Seamless R and Stata integration

E. F. Haghish
Center for Medical Biometry and Medical Informatics (IMBI)
University of Freiburg, Germany
and
Department of Mathematics and Computer Science
University of Southern Denmark
haghish@imbi.uni-freiburg.de

Abstract.

The \texttt{Rcall} package facilitates integrating R with Stata by allowing automatic interprocess communication between the two software. Running R interactively within Stata, \texttt{Rcall} maps a variety of R objects with different classes (integer, numeric, character, matrix, list, data.frame, logical, and null) and returns them to Stata simultaneously. Similarly, it communicates scalars, matrices, variables, and data sets from Stata to R, and also provides an optional real-time synchronization of objects between R and Stata. The article introduces \texttt{Rcall} package and provides examples how R language can be used interactively within Stata or embedded in Stata programs. The method is discussed for integrating other open-source computational languages such as Python and Octave in statistical languages to support object synchronization.

Keywords: language interfacing, interprocess communication, statistical programming, reproducible research

1 Introduction

R [R Core Team 2016] is an open-source statistical language and environment for computing and graphics with a fast-growing user and developer community. The R community consumes a vast amount of human resource for extending R functionalities and making it comprehensive. For example, the CRAN archive for R packages hosts nearly 20,000 packages and this number is growing rapidly everyday. In the current article, I introduce the \texttt{Rcall} package, which embeds the R language and environment in Stata.

The idea of embedding R language in other programming languages is not new. For example, there are several implementations available for calling R from Java [Lang 2005; Satman 2014], Python [Xia et al. 2010], PHP [Mineo et al. 2006], and other software and websites [Neuwirth and Baier 2001; Klinke and Zlatkin-Troitschanskaja 2007; Horner 2005; Neuwirth 2008]. In addition, major commercial statistical software such as SAS, SPSS, and Mathematica include a built-in interface for calling R [SAS 2016; Mathematica 2016; Dalzell 2016; IBM Corporation 2016].

All of these attempts for integrating R in other languages follow the mainstream approach to language interfacing in computer science, which does not
require interactive algorithm development. For example, SPSS and SAS introduce several functions for communicating data with R which makes working with R more difficult, unattractive, somehow unnatural for users who are familiar with R, and particularly pointless for interactive use. Such a workflow makes calling a foreign language unattractive because users are required to transfer their variables and data sets to the guest environment and request a variable or the results to have them back in the host environment. However, in the field of statistics, it is very common to code interactively. Therefore, language interfacing for statistical use demands a seamless integration, allowing convenient interactive practice. The more functions users have to apply to communicate data and results, the less interactive-friendly is the interface.

In the current article my main research interest is providing a method for seamless data communication in language interfacing.

In the current article I suggest a different approach to language interfacing, underscoring automated data communication between the host and guest languages to facilitate interactive statistical practice. I introduce the Rcall package which implements this approach for calling R in Stata and provide several examples for programming computational packages by embedding R in Stata ado-programs. In addition, I explore the possibility of real-time object synchronization between the two languages to further improve the interactive workflow. Finally, I discuss how the method introduced in the current article can be extended to embed other languages for statistical computing.

2 Rcall package

2.1 Features

Rcall provides a new approach for data communication in language interfacing, allowing Stata and R to share objects in several different modes. These modes allow the package to call R interactively or non-interactively in Stata. The package also includes a mode for synchronizing objects between R and Stata, where any change to the object in either environment alters the synchronized in the other (see section 3.1.4). Furthermore, Rcall includes a few commands for managing and monitoring R in Stata. For example, when R is called interactively, Rcall documents the commands in a history do-file that can be executed within Stata to reproduce the analysis. Finally, programming with a foreign language can lead to variety of bugs. The package includes a command to ease embedding R in Stata. All of these features make Rcall a comprehensive package for using R alongside Stata for data analysis or programming.

