JOURNAL OF APPLIED ECONOMETRICS
J. Appl. Econ. **24**: 698–704 (2009)
Published online in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/jae.1088

SOFTWARE REVIEWS

ECONOMETRICS WITH PYTHON

CHRISTINE CHOIRAT[a]* AND RAFFELLO SERI[b]

[a] *Department of Quantitative Methods, School of Economics and Business Management, Universidad de Navarra, Edificio de Bibliotecas, Pamplona, Spain*
[b] *Dipartimento di Economia, Università degli Studi dell'Insubria, Varese, Italy*

1. OVERVIEW

Many programming tools are available to the applied econometrician. There are high-level matrix languages mostly dedicated to econometrics (like GAUSS and Ox), to statistics (like Splus, R, or Stata) or to scientific computing broadly speaking (like Matlab, Octave, or Scilab). All these languages provide sufficient routines (in particular, related to matrix calculus and numerical optimization) to implement virtually any algorithm required by an econometric analysis.

Why should we then suggest Python as yet another possibility when it is not even designed for scientific computing? Let us make a parallel with what happened in applied statistics: Lisp-Stat, an extension of the general-purpose language Lisp, was gradually given up at the end of the 1990s in favor of the free and open-source R. In 2005 Jan de Leeuw, a prominent actor in the R community, wrote:

> Imagine the situation in which R was actually written in a combination of Python and C. There would be no need to replicate the general purpose programming tools and functions already available in Python. Thousands of Python extensions and hundreds of books and tutorials would immediately become available. ... The main difference between Lisp-Stat and S/R is that between a set of commands added to a large and popular general purpose programming language and a special-purpose little language for statistical computing. My personal opinion is that it is unfortunate that the statistical community made the choice that it made. (de Leeuw, 2005, pp. 3–4)

There is as yet no equivalent of R in applied econometrics. Therefore, the econometric community can still decide to go along the Python path.

* Correspondence to: Christine Choirat, Department of Quantitative Methods, School of Economics and Business Management, Universidad de Navarra, Edificio de Bibliotecas (Entrada Este), 31080 Pamplona, Spain. E-mail: cchoirat@unav.es

Python is a general-purpose scripting programming language. It is interpreted, object-oriented, and bears similarities with Perl (see Baiocchi, 2003, in particular p. 372, Section 2). Python was created in the early 1990s by Guido van Rossum with the objective of designing a simple, yet powerful language. Python is released under the so-called Python License (which is an approved open-source license) and can be freely downloaded and used. Unlike the GPL and more in the BSD style, this license allows for redistributing and even selling modified binary-only versions. Python is cross-platform and runs, among others, on all the major operating systems: Windows, Linux/Unix, Mac OS X. As of this writing, the current version of Python is 2.6.1 (released on December 4, 2008).

This review is organized as follows. In Section 2, we present the basics of Python. We perform a Monte Carlo simulation in Section 3. Section 4 shows that Python can easily interact with other software (R in particular). Section 5 concludes.

2. FIRST STEPS WITH PYTHON

2.1. Getting Python

The Python homepage is http://www.python.org/. Python can be compiled from source, but binaries that make the installation process straightforward are available for most operating systems. Enhanced Python distributions are available. These include Enthought Python Distribution, which bundles many scientific tools,[1] and ActivePython.[2]

Python is extremely well documented. On the Python homepage, it is possible to find a tutorial and manuals for advanced users (e.g., language reference, library reference, Python/C API), all written by van Rossum. A community-run Wiki (http://wiki.python.org/moin/) provides additional information (contributed tutorials, conferences, user groups, etc.). Several mailing lists (let us mention python-help in particular) and well-written free books (e.g., Pilgrim, 2004) are also available.

Python can be used either interactively in a shell (CPython by default, with a >>> prompt) or using a script file (with the `.py` extension). There are other Python shells, a particularly useful one being IPython,[3] which provides syntax highlighting, tab completion, powerful tools for interactive use (e.g., line editing, history of session states or the so-called 'magic' functions). Python comes with a script editor: IDLE (standing for 'Integrated DeveLopment Environment'). It provides a Tk user interface, syntax highlighting, tab completion, and debugging facilities. A detailed list of editors and integrated development environments can be found on the Python Wiki.

