IPYTHON AND MATPLOTLIB

Python for computational science

22 – 24 September 2014

CINECA

m.cestari@cineca.it

Introduction

- plotting the data gives us **visual feedback**

- Typical workflow:
  - write a python program to parse data
  - pass the data to a plot tool to show the results

- with **Matplotlib** we can achieve the same result in a single script and with more flexibility

Ipython (1)

- **improves** the interactive mode usage
  - tab completion for functions, modules, variables, files
  - Introspection, accessibile with "?", of objects and function
  - %run, to execute a python script
  - filesystem navigation (cd, ls, pwd) and bash like behaviour (cat)
    - `!cmd` execute command in the shell
  - Debugging and profiling

Ipython (2)

- **improves** the interactive mode usage

  - Search commands (Ctrl-n, Ctrl-p, Ctrl-r)

    (don't work with notebook)

  - magic commands:

    - %magic (list them all)
    - %whos
    - %timeit
    - %logstart

Ipython (3)

Ipython is recommended over python for interactive usage:

- Has a matplotlib support mode

```
$ ipython --pylab
```

- no need to `import` any modules; merges matplotlib.pyplot (for plotting) and numpy (for mathematical functions)
- spawn a thread to handle the GUI and another one to handle the user inputs
  - every plot command triggers a plot update

Ipython (4)

- **HTML notebook**
  - You can share .ipynb files

```
$ ipython notebook
```

  - Well integrates with **matplotlib**

```
%matplotlib inline
```

- **QT GUI console**

```
$ ipython qtconsole
```

Matplotlib:

- makes use of Numpy to provide good performance with large data arrays
- allows **publication quality** plots
- since it's a Python module can be easily **integrated** in a Python program

Module import

Let us be consistent with the official documentation

```
$ (i)python
>>> import matplotlib.pyplot as plt
```

Matplotlib 1st example

```
>>> # following imports are not necessary if --pylab
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> plt.interactive('on') # set interactive
                          # no need with --pylab
>>> x = np.arange(0,7,0.00001)
>>> plt.plot(x,x**3) # x,y values of the plot
[<matplotlib.lines.Line2D object at 0xa1750cc>]
>>> plt.show()
```

Matplotlib: multiple line plot

```
>>> x = np.arange(0,7,0.00001)
>>> plt.plot(x,x**3)
>>> plt.plot(x,x**2)
>>> plt.show()
```

Or by passing multiple (x,y) arguments to the plot function

```
>>> x = np.arange(0,7,0.00001)
>>> plt.plot(x,x**3, x,x**2)
>>> plt.show()
```