2.2 Installation

The Rcall package can be installed from GitHub:

```
    . net install Rcall, ///
    from("https://raw.githubusercontent.com/haghish/Rcall/master")
```

In addition, Rcall requires R software, which is available from [https://cran.r-project.org/](https://cran.r-project.org/) for Windows, Mac, and Linux free of charge. After
installation, \texttt{Rcall} attempts to find the path to executable R automatically. However, you can use a particular version by specifying the executable R path. The usual paths to executable R is as follows:

- **Windows:** `C:\Program Files\R\R-version\bin\R.exe`
- **Mac OS X:** `/usr/bin/R`
- **Linux:** `/usr/bin/R`

Alternatively, the R path can be permanently defined as shown below, which also allows the users to specify which version of R they wish to call from Stata, if they have several versions of R installed on the machine:

```
.R call setpath "/path/to/R"
```

Finally, \texttt{Rcall} includes two functions for passing data sets from Stata to R and loading a data set from R to Stata (see section 3). These functions rely on the \texttt{readstata13} (Garbuszus et al., 2016) R package which is used for reading and writing Stata data sets in R. The packages can be installed within Stata as follows:

```
.Rcall:
install.packages("readstata13", repos="http://cran.uk.r-project.org")
```

To test the package installation, call R to display a text from Stata:

```
.Rcall: print("Hello World")
```

3 Syntax
The syntax of the \texttt{Rcall} package can be generalized into two categories, which are \textit{modes} and \textit{subcommands}. The \texttt{Rcall} modes change the general behavior of the package, whereas the subcommands are used for managing and monitoring the package.

3.1 \texttt{Rcall} modes
\texttt{Rcall} can embed R in several modes which are described in table 3.1. The \texttt{Rcall} syntax for applying these modes can be summarized as follows:

```
.Rcall [ mode ] [:] [ R-code ]
```

If the \textit{mode} argument is not specified, R is called interactively and if the \textit{R-code} is not specified, the console mode starts which simulates the R console in Stata. The other modes are \texttt{vanilla} (non-interactive) and \texttt{sync} mode which is an extended interactive mode with object synchronization. The interactive and \texttt{sync} modes can also be used within the console mode.

\footnote{\texttt{Rcall} also looks for `C:\Program Files (x86)\R\` directory in Microsoft Windows}
Table 1: Summary of \texttt{Rcall} syntax

<table>
<thead>
<tr>
<th>Modes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{Rcall}</td>
<td>Enters interactive R console mode</td>
</tr>
<tr>
<td>\texttt{Rcall sync}</td>
<td>Synchronizes objects during R console mode</td>
</tr>
<tr>
<td>\texttt{Rcall [:]}</td>
<td>Calls R interactively from Stata</td>
</tr>
<tr>
<td>\texttt{Rcall sync [:]}</td>
<td>Calls R interactively and synchronizes objects</td>
</tr>
<tr>
<td>\texttt{Rcall vanilla [:]}</td>
<td>Calls R non-interactively from Stata</td>
</tr>
</tbody>
</table>

3.1.1 R console mode

When executed without any subcommand or \textit{R-code}, the \texttt{Rcall} command simulates the R console within Stata results window. This mode allows writing multiline R commands such as functions, executing the commands, and obtaining the results interactively. Similar to Mata interactive mode, the interactive console mode can be closed using the \texttt{end} command. Here is an example of defining an R function interactively in Stata and executing it. The \texttt{print.numeric} function will return the given input if its class is numeric or integer and return an error otherwise:

```
. R
   (type end to exit)
   . print.numeric <- function(x) {
     +     if (is.numeric(x) | is.integer(x)) {
     +       return(x)
     +     }
     +     else {
     +       stop("input of class ", class(x), " is not acceptable")
     +     }
     +   }
   . print.numeric(1:10)
   [1] 1 2 3 4 5 6 7 8 9 10
   . end
```

Technical note

In the R console mode, an R code paragraph can be copy-pasted in the Stata command window, which also preserves the code indentation. When curly brackets are used in the code, the console mode allows multi-line code. The “+” operator
is a sign of an in-progress multi-line code, which is also automatically indented to provide a visual guide for the current number of unclosed brackets, as shown in the example above.

The console mode cannot be used for executing R commands from Stata do-file editor and consequently, cannot be used for embedding R code in Stata programs. Nevertheless, this mode should be convenient for exploratory analysis or casual use. Moreover, Rcall registers R commands that are executed in interactive modes - including the console mode - and creates a do-file that can reproduce the analyses carried out by R (see section 3.2).