2.2. Data Types and Syntax

All the standard numeric data types (integer, float, complex) are available in Python. The associated operators respect the usual precedence rules. However, there is no native matrix type, and we have to rely on external tools (see Section 2.3). The generic sequence type includes, among others, strings (defined equivalently by single quotes and double quotes), lists (i.e., collections of arbitrary elements) and tuples (immutable lists). This generic type allows for indexing (i.e., selecting one element) and slicing (i.e., selecting using a range of indexes). Indices start at 0, and a slicing of the form `[i : j]` takes the elements of index k such that $i \leq k < j$. Instructions are

[1] See http://www.enthought.com/products/epd.php./.
[2] See http://www.activestate.com/Products/activepython/.
[3] By Fernando Pérez. See http://ipython.scipy.org/moin/.

separated by a carriage return or a semicolon. Comments start with the # symbol and stop at the end of the line.

Python is said to have a very clean syntax (of course, it is a matter of taste). For example, to read a csv file of data extracted from the Penn World Table (version 6.1)[4] similar to that of Baiocchi (2003, Section 4, pp. 373–374), we can use the standard csv module of Python. Variable names are read from the first line of the file (this is the default behavior, but it can be modified). Then we read all the rows and extract the relevant variables using their names (`country isocode` is the country code, `year` the year and `rgdpch` the real GDP per capita):

```
>>> import csv
>>> reader = csv.DictReader(open('pwt 61.csv'))
>>> for row in reader:
>>>     print row['country isocode'], row['year'], row['rgdpch']

AGO 1998 na
ALB 1998 2879.9000762
ARG 1998 11 638.992312
ARM 1998 2525.2046661
```

2.3. NumPy and SciPy

NumPy (see Oliphant, 2007) provides Python with an efficient implementation of N-dimensional arrays (that support indexing and slicing). Matrices are a special case of these arrays. SciPy[5] is a set of numeric tools based on NumPy. Both are free and released under the BSD License. Binaries are available for Windows, OS X, and most Linux distributions. Users familiar with R or Matlab can find equivalent NumPy/SciPy commands at http://mathesaurus.sourceforge.net/.

SciPy provides, among others, routines for: linear algebra (`linalg`), sparse matrices (`sparse`), optimization (constrained and unconstrained) and root-finding algorithms (`optimize`), basic statistical functions (`stats`), numeric integration and ODE solvers (`integrate`), special functions (`special`) and an easy way to include C/C++ code in a Python file (`weave`). The `optimize` submodule is very rich, and many tools are of possible interest: Levenberg–Marquardt least-squares, BFGS, Nelder–Mead, conjugate gradient, Powell's method, simulated annealing. Constrained optimization methods are also available: L-BFGS-B, COBYLA, and truncated Newton.

3. A MONTE CARLO EXAMPLE

As an illustration, we consider the Monte Carlo simulation of Cribari-Neto and Zarkos (1999) in R, studied again in Eddelbuettel (2000) in Octave.

3.1. Code and Comments

The vectorized code is the following (line numbers are not part of the code):

```
 1   import numpy as N, numpy.random as R, pylab as P
```

[4] The data can be retrieved at http://pwt.econ.upenn.edu/.

[5] *SciPy: Open Source Scientific Tools for Python*, by Eric Jones, Travis Oliphant, Pearu Peterson and others. See http://www.scipy.org/.

```
 2  def MCSimVec(r=1000):
 3      '''Monte Carlo simulation'''
 4      beta1, beta2, sigma2 = 7.3832, 0.2323, 46.852
 5      x = N.mat(N.loadtxt('data.txt')).T; T = len(x)
 6      X = N.hstack((N.ones((T, 1)), x))
 7      ysim = beta1 + beta2 * X[:, 1] + R.randn(T, r) * N.sqrt
   (sigma2)
 8      estimate = (X.T * X).I * X.T * ysim
 9      h = P.hist(estimate[1, :])
10      P.xlabel('b2'); P.ylabel('Density'); P.title('Histogram of
   b2')
11      P.show()
```

