R-Python Interactions

M. Fuen(t)es - SED - Bordeleko INRIA

November 27, 2015

Outline

Outline

- Why use **R** and **Python** ?

Outline

- Why use **R** and **Python** ?
- Calling **Python** from **R**

Outline

- Why use **R** and **Python** ?
- Calling **Python** from **R**
- Calling **R** from **Python**

Outline

- Why use **R** and **Python** ?
- Calling **Python** from **R**
- Calling **R** from **Python**
- Jupyter

Why use **R** ?

Why use **R** ?

- **R** is the *lingua franca* for statistical algorithms

Why use **R** ?

- **R** is the *lingua franca* for statistical algorithms
- Good statistical functions

Why use **R** ?

- **R** is the *lingua franca* for statistical algorithms
- Good statistical functions
- CRAN website http://www.cran.org is a good place to promote your code

Why use **R** ?

- **R** is the *lingua franca* for statistical algorithms
- Good statistical functions
- CRAN website http://www.cran.org is a good place to promote your code
- It also is free software ☺

Why use **R** ?

- **R** is the *lingua franca* for statistical algorithms
- Good statistical functions
- CRAN website http://www.cran.org is a good place to promote your code
- It also is free software ☺

Why use **R** ?

- **R** is the *lingua franca* for statistical algorithms
- Good statistical functions
- CRAN website http://www.cran.org is a good place to promote your code
- It also is free software ☺
- The best thing about **R** is that it was written by statisticians

Why use **R** ?

- **R** is the *lingua franca* for statistical algorithms
- Good statistical functions
- CRAN website http://www.cran.org is a good place to promote your code
- It also is free software ☺
- The best thing about **R** is that it was written by statisticians
- The worst thing about **R** is that it was written by statisticians

Why use **R** ?

- **R** is the *lingua franca* for statistical algorithms
- Good statistical functions
- CRAN website http://www.cran.org is a good place to promote your code
- It also is free software ☺
- The best thing about **R** is that it was written by statisticians
- The worst thing about **R** is that it was written by statisticians
- Floreal Morandat, "De la concision à l'efficacité, le cauchemar d'un informaticien",4èmes rencontres R 2015

Why use **Python** ?

Why use **Python** ?

- **Python** starts to replace Matlab for scripting numerical computations

Why use **Python** ?

- **Python** starts to replace Matlab for scripting numerical computations
- It is free software 

Why use **Python** ?

- **Python** starts to replace Matlab for scripting numerical computations
- It is free software ☺
- some modules like are well suited for numerical computations :

Why use **Python** ?

- **Python** starts to replace Matlab for scripting numerical computations
- It is free software ☺
- some modules like are well suited for numerical computations :
  - numpy : basic linear algebra

Why use **Python** ?

- **Python** starts to replace Matlab for scripting numerical computations
- It is free software 
- some modules like are well suited for numerical computations :
  - numpy : basic linear algebra
  - scipy : integration, ode, special functions,

Why use **Python** ?

- **Python** starts to replace Matlab for scripting numerical computations
- It is free software ☺
- some modules like are well suited for numerical computations :
  - numpy : basic linear algebra
  - scipy : integration, ode, special functions,
  - matplotlib : graphics plots

Why use **Python** ?

- **Python** starts to replace Matlab for scripting numerical computations
- It is free software ☺
- some modules like are well suited for numerical computations :
  - numpy : basic linear algebra
  - scipy : integration, ode, special functions,
  - matplotlib : graphics plots
  - mpi4Py, PyTrilinos, ...