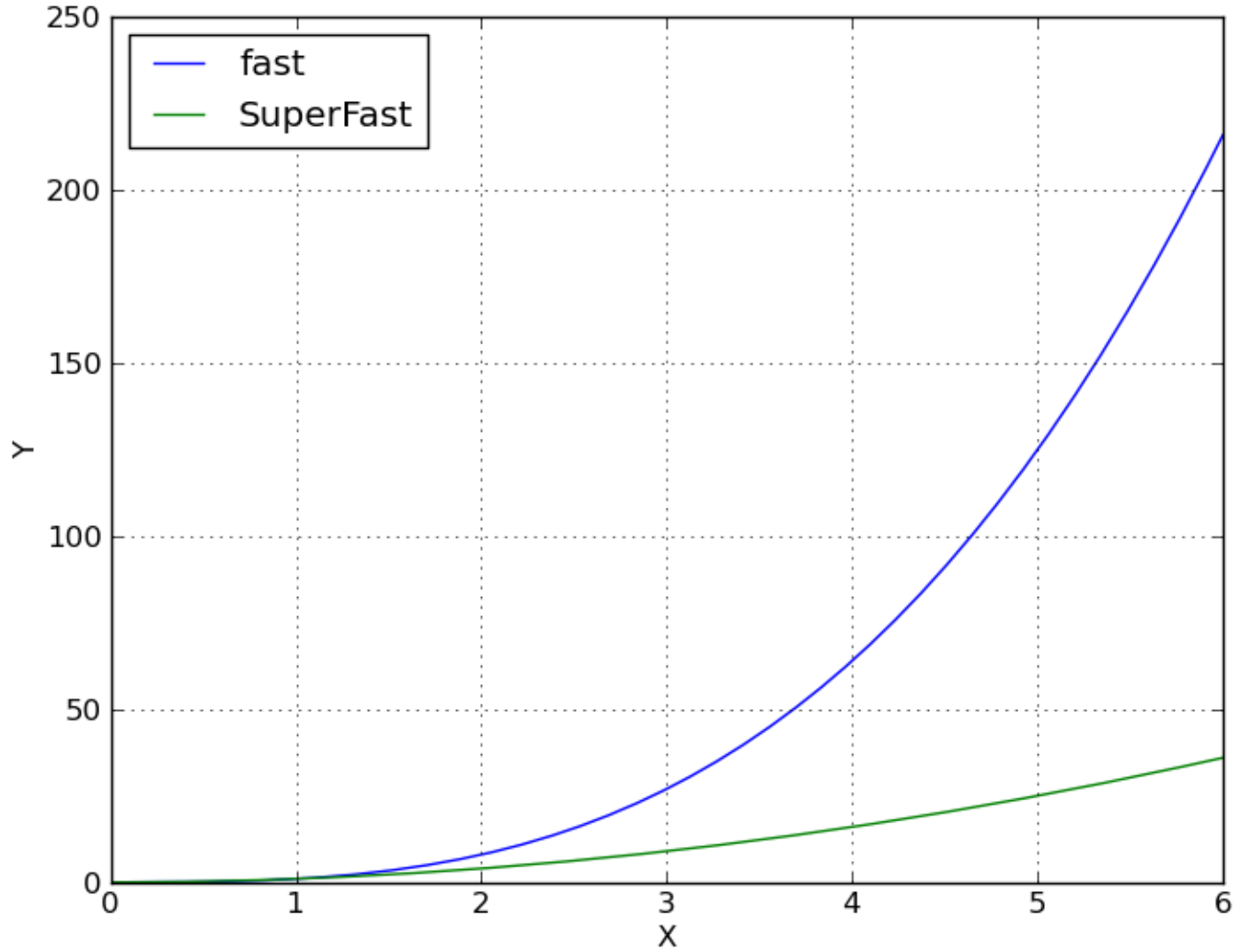
Plot control commands

Classic plot interaction is available

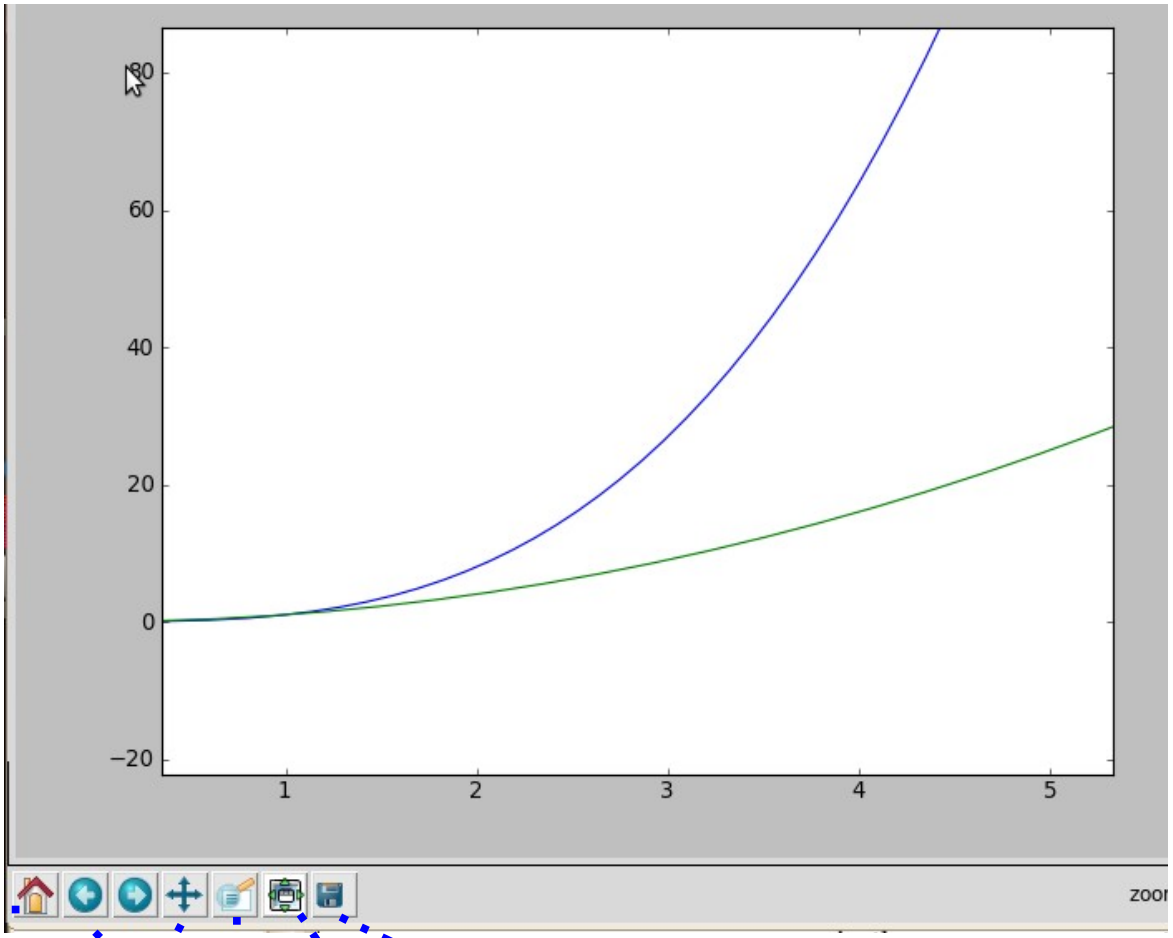
```
>>> plt.grid()
>>> plt.axis() # shows the current axis limits values
>>> plt.axis([0,5,0,10]) #[xmin,xmax,ymin,ymax]
>>> plt.xlabel('This is the X axis')
>>> plt.title('Outstanding title here')
>>> plt.legend(['Fast', 'SuperFast'],loc=2)
>>> plt.savefig('plot123.png', dpi=250)
```

