Connecting R & Octave

Duncan Temple Lang
Department of Statistics
Bell Labs

August 29, 2002

Abstract

This outlines a framework for embedding R in Octave and vice-versa. At present, only the basics are implemented. However, this provides much of the desired functionality.

1 Motivation

Octave (and Matlab) is a powerful language for performing a particular class of numerical computations. The focus of Octave is quite different from the S environment, but they do share some common features used in general scientific computing. For example, visualization, simulation, linear algebra, etc. are available in both systems. Octave relies on external programs to provide visualization techniques. It can be readily directed to use R's powerful graphics engine. At the same time, R can make use of the functionality in Octave that R does not supply. For example, differential equations, rapid matrix operations, etc. are characteristics of Matlab and Octave that aficianados of these systems frequently provide as reasons why they cannot migrate to S.

By embedding R in Octave and Octave in R, we can marry the best of the two system together and allow each camp of users stay within their familiar programming environments while accessing the functionality of both. There is a one time cost to make Octave embeddable and the small amount of generic code to implement the interface between the two systems. After this, except for the installation and configuration issues (to specify the `LD_LIBRARY_PATH` and `OCTAVE_EXEC_PATH`(what is the correct variable!), the result is essentially cost-free to the user. Octave users can access facilities for graphics, probability distributions, statistical modelling, inference, etc. from R. And R users can deploy arbitrary Octave code without much effort.

2 Accessing external Objects

2.1 Octave objects from R

2.2 Calling Functions

3

4 Conversion of objects/values between the systems

4.1 From R to Octave

The algorithm for converting an R object to an Octave value is relatively heuristic at present. If the R object has a non-trivial dimension attribute (i.e. *dim()* returns a value), we assume it is a matrix and create an Octave `Matrix`.

If the dimension attribute is `NULL`, then we check whether it is a list with names. If so, we create an Octave structure. Otherwise, if there are no names, we create a list of octave values (`octave_value_list`). (This is not quite functional yet.)

After testing for these special cases of R objects, we move on to the primitive types. The specifics of this are explained in the following subsection. If the R object is not a primitive, then we typically punt and create a reference to this complex R object. We have create a special class of Octave object – an `ROctaveObjectReference` – which stores a reference to the R object and acts as a proxy to that object. Almost always, we will only try to use the value of this object in Octave in a subsequent call to R.

4.1.1 Primitives and Basic Types

Converting from R to Octave is relatively obvious. The primitive type integer, logical and numeric are mapped to row vectors (`RowVector`). This should probably be a scalar if the R vector has length 1. A character vecotor of length 1 is mapped to a string object, and a longer character vector is mapped to a string vector (`string_vector`).

4.2 User-level converters

5 Memory Management

When to free objects

6 Examples

We can start the R session and find out about the packages available on is search path and the objects/variables available in one of these packages (e.g. position 2) using the following commands.

```
ROctave_callR("search")
ROctave_callR("objects", 2)
```

We can invoke any of the standard R functions by merely giving the name and the arguments to the function as Octave values. Here we generate 10 random values from a standard Normal distribution and then we pass them back from Octave to R for plotting them.

```
x = ROctave_callR("rnorm", 10)
ROctave_callR("plot", x)
```

Note that the labels are far from ideal. This is because R sees the argument passed to the *plot()* call as a literal vector and not a variable in its own lookup space. To be able to handle this more elegantly we either need to introduce a named argument call within Octave or drive the computations from R and have Octave as the embedded interpreter/engine.

The following shows a clumsy way to use named arguments. We use the Octave function `ROctave_namedCallR` (as distinct from `ROctave_callR`). This takes the name of the R function as its first argument, in the same way that `ROctave_callR` does. After this, the arguments to the R function are given as pairs of Octave arguments. The name is given first and then the value. If the name can be omitted in the R function call, one specifies it as the empty string (`""`) in Octave.

```
f = ROctave_namedCallR("system.file", "", "examples", "", "Robj.S", "package", "ROctave")
ROctave_callR("source", f)

x = ROctave_callR("rnorm", 10)
ROctave_namedCallR("plot", "", x, "ylab", "Random Normals")
```

Note that neither resizing or redrawing work in the graphics window. This is because the X events are not passed to R.

The examples above show how we can call R from Octave. These R functions can involve arbitrary computations including calling Octave functions. Essentially, these are callbacks from R to Octave. To do this, the ROctave package must be loaded into R. Then, we can use the *.Octave()* function in R to call arbitrary Octave functions. The script `examples/Octave.S` provides an example of this. We can invoke this from Octave as

```
f = ROctave_namedCallR("system.file", "", "examples", "", "Octave.S", "package", "ROctave")
ROctave_callR("source", f)
```

This finds the example script and the evaluates its contents in R. Part of the computations are to call the Octave function `xx` which must be available to the Octave session for this to work.