Installing rPython

Installing rPython

- to call **Python** from **R** , install rPython

```
install.packages("rPython")
```

Installing rPython

- to call **Python** from **R** , install rPython

```
install.packages("rPython")
```

- and use it

```
require("rPython")
python.exec("def f(x): return sum(range(x))")
python.call("f", 10)
```

Installing rPython

- to call **Python** from **R** , install rPython

```
install.packages("rPython")
```

- and use it

```
require("rPython")
python.exec("def f(x): return sum(range(x))")
python.call("f", 10)
```

will return 45

Using rPython

Using rPython

- API is simple

Using rPython

- API is simple
- python.assign, python.get $\rightarrow$ set or retrieve values of variables

Using rPython

- API is simple
- python.assign, python.get $\rightarrow$ set or retrieve values of variables
- python.exec $\rightarrow$ exec code with side effect

Using rPython

- API is simple
- python.assign, python.get $\rightarrow$ set or retrieve values of variables
- python.exec $\rightarrow$ exec code with side effect
- python.call, python.method.call $\rightarrow$ calling functions

Using rPython

- API is simple
- python.assign, python.get $\rightarrow$ set or retrieve values of variables
- python.exec $\rightarrow$ exec code with side effect
- python.call, python.method.call $\rightarrow$ calling functions
- python.load $\rightarrow$ load scripts

Simple example with rPython

Simple example with rPython

- using assign, exec and get

```
python.assign("a", 10)
python.exec("b=sum(range(a))")
python.get("b")
```

Simple example with rPython

- using assign, exec and get

```
python.assign("a", 10)
python.exec("b=sum(range(a))")
python.get("b")
```

- The following **Python** script

```
def sum_of_squares(n):
    return sum(map(lambda x:x*x,range(n)))
```

Simple example with rPython

- using assign, exec and get

```
python.assign("a", 10)
python.exec("b=sum(range(a))")
python.get("b")
```

- The following **Python** script

```
def sum_of_squares(n):
    return sum(map(lambda x:x*x,range(n)))
```

- could be call, such as :

```
python.load("un_script.py")
python.call("sum_of_squares",10)
```

Simple example with rPython

- using assign, exec and get

```
python.assign("a", 10)
python.exec("b=sum(range(a))")
python.get("b")
```

- The following **Python** script

```
def sum_of_squares(n):
    return sum(map(lambda x:x*x,range(n)))
```

- could be call, such as :

```
python.load("un_script.py")
python.call("sum_of_squares",10)
```

will return 285

How it works

This package is based on Jython and use JSON serialisation system to convert objects

How it works

This package is based on Jython and use JSON serialisation system to convert objects

- It is not possible to convert functions

How it works

This package is based on Jython and use JSON serialisation system to convert objects

- It is not possible to convert functions

```
f<-function (x) {  x*x;}
python.assign("g",f)
```

How it works

This package is based on Jython and use JSON serialisation system to convert objects

- It is not possible to convert functions

```
f<-function (x) {  x*x;}
python.assign("g",f)
```

will return a error : "converting an R function to JSON as null."

How it works

This package is based on Jython and use JSON serialisation system to convert objects

- It is not possible to convert functions

```
f<-function (x) {  x*x;}
python.assign("g",f)
```

will return a error : "converting an R function to JSON as null."

- It must be pure python : C compile modules cannot be imported.

How it works

This package is based on Jython and use JSON serialisation system to convert objects

- It is not possible to convert functions

```
f<-function (x) {  x*x;}
python.assign("g",f)
```

will return a error : "converting an R function to JSON as null."

- It must be pure python : C compile modules cannot be imported.

You cannot use Numpy nor Scipy ☹

Calling **R** from **Python**

- We will use the package `rpy2`

Calling **R** from **Python**

- We will use the package `rpy2`
- Be sure that your **R** is compiled with the `--enable-R-shlib` : `find /prefix/to/R/ -name "libR.so"` must return something. If not, try to recompile **R**

```
./configure --prefix=/path/to/install --enable-R-shlib
```

Calling **R** from **Python**

- We will use the package `rpy2`
- Be sure that your **R** is compiled with the `--enable-R-shlib` : `find /prefix/to/R/ -name "libR.so"` must return something. If not, try to recompile **R**

```
./configure --prefix=/path/to/install --enable-R-shlib
```