Line 1 imports NumPy, its `random` submodule, and the plotting module `pylab`, a Matlab-like interface to the interactive plotting library Matplotlib.[6] Each imported module corresponds to a namespace. A module can be imported in several ways that correspond to different function calls, as we can see in the following code, where we want to compute cos(π):

```
>>> import numpy; numpy.cos(numpy.pi)
>>> import numpy as N; N.cos(N.pi)
>>> from numpy import cos, pi; cos(pi)
>>> from numpy import *; cos(pi) # imports all the functions of
numpy
```

In line 2, we define a function (with a default value of 1000). A specificity of Python is that blocks of statements are determined by their indentation level. These compound statements come from control flow structures (`if-elif-else`, `while`, `for`) and function and class definitions (`def`, `class`). Line 3 is a docstring, that is, an optional string defined by triple quotes which provides documentation for the function, either in an interactive session (the output of `help(MCSimVec)`) or in HTML format using the `pydoc` standard Python module. Line 4 shows multiple assignment. In line 5, we load the data (taken from Cribari-Neto and Zarkos, 1999) from a text file and transform it to a ($T \times 1$) matrix. In the next few lines, we generate the dependent variable and calculate OLS estimates. Note that the transpose and the inverse of a NumPy matrix `X` are given by `X.T` and `X.I`, respectively. Lines 9–11 create and display a histogram.

To get a loop-based version, we could replace line 8 with

```
M = (X.T * X).I * X.T
estimate = N.mat(N.empty((2, r)))
for j in range(0, r):
   estimate[:, j] = M * ysim[:, j]
```

3.2. Speed Benchmarks

We compared the performance of NumPy (version 1.2.1) with that of other free matrix languages: Octave (version 3.0.1), Ox (version 5.10 for Linux, free for academic purposes only and known

[6] By John Hunter. See http://matplotlib.sourceforge.net/.

Table I. Execution times (in seconds)

r	Vectorized				Loop			
	NumPy	Octave	Ox	R	NumPy	Octave	Ox	R
1 000	0.02	0.00	0.00	0.02	0.07	0.14	0.01	0.03
10 000	0.11	0.07	0.06	0.26	0.69	1.34	0.08	0.42
100 000	0.99	0.65	0.59	1.92	7.09	13.39	0.73	3.67
500 000	4.64	3.26	2.91	9.44	34.58	66.81	3.67	17.62

for being fast; see Doornik, 2006), and R (version 2.8.1, see R Development Core Team, 2008). We implemented both a vectorized and a loop-based version.[7] From Table I, we see that the vectorized Python code performs well. It is almost twice as fast as R and only about 30% slower than Octave. The loop-based version is not very efficient, and loops should be avoided as much as possible. When they cannot, it is possible to extend Python with C/C++ or Fortran. The `weave` submodule of NumPy makes the extension very easy by allowing for inclusion of C/C++ code directly in Python and by automatically taking care of the compilation procedure.

4. INTERACTION WITH OTHER SOFTWARE

One of the strengths of Python is its ability to interact with other software, gnuplot and R being of particular interest.

There is an interface[8] between Python and gnuplot, so that all the examples of Racine (2006) can be replicated. The following code sample opens a gnuplot session `g` and passes a gnuplot command as a string:

```
>>> import Gnuplot; g = Gnuplot.Gnuplot(); g('plot sin(x)')
```

There is also an interface between Python and R. The Python module RPy[9] provides a way to manipulate R objects and functions from within Python (including the graphical functions and the many random generators of R). Importing the RPy module in Python amounts to importing an object called `r` which contains all the R functions. R code can be executed as a string, but R functions can also be called by prefixing the R function name with '`r.`'.[10] It is also possible to load R packages (even user-defined ones).

```
>>> import rpy; rpy.r('cos(100)'); rpy.r.cos(100)
>>> rpy.r.library('survival')
```

There is a conversion between R and Python (in both directions). The level of conversion can be selected. The RPy documentation provides detailed information about the mechanism. As an illustration, let us consider the binary choice model of Racine and Hyndman (2002, Section 6.2,

[7] Execution times were measured with `time.clock()` for Python, `cputime()` for Octave, `timer()` for Ox, and `system.time()` for R. In Ox, we do not measure the CPU time but rather the elapsed time. The benchmarks were performed on a laptop computer running the Linux operating system (kernel 2.6.27) with a 1.50 Ghz Intel Celeron M processor and 1.5 GB of RAM.

