need to be compiled, which means that \textit{Rtools} needs to be installed, with its bin/ sub-directory in the system PATH.

See \url{http://cran.r-project.org/bin/windows/Rtools/} for how to install the version of \textit{Rtools} compatible with your \textit{R} version;

\textbf{Octave:} development was performed using the \textit{mingw} version of Octave, which can be installed as described in the Octave wiki:

\url{http://wiki.octave.org/Octave_for_Windows#Octave-3.6.4-mingw_.2B_octaveforge_pkgs}

Octave binary bin/ sub-directory (e.g., C:\Octave\Octave3.6.4\gcc4.6.2\bin) must be in the system PATH as well, preferably after \textit{Rtools} own bin/ sub-directory.

3.3 Mac OS

Support for Mac OS is not yet official, but is currently being investigating. Discussion(s) on how to install and run under Mac can be found here:

\url{http://lists.r-forge.r-project.org/pipermail/rcppoctave-user/2013-October/000024.html} and \url{https://github.com/renozao/RcppOctave/issues/1}

The main installation procedure is based on the Octave version provided by \textit{homebrew}\footnote{\url{http://brew.sh/}}, which works with the binary build provided by CRAN\footnote{\url{http://cran.r-project.org/bin/macosx/}}:

\begin{verbatim}
\end{verbatim}
1. install XCode and its Command Line Tools
2. install homebrew
3. add the homebrew/science repository (tap in brewing language):
   
   ```
   brew tap homebrew/science
   brew update && brew upgrade
   brew tap --repair #may not be necessary
   brew install gfortran
   brew install octave
   ```

4. Accessing Octave from R

   The RcppOctave package defines the function `.CallOctave`, which acts as a single entry point for calling Octave functions from R. In order to make common function calls easier (e.g. `eval`), other utility functions are defined, which essentially wraps a call to `.CallOctave`, but enhance argument handling and result formatting.

4.1 Core interface: `.CallOctave`

   The function `.CallOctave` calls an Octave function from R, mimicking the way native C/C++ functions are called with `.Call`.

4.1.1 Overview

   The function `.CallOctave` takes the name of an Octave function (in its first argument `.NAME`) and pass the remaining arguments directly to the Octave function – except for the two special arguments `argout` (see next section) and `unlist`. Note that Octave function arguments are not named and positional, meaning that they must be passed in the correct order. Input names are simply ignored by `.CallOctave`. Calling any Octave function is then as simple as:

   ```
   .CallOctave('version')
   ## [1] "3.8.1"
   .CallOctave('sqrt', 10)
   ## [1] 3.162278
   .CallOctave('eye', 3)
   ## [,1] [,2] [,3]
   ## [1,] 1 0 0
   ## [2,] 0 1 0
   ## [3,] 0 0 1
   .CallOctave('eye', 3, 2)
   ## [,1] [,2]
   ## [1,] 1 0
   ## [2,] 0 1
4.1.2 Controlling output values

Octave functions have the interesting feature of being able to compute and return a variable number of output values, depending on the number of output variables specified in the statement. Hence, a call to an Octave function requires passing both its parameters and the number of desired output values.

The following sample code illustrates this concept using the function svd\(^{10}\):

```octave
% single output variable: eigen values only
S = svd(A);

% 3 output variables: complete SVD decomposition
[U, S, V] = svd(A);
```

The default behaviour of .CallOctave is to try to detect the maximum number of output variables, as well as their names, and return them all. This should be suitable for most common cases, especially for functions defined by the user in plain m-files, but does not work for functions defined in compiled modules (see examples with in the next section). Hence the default is to return the maximum number of output values if it can be detected, or only the first one.

For some functions, however, this behaviour may not be ideal, and complete control on the return values is possible via the special argument `argout`. The next section illustrates different situations and use case scenarios.

4.1.3 Examples

A sample m-file (i.e. a function definition file) is shipped with any RcppOctave installation in the “scripts/” sub-directory and provides some examples of different types of Octave functions:

```octave
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Example file for the R package RcppOctave
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [a] = fun1()
a = rand(1,4);
end

function [a,b,c] = fun2()
a = rand(1,4);
b = rand(2,3);
c = "some text";
end

function fun_noargout(x)
    % no effect outside the function
    y = 1;
    printf("% Printed from Octave: x="), disp(x);
end

function [s] = fun_varargin(varargin)
    if (nargin==0)
        s = 0;
    else
        s = varargin{1} + varargin{2} + varargin{3};
    end

This sample code is extracted from the manpage for svd. See `help(svd)` for more details.
```
function [u, s, v] = fun_varargout()

    if (nargout == 1) u = 1;
    elseif (nargout == 3)
        u = 10; s = 20; v = 30;
    else usage("Expecting 1 or 3 output variables.");
    endif;
end

These definitions can be loaded in the Octave session via the function sourceExamples.

# source example function definitions from RcppOctave installation
sourceExamples('ex_functions.m')
# several functions are now defined
o_ls()

## [1] "fun1" "fun2" "fun_noargout" "fun_varargin"

The functions fun1, fun2, fun_noargout, and fun_varargin perform the same computations independently of the number of output. For these a default call to .CallOctave is enough to get their full functionalities:

# single output value
.CallOctave('fun1')

## [1] 0.6776693 0.7570356 0.6148638 0.3308931

# 3 output values
.CallOctave('fun2')

## $a
## [1] 0.8452558 0.4754382 0.4435868 0.2553756
##
## $b
## [,1] [,2] [,3]
## [1,] 0.4847515 0.9711548 0.7379941
## [2,] 0.4536776 0.9089370 0.9022478
##
## $c
## [1] "some text"

# no output value
.CallOctave('fun_noargout', 1)

## % Printed from Octave: x= 1

.CallOctave('fun_noargout', 'abc')

## % Printed from Octave: x=abc
# variable number of arguments
.CallOctave('fun_varargin')

## [1] 0
.CallOctave('fun_varargin', 1, 2, 3)

## [1] 6

The function `fun_varargout` however, behaves differently when called with 1, 2 or 3 output variables, performing different computations. Since it is defined in a m-file, the maximum set of output variables is detectable and the default behaviour is then to call it asking for 3 output variables. The other types of computations can be obtained using argument `argout`:

.CallOctave('fun_varargout')

## $u
## [1] 10
##
## $s
## [1] 20
##
## $v
## [1] 30
.CallOctave('fun_varargout', argout=1)

## [1] 1

# this should throw an error
try( .CallOctave('fun_varargout', argout=2) )

## Error in .CallOctave("fun_varargout", argout = 2): RcppOctave - error in Octave function 'fun_varargout':
## usage: Expecting 1 or 3 output variables.
## fun_varargout at line 34, column 7

Argument `argout` may also be used to specify names for the output values. This is useful for functions defined in compiled modules (e.g. `svd`) for which expected outputs are not detectable (output names in particular), or when limiting the number of output variables in functions defined in m-files. Indeed, in this latter case, it is not safe to infer the names based on those defined for the complete output, as these may not be relevant anymore:

# single output variable: result is S
.CallOctave('svd', matrix(1:4, 2))

## [,1]
## [1,] 5.4649857
## [2,] 0.3659662

# 3 output variables: results is \([U,S,V]\)
.CallOctave('svd', matrix(1:4, 2), argout=3)

## [[1]]
## [,1] [,2]
# specifies output names (and therefore number of output variables)
.CallOctave('svd', matrix(1:4, 2), argout=c('U', 'S', 'V'))

## $U
## [,1] [,2]
## [1,] -0.576048 0.817416
## [2,] -0.817416 0.576048

## $S
## [,1] [,2]
## [1,] 5.464986 0.000000
## [2,] 0.000000 0.365966

## $V
## [,1] [,2]
## [1,] -0.404554 0.914514
## [2,] -0.914514 -0.404554

Note that it is quite possible for a compiled function to only accept calls with at least 2 output variables. In such cases, .CallOctave calls must always specify argument argout.

### 4.2 Direct interface: the .O object

An alternative and convenient shortcut interface is defined by the S4-class Octave. At load time, an instance of this class, an object named .O, is initialised and exported from RcppOctave's namespace. Using the .O object, calls to Octave functions are more compact:

.O
## <Octave Interface>
## - Use `$x` to call Octave function or get variable x.
## - Use `$x <- val` to assign a value val to the Octave variable x.

.O$version()
## [1] "3.8.1"

.O$eye(3)

## [,1] [,2] [,3]
## [1,] 1 0 0
## [2,] 0 1 0
## [3,] 0 0 1
4.2.1 Manipulating variables

The .O object facilitates manipulating single Octave variables, as it emulates an R environment-like object whose elements would be the objects available in the current Octave embedded session:

```r
# define a variable
.O$myvar <- 1:5
# retrieve value
.O$myvar
## [1] 1 2 3 4 5

# assign and retrieve new value
.O$myvar <- 10
.O$myvar
## [1] 10

# remove
.O$myvar <- NULL
# this should now throw an error since 'myvar' does not exist anymore
try( .O$myvar )
## Error: RcppOctave:o_get - Could not find an Octave object named 'myvar'.
```

4.2.2 Calling functions

As illustrated above, Octave functions can be called through the .O object, by passing specifying its arguments as a function call:
# density of x=5 for Poisson(2)
.O$poisspdf(5, 2)

## [1] 0.03608941

# E.g. compare with R own function
dpois(5, 2)

## [1] 0.03608941

They may also be retrieved as R functions in a similar way as variables, and called in subsequent statements:

# retrieve Octave function
f <- .O$poisspdf

# call (in Octave)
f(5, 2)

## [1] 0.03608941

4.2.3 Auto-completion

An advantage of using the .O object is that it has auto-completion capabilities similar to the R console. This greatly helps and speeds up the interaction with the current embedded Octave session. For example, typing .O$std + TAB + TAB will show all functions or variables available in the current session, that start with “std”.

4.3 Utility functions

The RcppOctave package defines some utilities to enhance the interaction with Octave, and alleviate calls to a set of commonly used Octave functions. All these functions start with the prefix “o” (e.g. o_source), so that they can be listed by typing o_ + TAB + TAB in the R console. Their names have been chosen to reflect the corresponding Octave function, and, in some cases, aliases matching standard R names are also provided, so that users not familiar with Octave can find their way quickly (e.g. o_rm is an alias to o_clear).

4.3.1 Assign/get variables

The functions o_assign and o_get facilitates assigning variables and retrieving objects (variables or functions). Variables may be assigned or retrieved individually in separate calls to o_assign or o_get\(^\text{11}\), or simultaneously in a variety of ways (see ?o_get for more details and examples):

## ASSIGN
o_assign(a=1)
o_assign(a=10, b=20)
o_assign(list(a=5, b=6, aaa=7, aab=list(1,2,3)))

## GET

\(^{11}\)This would be similar to using the .O object as described above
```r
# get all variables
str( o_get() )
## List of 4
## $ a : num 5
## $ aaa: num 7
## $ aab: List of 3
## ..$ : num 1
## ..$ : num 2
## ..$ : num 3
## $ b : num 6

# selected variables
o_get('a')
## [1] 5
o_get('a', 'b')
## $a
## [1] 5
## $b
## [1] 6

# rename on the fly
o_get(c='a', d='b')
## $c
## [1] 5
## $d
## [1] 6

# o_get throw an error for objects that do not exist
try( o_get('xxxxx') )
## Error: RcppOctave::o_get - Could not find an Octave object named 'xxxxx'.
# but suggests potential matches
try( o_get('aa') )
## Error: RcppOctave::o_get - Could not find an Octave object named 'aa'.
## Match(es):  aaa aab