- install it

```
CFLAGS=-I/path/to/your/R/include/ pip install rpy2
```

Calling **R** from **Python**

- We will use the package `rpy2`
- Be sure that your **R** is compiled with the `--enable-R-shlib` : `find /prefix/to/R/ -name "libR.so"` must return something. If not, try to recompile **R**

```
./configure --prefix=/path/to/install --enable-R-shlib
```

- install it

```
CFLAGS=-I/path/to/your/R/include/ pip install rpy2
```

- check everything is ok

```
LD_LIBRARY_PATH=/path/to/R/lib python2.7 -m rpy2.tests
```

Calling **R** from **Python**

- We will use the package `rpy2`
- Be sure that your **R** is compiled with the `--enable-R-shlib` : `find /prefix/to/R/ -name "libR.so"` must return something. If not, try to recompile **R**

```
./configure --prefix=/path/to/install --enable-R-shlib
```

- install it

```
CFLAGS=-I/path/to/your/R/include/ pip install rpy2
```

- check everything is ok

```
LD_LIBRARY_PATH=/path/to/R/lib python2.7 -m rpy2.tests
```

The **R** instance

The **R** instance

- first import rpy2

```
import rpy2
import rpy2.robjects as robjects # create alias
r = robjects.r
```

The **R** instance

- first import rpy2

```
import rpy2
import rpy2.robjects as robjects # create alias
r = robjects.r
```

- trough `robjects.r` we interact with a **R** instance

```
r["pi"] # get the pi-approximation in R
rsum1 = r["sum"] # create an alias to R sum
rsum2 = r.sum # another alias to sum
```

The R instance

- first import rpy2

```
import rpy2
import rpy2.robjects as robjects # create alias
r = robjects.r
```

- trough `robjects.r` we interact with a R instance

```
r["pi"] # get the pi-approximation in R
rsum1 = r["sum"] # create an alias to R sum
rsum2 = r.sum # another alias to sum
```

- to create a variable, you need to use `robjects.globalenv`

```
r["mavar"] = 3 # failed, TypeError
r.mavar = 3 # failed, just add a python attribute
robjects.globalenv["mavar"] = 3 # ok, fine !
```

Data structures

Data structures

- Objects returned by `robjects.r` are **R** data structures :

```
r["pi"] # FloatVector of size 1
r.seq(3) # IntVector of size 3
```

Data structures

- Objects returned by `robjects.r` are **R** data structures :

```
r["pi"] # FloatVector of size 1
r.seq(3) # IntVector of size 3
```

- You can directly create vectors

```
z = robjects.FloatVector([0]*10)
x = robjects.ListVector({'a': 1, 'b': 2, 'c': 3})
```

Data structures

- Objects returned by `robjects.r` are **R** data structures :

```
r["pi"] # FloatVector of size 1
r.seq(3) # IntVector of size 3
```

- You can directly create vectors

```
z = robjects.FloatVector([0]*10)
x = robjects.ListVector({'a': 1, 'b': 2, 'c': 3})
```

- You can access data trough **Python** style or **R** style :

```
z[0] = 1 # python style, indexing from zero
z.rx(1) = 1 # R style, indexing from 1
```

Interact with Numpy

Interact with Numpy

- convert **to** numpy : use `np.array` or `np.asarray` (without copy)

Interact with Numpy

- convert **to** numpy : use `np.array` or `np.asarray` (without copy)
- convert **from** numpy : use `numpy2ri.activate()`

```
from rpy2.robjects import numpy2ri
numpy2ri.activate()
r.sum(np.arange(100)) # automatic cast
```

Packages and Functions

Packages and Functions

- one can add function with string

```
robjects.reval("f<-function (x,y)  x+y")
r.f(3,4) #  returns 7
```

Packages and Functions

- one can add function with string

```
robjects.reval("f<-function (x,y)  x+y")
r.f(3,4) #  returns 7
```

- you could also load package

```
from rpy2.robjects.packages import importr
utils = importr("utils")
```

Packages and Functions

- one can add function with string

```
robjects.reval("f<-function (x,y)  x+y")
r.f(3,4) #  returns 7
```