[8] By Michael Haggerty. See http://gnuplot-py.sourceforge.net/.

[9] By Walter Moreira and Gregory R. Warnes. See http://rpy.sourceforge.net/.

[10] Note that we have to use two prefixes: the first one corresponds to the RPy module and the second one to the call of an R function.

p. 183). The result `probitfit` is a Python dictionary (on the R side, it is a list).[11] The keys of the Python dictionary correspond to the arguments of the R list.

```
>>> rpy.r('x <- rnorm(100)')
>>> rpy.r('y <- as.integer(1 + x + rnorm(100) < 0)')
>>> probitfit = rpy.r('glm(y ~ x, family=binomial(link=
probit))')
>>> probitfit['coefficients']
{'x': −0.65845162756748177, '(Intercept)': −1.0254885172602837}
```

5. FINAL REMARKS

Python lacks modules specifically dedicated to econometric analysis. This is also true of Octave (see Eddelbuettel, 2000, Section 6, p. 541), Scilab (see Mrkaic, 2001, Section 6, p. 559), and even R to a lesser extent (see Racine and Hyndman, 2002, p. 177). But, unlike them, Python is a full-featured language (with facilities for exception handling, debugging, profiling, testing, threading, parallel computing, etc.), already production-stable and with a promising future (it is becoming a standard in the computer industry: Google uses Python extensively and has even hired Guido van Rossum). It is customary to say that it comes with 'batteries included' and many contributed categorized modules are available from the Python Package Index (http://pypi.python.org/pypi/). There are, for instance, modules dealing with operating system interface, regular expression processing, date and time manipulation, data compression or even web programming. Moreover, Python is also a suitable choice for numerical computing, as we have seen in the benchmarks. It is therefore possible to perform a whole econometric analysis (e.g., database connection, data manipulation, model estimation, simulation, graphical output, upload of zipped results on ftp or web server) using one language only, which dramatically increases programming productivity.

Since there is no free programming language that can be considered a lingua franca of applied econometrics, choosing Python and writing or translating econometric routines may be worth the effort.

ACKNOWLEDGEMENTS

The authors would like to thank James MacKinnon, the Software Review Editor, for his detailed comments and suggestions on a previous version of this review.

REFERENCES

Baiocchi G. 2003. Managing econometric projects using Perl. *Journal of Applied Econometrics* **18**(3): 371–378.

Cribari-Neto F, Zarkos S. 1999. R: yet another econometric programming environment. *Journal of Applied Econometrics* **14**(3): 319–329.

de Leeuw J. 2005. On abandoning XLISP-STAT. *Journal of Statistical Software* **13**(7): 1–5.

Doornik J. 2006. *An Object-Oriented Matrix Language—Ox 4* (4th edn). Timberlake Consultants Press: London.

[11] A dictionary is an associative array, i.e. a list of values indexed by a list of keys.

Eddelbuettel D. 2000. Econometrics with Octave. *Journal of Applied Econometrics* **15**(5): 531–542.

Mrkaic M. 2001. Scilab as an econometric programming system. *Journal of Applied Econometrics* **16**(4): 553–559.

Oliphant T. 2007. Python for scientific computing. *Computing in Science and Engineering* **9**(3): 10–20.

Pilgrim M. 2004. *Dive into Python*. Apress. http://diveintopython.org/ [29 March 2009].

R Development Core Team. 2008. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing: Vienna. http://www.R-project.org/ [29 March 2009].

Racine J. 2006. gnuplot 4.0: a portable interactive plotting utility. *Journal of Applied Econometrics* **21**(1): 133–141.

Racine J, Hyndman R. 2002. Using R to teach econometrics. *Journal of Applied Econometrics* **17**(2): 175–189.