# get a function
f <- o_get('svd')
f
## <OctaveFunction::svd>
```

### 4.3.2 Evaluate single statements

To evaluate a single statement, one can use the `o_eval` function, that can also evaluate a list of statements sequentially:
# assign variable 'a'
o_eval("a=1")
## [1] 1
o_eval("a") # or .O$a
## [1] 1
o_eval("a=svd(rand(3))")
## [,1]  
## [1,] 1.0439442  
## [2,] 0.4677600  
## [3,] 0.1736197  
.O$a
## [,1]  
## [1,] 1.0439442  
## [2,] 0.4677600  
## [3,] 0.1736197
# eval a list of statements
l <- o_eval("a=rand(1, 2)", "b=randn(1, 2)", "rand(1, 3)")
l
## [[1]]  
## [1] 0.4794507 0.5119345  
## [2] -1.043102 -1.678494  
## [3] 0.8126957 0.9962844 0.1317312
# variables 'a' and 'b' were assigned the new values
identical(list(.O$a, .O$b), l[1:2])
## [1] TRUE
# multiple statements are not supported by o_eval
try(o_eval("a=1; b=2") )
## Error in FUN(X[[i]], ...) : RcppOctave - error in Octave function 'eval':  
## eval: invalid use of statement list
.O$a
## [1] 0.4794507 0.5119345
# argument CATCH allows for recovering from errors in statement
o_eval("a=usage('ERROR: stop here')", CATCH="c=3")
## [1] 3
.O$a
## [1] 0.4794507 0.5119345
.O$c
## [1] 3
More details and examples are provided in the manual page ?o_eval. If more than one statement is to be evaluated, then one should use the function o_source, with argument text as described in Section 4.3.3 below.

4.3.3 Source m-files

Octave/Matlab code generally are generally provided as so called m-files, which are plain text files that contain function definitions and/or sequences of multiple commands that perform a given task. This is the form most public third party algorithms are published.

The function o_source allows to load these files in the current Octave session, so that the object they define are available, or the commands they contain are executed. RcppOctave ships an example m-file in the “scripts/” sub-directory of its installation:

```octave
# clear all session
o_clear(all=TRUE)
o_ls()

## character(0)

# source example file from RcppOctave installation
mfile <- system.file("scripts/ex_source.m", package='RcppOctave')
cat(readLines(mfile), sep="\n")

## Example m-file to illustrate the usage of the function o_source
## %
## % This file defines 3 dummy variables ('a', 'b' and 'c')
## % and a dummy function 'abc', that adds up its three arguments.
## %
## %
## % a = 1;
## % b = 2;
## % c = 3;
## %
## % function [res] = abc(x, y, z)
## % res = x + y + z;
## % end

o_source(mfile)
# Now objects 'a', 'b', and 'c' as well as the function 'abc'
# should be defined:
o_ls(long=TRUE)

## <Octave session: 4 object(s)>
## name size bytes class global sparse complex nesting persistent
## a 1x1 8 double FALSE FALSE FALSE 1 FALSE
## b 1x1 8 double FALSE FALSE FALSE 1 FALSE
## c 1x1 8 double FALSE FALSE FALSE 1 FALSE
## abc NA NA function TRUE NA NA 1 NA

#
o_eval("abc(2, 4, 6)"")
# [1] 12

o_eval("abc(a, b, c)"")
# [1] 6
```
This function can also conveniently be used to evaluate multiple statements directly passed from the R console as character strings via its argument `text`:

```octave
o_source(text="clear a b c; a=100; a*sin(123)"
# last statement is stored in automatic variable 'ans'
o_get('a', 'ans')
```

```r
## $a
## [1] 100
##
## $ans
## [1] -45.99035
```

### 4.3.4 List objects

The function `o_ls` (as used above) lists the objects (variables and functions) that are defined in the current Octave embedded session. It is an enhanced version over Octave standard listing functions such as `who` (see `?o_who`), which only lists variables, and not user-defined functions. With argument `long` it returns details about each variable and function, in a similar way `whos` does (see `?o_whos`).

```octave
o_ls()
```

```octave
## [1] "a" "abc"
o_ls(long=TRUE)
```

```octave
## <Octave session: 2 object(s)>
## name size bytes class global sparse complex nesting persistent
## a 1x1 8 double FALSE FALSE FALSE 1 FALSE
## abc NA NA function TRUE NA NA 1 NA
```

#clear all (variables + functions)
o_clear(all=TRUE)
o_ls()
```

```octave
## character(0)
```

See `?o_ls` for more details as well as Section 7 for a known issue in Octave versions older than 3.6.1.

### 4.3.5 Browse documentation

Octave has offers two ways of browsing documentation, via the functions `help` and `doc`, which display a manual page for a given function and lookup the whole documentation for a given topic respectively.

The RcppOctave package provides wrapper for these two functions to enable browsing Octave help pages in the way R users are used to. Hence, to access the manpage for a given function one types for example the following, which displays using the R function `file.show`:

```octave
o_help(std)
```

To display all documentation about a topic one types for example the following, which displays using the GNU Info browser

```octave
o_help(std)
```

12At least on Linux machines.
Once the GNU Info browser is running, help for using it is available using the command ‘Ctrl + h’ – as stated in the Octave documentation for doc (see o_help(doc)).