### 3.1.2 Interactive mode

If the mode argument is not specified, Rcall executes R commands interactively without entering the console mode. The interactive mode retains all of the defined objects, attached objects, and loaded packages. For instance, the previous example can be continued by calling the function that was previously defined in the console mode, since the console mode is also interactive and memorizes the function `print.numeric`:

```
. R: print.numeric(10)
[1] 10
```

The interactive mode has an advantage over the console mode, namely, R commands can be executed from Stata do-file editor or programs. Therefore, when there is a concern regarding the reproducibility of the computation, the interactive mode is preferable to the console mode.

### 3.1.3 vanilla mode

The vanilla mode calls R non-interactively by evoking a new R session, in parallel to the previous interactive R session. The example below demonstrates this point by calling the `print.numeric` function – that was previously defined in an interactive session – in vanilla mode. Since the vanilla begins with opening a new R session, an error is expected because the `print.numeric` function is not yet defined:

```
. R vanilla: print.numeric(10)
Error: could not find function "print.numeric"
```

After executing R commands in vanilla mode, Rcall quits R without saving the session for future use. This mode is particularly useful for embedding R in a Stata ado-program or package because it ensures that programs will not interfere with the interactive R session of the user.

### 3.1.4 sync mode

The sync mode is an extension of the interactive mode. However, this mode provides an unusual feature by automatically synchronizing objects between R and Stata in real-time. The sync mode allows R and Stata to mirror one another. Matrices and scalars will be automatically migrated from Stata to R
upon synchronization and after executing the R command, the newly defined scalars and matrices in R environment – if there is any – will be synchronized with Stata. This mode can also be used within the console mode by evoking the console using `Rcall sync`:

**Example**

```
. scalar str = "myname"
. R sync: print(str)
[1] "myname"

. R sync: str = NULL
. end

. display str
NULL
```

The `sync` mode considerably facilitates requesting and returning objects from the host language to the guest language and the other way around. Moreover, users do not have to constantly apply this mode for executing R commands. Instead, they can simply auto-synchronize objects between R and Stata at the beginning of the session to transport scalars and matrices from Stata to R or get the same class of objects from R to Stata. The example below demonstrates how convenient is to communicate matrices between Stata and R in the `sync` mode:

**Example**

```
. matrix A = (1,2;3,4)
. matrix list A
. R sync: B = A+A
. matrix list B
```

Synchronizing matrices and scalars for each executed R command is convenient, but not an efficient way for data communication. Especially, if the user does not wish to synchronize all of the objects between R and Stata. `Rcall` provides an alternative approach for communicating data from Stata to R which is discussed in section 4.
3.2 Rcall subcommands

A few subcommands were defined for Rcall package to conveniently manage R within Stata. These subcommands are summarized in table 3.2 and can be executed using the following syntax:

\texttt{Rcall \ [subcommand]} \\

<table>
<thead>
<tr>
<th>Subcommands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{Rcall setpath &quot;path/to/R&quot;}</td>
<td>Permanently defines the R path</td>
</tr>
<tr>
<td>\texttt{Rcall clear}</td>
<td>Clears the memory of interactive R session</td>
</tr>
<tr>
<td>\texttt{Rcall describe}</td>
<td>Describes the status of Rcall</td>
</tr>
<tr>
<td>\texttt{Rcall history}</td>
<td>Opens the \texttt{Rhistory.do} in do-file editor</td>
</tr>
<tr>
<td>\texttt{Rcall site}</td>
<td>Opens the \texttt{Rprofile.site} in do-file editor</td>
</tr>
</tbody>
</table>

The \texttt{setpath} subcommand was already introduced in section 2.2 for permanently setting up the path to R in Rcall. The \texttt{clear} subcommand removes the interactive R workspace including all defined and attached objects, packages, and the R history file. The \texttt{describe} subcommand provides useful information regarding Rcall which are the path to the executable R used by Rcall, the version of the R, and the paths to the \texttt{Rhistory.do} and \texttt{Rprofile.site} files. The \texttt{Rhistory.do} is created and updated by Rcall to keep track of the interactive R session and the \texttt{Rprofile.site} allows the users to customize the R, when it is called from Stata. For example, the \texttt{Rprofile.site} file can be used to change the setting of R (such as language, etc) when it is called from Stata. The \texttt{site} and \texttt{history} subcommands open the \texttt{Rprofile.site} and \texttt{Rhistory.do} files in do-file editor respectively.