extension determines the file format

Plot example



Plot window



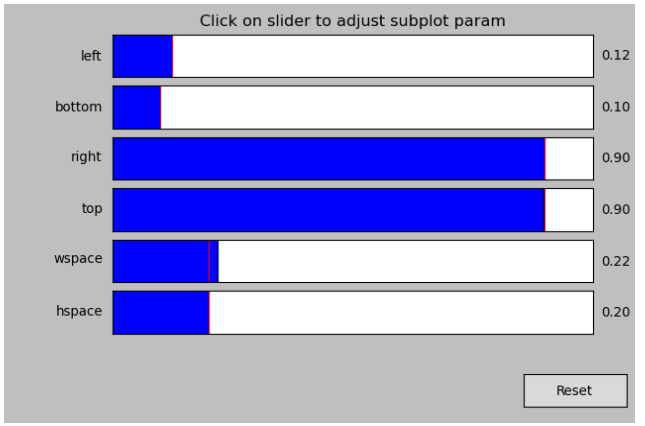
First view of the plot

Back and forward among views

Move (left clik) and zoom (right click)

Draw a view of the plot

Save file



Object-oriented interface (1)

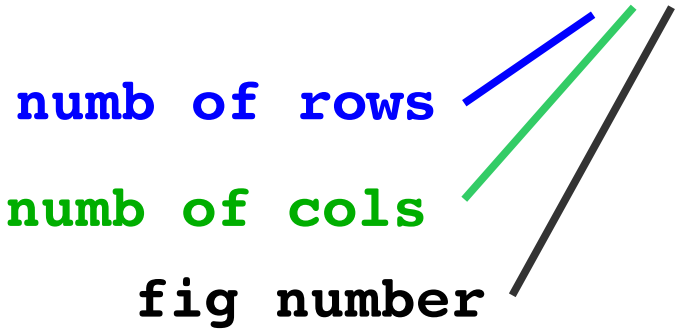
A figure is composed by a hierarchical series of Matplotlib objects

- **Figure**: Container for one or more Axes instances
- **Subplot (Axes)**: The rectangular areas that holds the basic elements, such as lines, text, and so on
- **Lines**: real plotting objects (lines, histograms, …)

Object-oriented interface (2)

```
>>> fig = plt.figure()

>>> ax = fig.add_subplot(221)
```