- you could also load package

```
from rpy2.robjects.packages import importr
utils = importr("utils")
```

- you could write R code as a package

```
from rpy2.robjects.packages import STAP
string = """ square <- function(x) { return(x^2) } """
mypack = STAP(string, "mypack")
mypack.square(2) # returns 4
```

Graphics

Graphics

- we could use also graphics : for instance, let us do a PCA

```
r = robjects.r
m = r.matrix(r.rnorm(100), ncol=5)
pca = r.princomp(m)
r.plot(pca, main="Eigen values")
r.biplot(pca, main="biplot")
```

Graphics

- we could use also graphics : for instance, let us do a PCA

```
r = robjects.r
m = r.matrix(r.rnorm(100), ncol=5)
pca = r.princomp(m)
r.plot(pca, main="Eigen values")
r.biplot(pca, main="biplot")
```

- use `robjects.reval("graphics.off()")` to close the graphics

Graphics

- we could use also graphics : for instance, let us do a PCA

```
r = robjects.r
m = r.matrix(r.rnorm(100), ncol=5)
pca = r.princomp(m)
r.plot(pca, main="Eigen values")
r.biplot(pca, main="biplot")
```

- use `robjects.reval("graphics.off()")` to close the graphics
- another packages like ggplot could be use

Jupyter

Jupyter

- Jupiter is a fork of ipython

Jupyter

- Jupiter is a fork of ipython
- user-friendly interface to user kernels :

Jupyter

- Jupiter is a fork of ipython
- user-friendly interface to user kernels :
  - Python

Jupyter

- Jupiter is a fork of ipython
- user-friendly interface to user kernels :
  - Python
  - R

Jupyter

- Jupiter is a fork of ipython
- user-friendly interface to user kernels :
  - Python
  - R
  - Julia

Jupyter

- Jupiter is a fork of ipython
- user-friendly interface to user kernels :
  - Python
  - R
  - Julia
  - and lots of other : Ocaml, Bash, Cling

Jupyter

- Jupiter is a fork of ipython
- user-friendly interface to user kernels :
  - Python
  - R
  - Julia
  - and lots of other : Ocaml, Bash, Cling
- usages :

Jupyter

- Jupiter is a fork of ipython
- user-friendly interface to user kernels :
  - Python
  - R
  - Julia
  - and lots of other : Ocaml, Bash, Cling
- usages :
  - to do reproducible science

Jupyter

- Jupiter is a fork of ipython
- user-friendly interface to user kernels :
  - Python
  - R
  - Julia
  - and lots of other : Ocaml, Bash, Cling
- usages :
  - to do reproducible science
  - automatically generate documents

Jupyter

- Jupiter is a fork of ipython
- user-friendly interface to user kernels :
  - Python
  - R
  - Julia
  - and lots of other : Ocaml, Bash, Cling
- usages :
  - to do reproducible science
  - automatically generate documents
  - compare languages or methods

Jupyter

- Jupiter is a fork of ipython
- user-friendly interface to user kernels :
  - Python
  - R
  - Julia
  - and lots of other : Ocaml, Bash, Cling
- usages :
  - to do reproducible science
  - automatically generate documents
  - compare languages or methods
- Let's see some examples : En voiture Simone!

Conclusion

Conclusion

- Calling **Python** from **R** is possible but limited

Conclusion

- Calling **Python** from **R** is possible but limited
- Calling **R** from **Python** is not so hard :

Conclusion

- Calling **Python** from **R** is possible but limited
- Calling **R** from **Python** is not so hard :
  - numpy arrays could be used without cast

Conclusion

- Calling **Python** from **R** is possible but limited
- Calling **R** from **Python** is not so hard :
  - numpy arrays could be used without cast
  - have a look to **pandas** : it enables to used R datasets

Conclusion

- Calling **Python** from **R** is possible but limited
- Calling **R** from **Python** is not so hard :
  - numpy arrays could be used without cast
  - have a look to **pandas** : it enables to used R datasets
- Jupyter is a unified interface to call multiple kernels