4 Data communication between Stata and R

Communicating Stata local and global macros from Stata to R is effortless because values of macros are interpreted by Stata when they are given to any command. For example:

\begin{verbatim}
. global num 10
. global str "my string"
. R: print($num); print("$str");
[1] 10
[1] "my string"
\end{verbatim}

In addition to the \texttt{sync} mode, Rcall provides a few functions for transferring scalars, matrices, and datasets, which are summarized in table 4. These functions can only be interpreted by Rcall command and are available for all \texttt{modes} of the Rcall package.
### Table 3: Summary of functions for transferring data between Stata and R

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>st.scalar()</code></td>
<td>transfers numeric and string scalar to R</td>
</tr>
<tr>
<td><code>st.matrix()</code></td>
<td>transfers numeric matrix to R</td>
</tr>
<tr>
<td><code>st.var()</code></td>
<td>transfers numeric or string variable to R</td>
</tr>
<tr>
<td><code>st.data()</code></td>
<td>transfers Stata dataset to R</td>
</tr>
<tr>
<td><code>st.load()</code></td>
<td>transfers data frame from R to Stata</td>
</tr>
</tbody>
</table>

While passing data from Stata to R requires using these functions (unless the `sync` mode is used), R automatically returns objects of classes such as `numeric`, `integer`, `character`, `logical`, `matrix`, `list`, and `NULL` to Stata as `rclass` scalars, locals, and matrices.

In the example below, a Stata scalar named `num` is defined to equal 10. Then the scalar is given to R, where an object with an identical name is created in R intentionally. After the manipulation, R automatically returns the new object to Stata as an `rclass` object, which coexists temporarily in Stata memory and does not influence the value of Stata scalar, unless the `Rcall` is executed in `sync` mode.

```
. scalar num = 10
. R: (num = st.scalar(num)*st.scalar(num))
   1   100
. return list
   scalars:
      r(rc) = 0
      r(num) = 100
. display num
   10
```
The `r(rc) = 0` indicates that R was executed without an error. The example can be continued by defining a string scalar and a matrix, named `str` and `C` respectively:

```
. scalar str = "my string"
. R: (str = paste(st.scalar(str), "has changed"))
   1   "my string has changed"
. matrix A = (1,2|3,4)
. matrix B = (96,96|96,96)
. R: (C <- st.matrix(A) + st.matrix(B))
   [,1] [,2]
   [1,]  97  98
   [2,]  99 100
```
The `st.var()`, `st.data()` functions can be used to transfer data from Stata to R and the `st.load()` function forcefully loads a data frame from R to Stata. The `st.data()` function can transfer the currently loaded dataset to R, or if
the path to a Stata dataset is given to the function, it transfers the specified
dataset to R without loading it in Stata. Using these functions is demonstrated
in the example below.

. quietly sysuse auto, clear
. R: (price_mean = mean(st.var(price)))
[1] 6165.257
. R: data = st.data()
. clear
. R: data = data[,1:5]
. R: st.load(data)
. list in 1
+----------------------------------------------+
<table>
<thead>
<tr>
<th>make  price  mpg  rep78  headroom</th>
</tr>
</thead>
</table>
1. | AMC Concord  4099  22    3    2.5 |

Since Rcall was executed interactively, all of the defined objects apart from the
datasets are returned from R to Stata automatically:

. return list
scalars:
  r(rc) = 0
  r(price_mean) = 6165.257
  r(num) = 100

macros:
  r(str) : "my string has changed"

matrices:
  r(C) : 2 x 2

4.1 Controlling returned objects from R to Stata

R supports a wider range of modes and classes compared to Stata. As a result, some of the R objects that have classes unsupported by Stata will be coerced
to scalar or macros. For example, R objects of class NULL and logical are
returned as string macro to Stata. Moreover, numeric and character vectors
are returned as macros to Stata. Finally, objects of class list are broken into
their components and are returned accordingly, since Stata does not support a recursive object such as a list. This default behavior can be altered if the
user converts the type of the object in R. For example, a numeric vector can
be returned as a single column matrix, if the type conversion takes place within
the R code, as shown in the example below:

Example

. Rcall clear
   (R memory cleared)
. Rcall: a = 1:10
. return list
scalars:
    \[ r(rc) = 0 \]

macros:
    \[ r(a) : "1 2 3 4 5 6 7 8 9 10" \]

. Rcall: a = matrix(a)
. return list

scalars:
    \[ r(rc) = 0 \]

matrices:
    \[ r(a) : 10 \times 1 \]

By default, all of the supported R classes (numeric, character, and logical vectors, numeric matrices, lists, and NULL) are returned from R to Stata. There are occasions when the user is only interested to return only a particular object or a few objects from R to Stata. For example, a programmer who is using \texttt{Rcall} to create a package for Stata might create several objects in the program but only wants to return a list. There are two ways to limit the returned objects. First, the R objects can be removed from R workspace as shown in the example below:

. Rcall vanilla : a = 1; b = 2; x = a + b; rm(a,b);
. return list

scalars:
    \[ r(rc) = 0 \]
    \[ r(x) = 3 \]

Another possibility is creating a character vector named \texttt{st.return} in R and specifying the name of the objects that \texttt{Rcall} should return. The previous example can be repeated as follows which returns only the value of \( r(x) \).

\textbf{Example}

. Rcall vanilla : a = 1; b = 2; c = a + b; st.return = "x";
. return list

scalars:
    \[ r(x) = 3 \]

To include the \( r(rc) \) in the returned objects, the \texttt{c} function can be applied to create a character vector, listing all of the objects that should be returned e.g. \texttt{st.return = c("c", "rc")}. Note that in the interactive modes (including console and \texttt{sync} modes) the \texttt{st.return} object will remain in the R memory and continue to limit the data communication from R to Stata unless you remove or redefine it. However, using \texttt{st.return} is convenient in the \texttt{vanilla} mode, which I discuss in the following section.
5 Embedding R in Stata programs

Using Rcall package, Stata users can write ado programs to embed R functions and packages in Stata programs. To demonstrate the potential of the package for making use of R language in Stata several example programs are discussed in this section. All of these examples evoke R in vanilla mode to ensure the users’ previous interactive sessions do not influence the program. The examples mentioned below are available for download on GitHub [https://github.com/haghish/Rcall/tree/master/examples](https://github.com/haghish/Rcall/tree/master/examples).

5.1 Examples

5.1.1 echo program

To demonstrate embedding R in Stata programs, I begin with a simple example program that echoes the given string. The string is obtained from Stata and given to R as a local macro, which I mentioned to be the simplest method of data communication from Stata to R. The macro `0` returns whatever is given to the echo program and next is passed to Rcall. Therefore, all of the functions that are used for communicating data from Stata to R can also work with the echo program:

Example

```stata
program echo
    Rcall vanilla: cat(`0´)
end

. echo "Hello World"
Hello World
. scalar a = "hello world"
. echo st.scalar(a)
hello world
```

5.1.2 summary program

The following example uses the summary function in R to summarize the list of variables. The program can be used with by to repeat the command on a subset of the data as well as if and in arguments which are used for executing the command on a subset of the data. To send the subset of the data specified by Stata syntax, the program temporarily keeps the marked sampled and the specified variables. Next, it applies the st.data function to transport the loaded data to R, where the summary function is called for each variable in the dataset using the sapply R function.
program summary, byable(recall)
version 12
syntax varlist [if] [in]
marksample touse
preserve
quietly keep if `touse´
quietly keep `varlist´
Rcall vanilla: sapply(st.data(), summary)
restore

end

Example

. by foreign: summary price mpg

----------------------------------------
| -> foreign = Domestic
| price mpg
| Min.  3299  26.0
| 1st Qu. 4389  26.0
| Median 4453  28.0
| Mean   4655  27.4
| 3rd Qu. 4647  28.0
| Max.   6486  29.0
----------------------------------------

| -> foreign = Foreign
| price mpg
| Min.  3895  26.0
| 1st Qu. 4046  26.5
| Median 4197  27.0
| Mean   4197  27.0
| 3rd Qu. 4348  27.5
| Max.   4499  28.0