4.4 Errors and warning handling

All i/o messages written by Octave are redirected to R own i/o functions, with errors and warnings generating corresponding messages in R:\footnote{On Windows, output redirection does not working properly and all output is “mysteriously” directly displayed by Octave.}

```r
# error
res <- try(.CallOctave('error', 'this is an error in Octave'))

## Error in .CallOctave("error", "this is an error in Octave") : RcppOctave - error
## in Octave function ‘error’:
## this is an error in Octave
geterrmessage()

# warning
res <- .CallOctave('warning', 'this is a warning in Octave')

## Warning in .CallOctave("warning", "this is a warning in Octave") : this is a warning
## in Octave
```

4.5 Low-level $C/C++$ interface

RcppOctave builds upon the Rcpp package, and defines specialisation for the Rcpp template functions Rcpp::as and Rcpp::wrap, for converting R types to Octave types and vice versa. Currently these templates are not exported, but will probably be in the future.

5 Calling R functions from Octave

This is currently under development. Interested users can find this feature under the branch feature/Rfun in the GitHub repository:

https://github.com/renozao/RcppOctave/tree/feature/Rfun

6 Examples

6.1 Comparing implementations

Comparing equivalent $R$ and Octave functions is as easy as comparing two $R$ functions. For example, one can compare the respective functions $svd$ with the following code, which defines a wrapper functions to format the output of Octave $svd$ function as $R$ (see $?svd$ and o_help(svd)):

```r
o_svd <- function(x){
  # ask for the complete decomposition
  res <- .O$svd(x, argout=c('u','d','v'))
}
```

\footnotetext{On Windows, output redirection does not working properly and all output is “mysteriously” directly displayed by Octave.}
# reformat/reorder result
res$d <- diag(res$d)
res[2, 1, 3]

# define random data
X <- matrix(runif(25), 5)

# run SVD in R
svd.R <- svd(X)

# run SVD in Octave
svd.O <- o_svd(X)
str(svd.O)

## List of 3
##$ d: num [1:5] 2.6127 0.7663 0.439 0.1784 0.0403
##$ u: num [1:5, 1:5] -0.526 -0.51 -0.303 -0.552 -0.258 ...
##$ v: num [1:5, 1:5] -0.516 -0.563 -0.369 -0.459 -0.265 ...

# check results
all.equal(svd.R, svd.O)

## [1] TRUE

# but not exactly identical
all.equal(svd.R, svd.O, tol=10^-16)

## [1] "Component \"u\": Mean relative difference: 4.093149e-16"
## [2] "Component \"v\": Mean relative difference: 4.084112e-16"

### 6.2 Random computations

In order to ensure reproducibility of results and facilitate the comparability of implementations between R and Octave, ReppOctave ships a custom Octave module that redefine Octave standard random number generator functions rand, randn, rande and randg, so that they call R corresponding functions runif, rnorm, rexp and rgamma. This module is loaded when the ReppOctave package is itself loaded. As a result, random computation – that use these functions – can be seeded in both Octave and R, using R standard function set.seed. This facilitates, in particular, the validation of ports of stochastic algorithms (e.g. simulations, MCMC-based estimations):

Rf <- function()
{
    x <- matrix(runif(100), 10)
    y <- matrix(rnorm(100), 10)
    (x * y) %*% (x / y)
}

Of <- {
# define Octave function
  o_source(text="
  function [res] = test()
  x = rand(10);
  y = randn(10);
  res = (x .* y) * (x ./ y);
  ")

7 Known issues

- In Octave versions older than 3.6.1, the function `o_ls` may not list user-defined functions. This is due to the built-in Octave function `completion_matches` that does not return them. The issue seems to have been fixed by Octave team at least in 3.6.1.

- The detection of output names by `.CallOctave` in Octave versions older than 3.4.1 does not work, meaning that Octave functions are always called with a single output variable. For obtaining more outputs, the user must specify argument `argout` accordingly.

- Redirection of Octave output sent to stdout and stderr on Windows does not work.

