# *RcppOctave*: Seamless Interface to Octave – and Matlab

Renaud Gaujoux

*RcppOctave* package – Version 0.7.5 [May 30, 2012]*

**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

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) [4], effectively running or using it require either to have Matlab®, which is a nice but expensive proprietary software[1], 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

*This vignette was built using *Octave* 3.6.1
[1] http://www.mathworks.com

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] do 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 post[3] brought back some interest on binding these two environments, but apparently without any following.

The *RcppOctave* package[4] [5] 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 *Rcpp* package[5] [2].

## 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* code[6];
- 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 *RInside* package[7] [3].

## 3 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 formating.

[2] http://www.omegahat.org/ROctave
[3] http://octave.1599824.n4.nabble.com/ROctave-bindings-for-2-1-73-2-9-x-td1602060.html
[4] http://cran.r-project.org/package=RcppOctave
[5] http://cran.r-project.org/package=Rcpp
[6] Currently only when run from *R* through *RcppOctave*.
[7] http://cran.r-project.org/package=RInside

## 3.1 Core interface: `.CallOctave`

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

### 3.1.1 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] "3.6.1"

.CallOctave("sqrt", 10)

## [1] 3.162

.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
## [3,]    0    0
```

### 3.1.2 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`[8]:

```
% 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` `.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.

[8] The sample code is extracted from the manpage for `svd`. See `o_help(svd)` for more details.

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.

### 3.1.3 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:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 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};
  endif
end

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"
## [5] "fun_varargout"
```

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] 0.7827 0.9582 0.9519 0.1077

# 3 output values
.CallOctave("fun2")

## $a
## [1] 0.5901 0.3329 0.9827 0.6624
##
## $b
##        [,1]   [,2]   [,3]
## [1,] 0.6873 0.7794 0.1374
## [2,] 0.9305 0.4670 0.5597
##
## $c
## [1] "some text"
##


# no output value
.CallOctave("fun_noargout", 1)
.CallOctave("fun_noargout", "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
```

```
.CallOctave("fun_varargout", argout = 2)
```

```
## Error: RcppOctave - error in Octave function `fun_varargout`.
```

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,] 5.465
## [2,] 0.366
```

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

```
## [[1]]
##         [,1]    [,2]
## [1,] -0.5760 -0.8174
## [2,] -0.8174  0.5760
##
## [[2]]
##       [,1]  [,2]
## [1,] 5.465 0.000
## [2,] 0.000 0.366
##
## [[3]]
##         [,1]    [,2]
## [1,] -0.4046  0.9145
## [2,] -0.9145 -0.4046
##
```

```
# specify output names (and therefore number of output variables)
.CallOctave("svd", matrix(1:4, 2), argout = c("U", "S", "V"))
```

```
## $U
##         [,1]    [,2]
## [1,] -0.5760 -0.8174
## [2,] -0.8174  0.5760
```

```
##
## $S
##       [,1]  [,2]
## [1,] 5.465 0.000
## [2,] 0.000 0.366
##
## $V
##         [,1]    [,2]
## [1,] -0.4046  0.9145
## [2,] -0.9145 -0.4046
##
```

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`.

## 3.2 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] "3.6.1"
```

```
.O$eye(3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```
.O$svd(matrix(1:4, 2))
```

```
##       [,1]
## [1,] 5.465
## [2,] 0.366
```

```
# argout can still be specified
.O$svd(matrix(1:4, 2), argout = 3)
```

```
## [[1]]
##         [,1]    [,2]
## [1,] -0.5760 -0.8174
## [2,] -0.8174  0.5760
##
## [[2]]
##       [,1]  [,2]
## [1,] 5.465 0.000
## [2,] 0.000 0.366
```

```
##
## [[3]]
##         [,1]    [,2]
## [1,] -0.4046  0.9145
## [2,] -0.9145 -0.4046
##
```

### 3.2.1 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:

```
# 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$myvar <- NULL
.O$myvar

## Error: RcppOctave::o_get - Could not find an Octave object named 'myvar'.
```

### 3.2.2 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] 0.03609

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

## [1] 0.03609
```

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
show(f)

## <OctaveFunction::`poisspdf`>
```

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

## [1] 0.03609
```

### 3.2.3 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".

## 3.3 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`).

### 3.3.1 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`[9], 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 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")
```

[9] This would be similar to using the `.O` object as described above

```
## $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
o_get("xxxxx")

## Error: RcppOctave::o_get - Could not find an Octave object named 'xxxxx'.

# but suggests potential matches
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")
show(f)

## <OctaveFunction::`svd`>
```

### 3.3.2 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,] 1.6488
## [2,] 0.5735
## [3,] 0.1178
```

```
.O$a
```

```
##         [,1]
## [1,] 1.6488
## [2,] 0.5735
## [3,] 0.1178
```

```
# eval a list of statements
l <- o_eval("a=rand(1, 2)", "b=randn(1, 2)", "rand(1, 3)")
l
```