5.1.3 Using ggplot2 R package

There are numerous R packages that can produce a variety of interesting graphics such as ggplot2 (Wickham 2009). Rcall can be used to make use of the qplot function from the ggplot2 package that can produce a wide range of plots. The example below demonstrates the qplot program that simply loads the ggplot2 library in R vanilla mode, uses the st.data() function to transports the currently loaded data to R, and passes all of the arguments to qplot function in in R. By default, the program creates a pdf graphical file named Rplot.pdf.

program qplot
version 12
Rcall vanilla : library(ggplot2); qplot(data=st.data(), `0´)
end

Example

. sysuse auto, clear
. qplot mpg, price, colour = foreign, shape = foreign

12
The `ggplot2` package has an argument for the dataset and it also recognizes the variable names which makes embedding the package in Stata easy. However, the example above is oversimplified because it just passes the `ggplot2` syntax to R via `Rcall`. As a result, the syntax of the program does not look like a usual Stata program with `varlist` and options. Moreover, the program only creates pdf plots, whereas options can be added to store the plot in other supported formats which are `png`, `tiff`, `bmp`, `jpeg`, in addition to `pdf`. The example can be reprogrammed to use Stata syntax as shown in the example below. To keep the program short, only a few options of the `qplot` R function are included. This example will produce an identical plot to the previous example.

```
program qplot
    version 12
    syntax varlist [, colour(name) shape(name) format(name)]
```
// selecting x and y
tokenize `varlist´
if !missing("`2´") {
    local x "`2´"
    local y "`1´"
}
else {
    local x "`1´"
    local y NULL
}

// default options
if !missing("`colour´") local `colour´ = "`colour´"
if !missing("`shape´") local `shape´ = "`shape´"
if missing("`format´") local `format´ = pdf

Rcall vanilla : "`format´("Rplot."`format´")
library(ggplot2);
qplot(data=st.data(),x="`2´",y="`1´", `colour´, `shape´)
end

Example

. qplot price mpg , colour(foreign) shape(foreign) format(png)
(output omitted)

5.2 Defensive programming strategies for Rcall

The previous programs are not ideal examples of embedding R in ado programs because they are not disciplined enough and do not control for potential problems. For example, the previous example must ensure that a maximum of two variables can be specified. Furthermore, it had to check that the `format´ option is one of the supported formats and the other options are valid variables. Otherwise, R will handle the errors instead of Stata and that might lead to confusion caused by unclear error messages. For instance, let’s assume a user wishes to export a `giff´ file, which is not a supported format by R:

Example

. qplot price mpg , colour(foreign) shape(foreign) format(giff)
Error: could not find function "giff"

---Break---
r(1);

Such an error might be confusing for users because they have no clue about the function `giff´. Therefore, before calling R, the syntax processing should be done as disciplined as possible. As shown in this example, when an error occurs in R, Rcall returns the error in Stata and breaks the Stata program. Therefore, Rcall ensures that R is safely executing the code before continuing with the rest of the Stata program.
Moreover, the `qplot` program example is based on several assumptions that the program does not validate. Let’s assume I have released the `qplot.ado` program, allowing Stata use `ggplot2` R package in Stata. The program assumes that:

1. the user has installed `Rcall`
2. the user’s `Rcall` package has a minimum acceptable version or higher
3. R is installed on the users machine and can be accessed by `Rcall`
4. R has an acceptable version to be used with `ggplot2`
5. `ggplot2` R package is installed
6. `ggplot2` has a certain version which supports the specified arguments

Therefore any Stata package that uses `Rcall` to ember R in an ado program should check for similar assumptions. For the first assumption, it might suffice to search for a program file within Stata, for example, `Rcall.ado`:

```stata
Example

. capture findfile Rcall.ado
if _rc == 0 {
    display as error "Rcall package is required"
    error 198
}
```

For the remaining assumptions, `Rcall` package already has a solution. The package includes a program named `Rcall_check.ado`, which can be used to check that R is accessible via `Rcall`, check for the required R packages, and specify a minimum acceptable version for R, `Rcall`, and all the required R packages. The syntax of the command is as follows:

```
Rcall_check [ pkgname>=version ] [...] [, rversion(str) rcallversion(str) ]
```

As showed in the syntax, all of the arguments are optional. When `Rcall_check.ado` is executed without any argument or option, it simply checks whether R is accessible via `Rcall` and returns `r(rc)` and `r(version)` which is the version of the R that is used by the package. If R is not accessible, an error is returned accordingly.