8 News and changes

********************
Changes in 0.18
********************
**FIXES**
- Fixed installation problem on Windows for R >= 3.2, due to toolchain incompatibility. Now requires that the 4.9.2 toolchain available from CRAN is located under the `Rtools` directory.

********************
Changes in 0.16
********************
**NEW FEATURES**
- Added Octave function `randp` for Poisson distribution (issue #8)
**FIXES**
- Fixed compatibility bug with Octave 4.0.0. See `Octave::ConfigInfo` (issue #5).
- Added configure option `--disable-profile` to disable loading in internal R calls during installation (issue #6)

********************
Changes in 0.14.5
********************
**FIXES**
- Mac support: look also for `.dylib` files when default `.so` files don't exist (issue #2)
- Mac support: make sure Octave modules and package shared object are built using the same compiler/toolchain used by R (issue #1)
- Makevars.in now has `CXX_STD = CXX11`, which gets rid of C99 warnings when compiling shared object.
Changes in 0.14.3

FIXES
- Fixed typo in rcpp_octave.h which was causing an installation error on FreeBSD: SWIF_OCTAVE_PREREQ should be SWIG_OCTAVE_PREREQ.

Changes in 0.14.2

NEW FEATURES
- Function .CallOctave gains a new argument `verbose` to specify a temporary verbose level.

FIXES
- Fixed installation issue on Windows that were introduced with new installation features in 0.14.
- Fixed bug in output redirection when linking against old octave 3.2: method str() was not defined.

Changes in 0.14

CHANGES
- Added a configure option --with-octave to specify the path to a non-standard Octave installation. For example:
  
  R CMD INSTALL --configure-args="--with-octave=/opt/octave" rcppOctave_0.14.tar.gz

- Handling of Octave outputs and error messages have been improved.

Fixes
- Some changes have been made so that the package is compatible with Octave 3.8 (reported by Bernard).
- Installation was failing when Octave was compiled with hdf5-mpi support.

Changes in 0.11

NEW FEATURES
- The package have successfully been installed on Windows machines, although only basic functionalities have been tested (see README file).
- Support for Mac have also started, although even more slightly, using the Octave version available from homebrew.

CHANGES
- Moved all developments/bug reports/static docs to GitHub repository: https://github.com/renozao/RcppOctave/
- octave functions' stdout and stderr messages are now buffered by default, so that it does not bypass R own i/o functions. All messages/warnings sent to stdout/stderr from octave are displayed on exiting the function call.
- function .CallOctave gains a new argument buffer.std to enable/disable stdout and/or stderr buffering (see ?.CallOctave).
- octave startup warnings (e.g. shadowing of core functions by octave modules) are not shown anymore.
- Minor adaptations to pass new CRAN checks
- The package now depends on R >= 3.0.0 to properly handle the knitR vignettes.
- Errors are now more properly handled, thanks to hints found in this old post by Romain François: http://lists.r-forge.r-project.org/pipermail/rcpp-devel/2010-May/000651.html

Changes in 0.9.3

CHANGES
- Conditional use of function packageRName: use the one from pkgmaker in R <= 2.15.3, or the one exported by utils in R >= 3.0.

Session information

R version 3.2.2 (2015-08-14)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 14.04.3 LTS
locale:
[1] LC_CTYPE=en_US.UTF-8 LC_NUMERIC=C
[3] LC_TIME=C LC_COLLATE=en_US.UTF-8
[5] LC_MONETARY=en_ZA.UTF-8 LC_MESSAGES=en_US.UTF-8
[7] LC_PAPER=en_ZA.UTF-8 LC_NAME=C
[9] LC_ADDRESS=C LC_TELEPHONE=C

attached base packages:
[1] methods stats graphics grDevices utils datasets base

other attached packages:
[1] knitr_1.11 RcppOctave_0.18.1 pkgmaker_0.25.10 registry_0.3
[5] Rcpp_0.12.1

loaded via a namespace (and not attached):
[1] magrittr_1.5 formatR_1.2.1 tools_3.2.2 stringi_0.5-5
[5] codetools_0.2-14 highr_0.5.1 stringr_1.0.0 digest_0.6.8
[9] xtable_1.7-4 evaluate_0.8

References