```
## [[1]]
## [1] 0.6600 0.8139
##
## [[2]]
## [1] -1.076  1.280
##
## [[3]]
## [1] 0.2209 0.9951 0.4124
##
```

```
# 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
o_eval("a=1; b=2")
```

```
## Error: RcppOctave - error in Octave function `eval`.
```

```
.O$a
```

```
## [1] 0.6600 0.8139
```

```
# argument CATCH allows for recovering from errors in statement
o_eval("a=usage('ERROR: stop here')", CATCH = "c=3")
```

```
## [1] 3
```

```
.O$a
```

```
## [1] 0.6600 0.8139
```

```
.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 3.3.3 below.

### 3.3.3 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:

```
# 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`:

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

## $a
## [1] 100
##
## $ans
## [1] -45.99
##
```

### 3.3.4 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_who`).

```
o_ls()

## [1] "a"   "abc"

o_ls(long = TRUE)

##  <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()

## character(0)
```

See `?o_ls` for more details as well as Section 6 for a known issue in *Octave* versions older than 3.6.1.

### 3.3.5 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`:

```
o_help(std)
```

To display all documentation about a topic one types for example the following, opens the documentation using the GNU Info browser[10]:

```
o_doc(poisson)
```

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)`).

### 3.4 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.

## 4 Calling R functions from Octave

This is currently not implemented but is on the TODO list for future developments.

## 5 Examples

### 5.1 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)`):

```
o_svd <- function(x) {
    # ask for the complete decomposition
    res <- .O$svd(x, argout = c("u", "d", "v"))
    # reformat/reorder result
    res$d <- diag(res$d)
    res[c(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] 2.603 0.903 0.817 0.343 0.276
##  $ u: num [1:5, 1:5] -0.504 -0.51 -0.417 -0.468 -0.305 ...
##  $ v: num [1:5, 1:5] -0.309 -0.235 -0.475 -0.483 -0.625 ...

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

## [1] TRUE
```

[10] At least on Linux machines.

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

## [1] "Component 2: Mean relative difference: 3.653e-16"
## [2] "Component 3: Mean relative difference: 2.4e-16"
```

## 5.2 Random computations

In order to ensure reproducibility of results and facilitate the comparability of implementations between *R* and *Octave*, *RcppOctave* 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 *RcppOctave* 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);
end
")
# return the function
.O$test
}

# run both computations with a common seed
set.seed(1234); res.R <- Rf()
set.seed(1234); res.O <- Of()
# compare results
identical(res.R, res.O)

## [1] TRUE


# not seeding the second computation would give different results
set.seed(1234);
identical(Rf(), Of())

## [1] FALSE
```

# 6 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.
- Errors and warnings thrown by *Octave* do not show up in Sweave documents processed using the *knitr* package[11] [6] – like this vignette. The issue needs further investigation.

# Session information

```
R version 2.15.0 (2012-03-30)
Platform: x86_64-pc-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_ZA.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_ZA.UTF-8        LC_COLLATE=en_ZA.UTF-8
 [5] LC_MONETARY=en_ZA.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=C                 LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_ZA.UTF-8 LC_IDENTIFICATION=C

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

other attached packages:
[1] RcppOctave_0.7.5 Rcpp_0.9.10      stringr_0.6      digest_0.5.2
[5] knitr_0.5

loaded via a namespace (and not attached):
[1] codetools_0.2-8 evaluate_0.4.2  formatR_0.4     highlight_0.3.1
[5] parser_0.0-14   plyr_1.7.1      tools_2.15.0
```

# References


[1] John W Eaton. *GNU Octave Manual*. Network Theory Limited, 2002. ISBN: 0-9541617-2-6. URL: http://www.octave.org/.

[2] Dirk Eddelbuettel and Romain François. "Rcpp: Seamless R and C++ Integration". In: *Journal of Statistical Software* 40.8 (2011), pp. 1–18. URL: http://www.jstatsoft.org/v40/i08/.

[3] Dirk Eddelbuettel and Romain Francois. *RInside: C++ classes to embed R in C++ applications*. R package version 0.2.6. 2012. URL: http://CRAN.R-project.org/package=RInside.

[4] Free Software Foundation. *GNU General Public License*. 2011. URL: http://www.gnu.org/licenses/gpl.html.

[5] Renaud Gaujoux. *RcppOctave: Seamless Interface to Octave – and Matlab*. R package version 0.7.5. 2011. URL: http://r-forge.r-project.org/projects/rcppoctave/.

[6] Yihui Xie. *knitr: A general-purpose package for dynamic report generation in R*. R package version 0.5. 2012. URL: http://CRAN.R-project.org/package=knitr.


[11] http://cran.r-project.org/package=knitr