In the example of the `qplot` function, let’s assume that we wish to check that the user has `Rcall` version 1.3.3 or higher, `ggplot2` version 2.1.0 or higher, which itself requires a minimum R version of 3.1.0:

```
Rcall_check ggplot2>=2.1.0 , r(3.1.0) rcall(1.3.3)
```
Applying the `Rcall_check` program in an ado program ensures that the user of a distributed Stata package will get clear and informative error messages if any of the required packages is missing or if he has an older version of them installed on his machine. These are the bare minimum requirements of a defensive ado program that embeds a foreign language and calls packages that might not be installed on the users’ machine.  

5.3 Returning stored results

`Rcall` makes returning stored results from R to Stata very convenient because objects of several classes are automatically transferred from R to Stata (see section 4). As a result, using `return add` command in the `ado` program is enough to return the values received from `Rcall` in the ado program as shown in the example below.

The `lm` ado program calls the `lm()` function in R to fit a linear model. Since `lm()` function requires the predictors to be added using a “+” sign, the `subinstr` function was used to replace the white-spaces between predictors with plus sign. After running the linear model, several objects and matrices are created to store the results of the `lm()` function, which are returned to Stata using the `return add` command.

```stata
program lm, rclass
    syntax varlist [if] [in]
    Rcall_check /// check R
    tokenize `varlist' 
    local first `1'
    macro shift
    local rest `*'
    local rest : subinstr local rest " " "+", all

    marksample touse
    preserve
    quietly keep if `touse' /// subset the data
    quietly keep `varlist' /// subset the data

    Rcall vanilla:
    ///
    attach(st.data()); /// attach data
    out = summary(lm(`first' ~ `rest'));
    /// fit the model
    print(out);
    /// display output
    coefficients = as.matrix(out$coefficients);
    /// return coef
    res_se = out$sigma;
    /// residual SE
    res_df = out$df[2];
    /// residual DF
    r_squared = out$r.squared;
    /// R-squared
    adj_r_squared = out$adj.r.squared;
    /// Adj R-squared
    f_statistic = as.matrix(out\verb|$|fstatistic);
    /// F-statistics
    f_statistic_p = 1 - pf(out[10][1], /// F-p value
        out[[10]][1], out[[10]][1]);
    ///
    rm(out);
```

Rcall checks for installation of `readstata13` R package and its required version when the `st.data` function is called.
To test the program and the returned results, I use the `auto.dta` data set and for a simple linear model, having the `price` variable as dependent and `mpg` and `turn` as predictors:

### Example

```stata
.sysuse auto, clear
.lm price mpg turn if price < 16000
```

**Call:**

```
lm(formula = price ~ mpg + turn)
```

**Residuals:**

```
Min 1Q Median 3Q Max
-3083.7 -1854.7 -970.6 1412.9 9867.1
```

**Coefficients:**

```
Estimate Std. Error t value Pr(>|t|)
(Intercept) 13204.27 5316.19 2.484 0.01536 *
mpg -259.70 76.85 -3.379 0.00118 **
turn -38.04 101.06 -0.376 0.70775
```

---

**Signif. codes:** 0 *** 0.001 ** 0.01 * 0.05 . 0.1 1

Residual standard error: 2639 on 71 degrees of freedom
Multiple R-squared: 0.2211, Adjusted R-squared: 0.1992
F-statistic: 10.08 on 2 and 71 DF, p-value: 0.0001402

```stata
.return list
```

**Scalars:**

```
    r(res_df) = 71
    r(res_se) = 2639.433
    r(rc) = 0
    r(r_squared) = 0.2211369
    r(f_statistic_p) = 0.0005269716
    r(adj_r_squared) = 0.1991971
```

**Matrices:**

```
    r(f_statistic) : 3 x 1
    r(coefficients) : 3 x 4
```

### Technical remarks

1. The dollar sign "$" is used by Stata to specify global macros. Unless you are trying to pass the value of a global macro to R, the dollar sign should begin with a backslash whenever it is aimed to be used in R. Examples were provided in section 5.3.
2. The R commands can be separated by semicolon “;” and executed in a single call which saves a lot of execution time that is wasted in the process of evoking R and returning objects to Stata for each call. Yet, you can maintain the readability of your code by writing each R command on a single line.

3. Using \texttt{#delimit ;} in Stata programs is generally not favored in Stata programs (Cox 2005). Especially because it seems if it is used for writing multiple-line code and comments in the ado file because it reduces the readability of the program. Nevertheless, when R code is written in an ado program it should be executed at once i.e. R commands should be separated using the semicolon sign. Therefore, using \texttt{#delimit ;} in Stata must be avoided.

4. Use the interactive mode whenever you need to call R in several successive commands to carry out a particular analysis. Avoid using the interactive mode in ado programs.

5. Use \texttt{sync} mode carefully since changes made in R are pushed to Stata. This is also the slowest \texttt{Rcall} mode.

6. Use \texttt{vanilla} mode for executing R in ado programs. This mode ensures that the R objects defined in the interactive mode do not influence your program.

7. use the \texttt{Rcall\_check} in ado programs to check for required version of R, \texttt{Rcall}, and R packages.

8. There is no necessity to write all of the R code in an \texttt{ado} file. The functions can be written in one or more R script files and distributed with the Stata package. Next, \texttt{Rcall} can begin with sourcing the script file using the \texttt{source()} function, followed by code for getting the input from Stata and running the analysis. This not only makes the package easy to read, but also easy to debug and maintain. However, the \texttt{Rcall} functions that are used for data communication cannot be included in R script files because they can only interpreted by \texttt{Rcall}.

9. Using the colon sign “:” is optional. Applying it does not influence \texttt{Rcall} and it is merely a visual guide, separating Stata code from R code. However, if you are trying to access an R object with the same name as an \texttt{Rcall} subcommands (\texttt{setpath}, \texttt{clear}, \texttt{history}, \texttt{describe}, and \texttt{site}), the colon sign should be placed before the object name as shown in the example below:

\textbf{Example}

```r
. R clear
(R memory cleared)
```

18
7 Discussion

Embedding one computer language into another one is usually done to avoid reinventing the wheel by using existing functions and libraries of the other languages as well as taking advantage of the speed or characteristics of the embedded language. When it comes to statistics, R has the largest developer community and many of the new methods first appear in R. In the current article, I discussed a method for embedding R in Stata that provides features not available in similar interfacing applications that are created for other statistical software such as SPSS, SAS, and Mathematica. In addition, I argued that in the field of statistics the analysis scripts is often developed interactively and therefore, a point that is usually neglected in interfacing languages in statistical software. Besides, I also underscored that if interactive workflow is interrupted by requiring several commands for communicating data back and forth between the host and the guest languages, it will no longer be a pleasing workflow. However, embedding computational languages in statistics software have followed the usual procedure of computer science without considering the practical demands of applied statisticians.

To address these problems, Rcall package was developed to examine the idea of automatic reciprocal data communication between the host and the embedded language. Rcall package included several modes that behave differently to satisfy the features required for using R interactively as well as embedding it in Stata. The main feature of the Rcall package is that it allows the users to specify to what extent they wish to automatize the process of data communication, which can vary from the sync mode – synchronizing objects between Stata and R – to a restricted and non-interactive mode that can be used for embedding R in Stata packages.

Rcall provides a fully reproducible and automated procedure for carrying out a part of the data analysis in R within Stata. A fully reproducible procedure implies that all of the results produced by R should be dynamically accessible by Stata and can be used to produce an automated dynamic analysis document. Such an action can be carried out using any of the common literate programming packages such as weaver or markdoc, which also can highlight the syntax of R as well as Stata in the document (Haghish 2014a,b 2015).

Nevertheless, presumably the biggest contribution of Rcall would be facilitating programming with R in Stata. Rcall along with Rcall_check program, notably facilitates embedding R in Stata programs which allows calling thousands of R functions in Stata programs.

Finally, the engine that was written for embedding R and process of re-
ciprocal data communication can also be used to embed other computational languages in Stata (or other statistical software) such as Python and Octave. The call_return.ado program analyzes a file returned from R to create the same objects in Stata. This program can be used for reading results from files written by any language, as long as the file is written with a specific format defined by Rcall\(^3\).

8 References


\(^3\)https://github.com/haghish/Rcall/wiki/call_return

Neuwirth, E. 2008. R meets the workplace-embedding R in Excel to make it more accessible.