221	222
223	224

Object-oriented interface

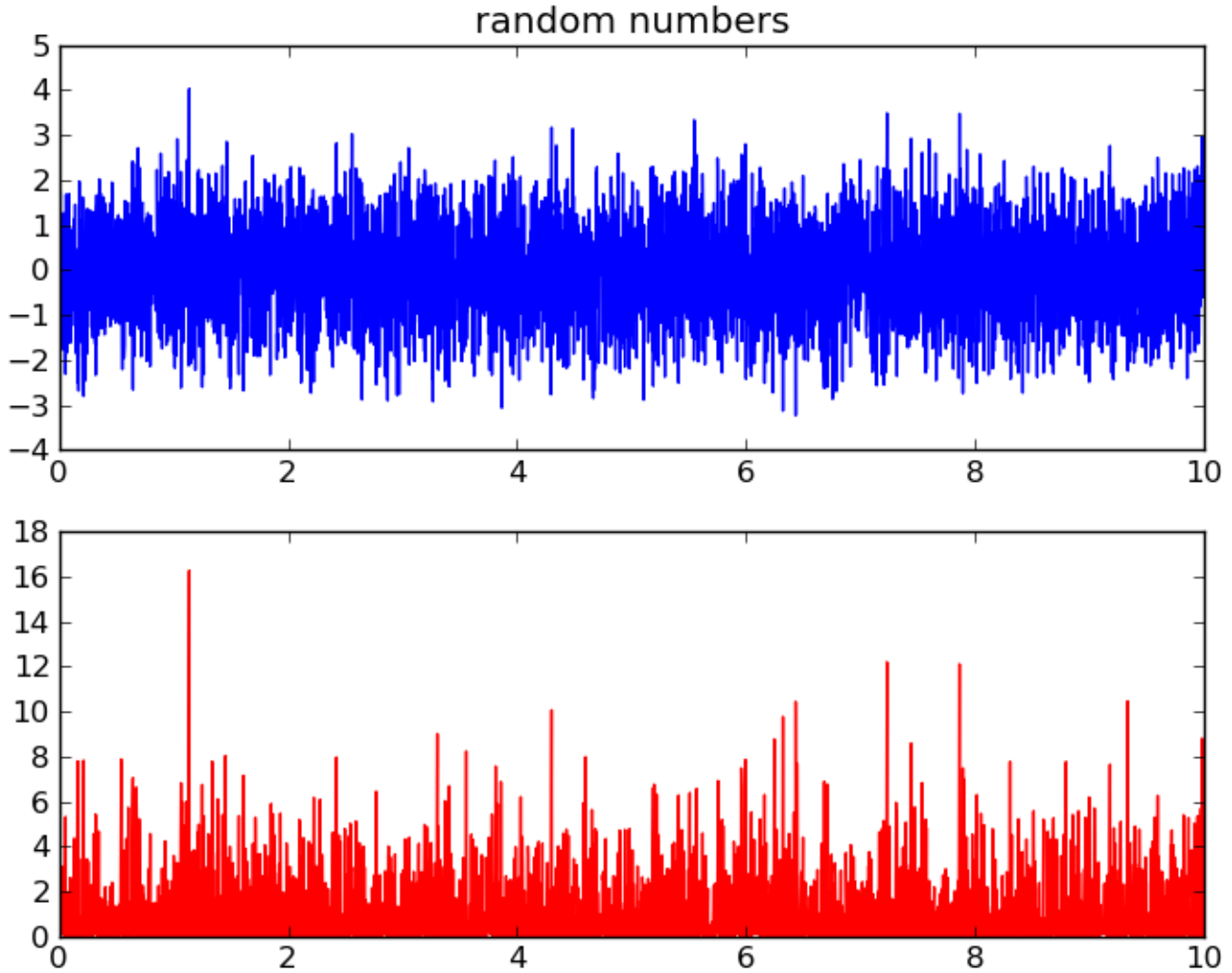
- *OO* use of matplotlib makes the code **more explicit** and allows a lot more **customizations**

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> x = np.arange(0, 10, 0.1)
>>> y = np.random.randn(len(x))
>>> fig = plt.figure()       # instance of the fig obj
>>> ax = fig.add_subplot(111) # instance of the axes
                              # obj
>>> l, m = ax.plot(x, y, x, y**2) # returns a tuple of obj
>>> l.set_color('blue')
>>> m.set_color('red')
>>> t = ax.set_title('random numbers')
>>> plt.show()
```

Object-oriented interface: multiple plot

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> x = np.arange(0, 10, 0.001)
>>> y = np.random.randn(len(x))
>>> fig = plt.figure()       # instance of the fig obj
>>> ax = fig.add_subplot(211)  # two axes instances in
>>> ax2 = fig.add_subplot(212) # the same column
>>> l, = ax.plot(x, y) # returns a tuple of obj
>>> m, = ax2.plot(x, y**2) # returns a tuple of obj
>>> l.set_color('blue')
>>> m.set_color('red')
>>> t = ax.set_title('random numbers')
>>> plt.show()
```

Object-oriented interface: multiple plot

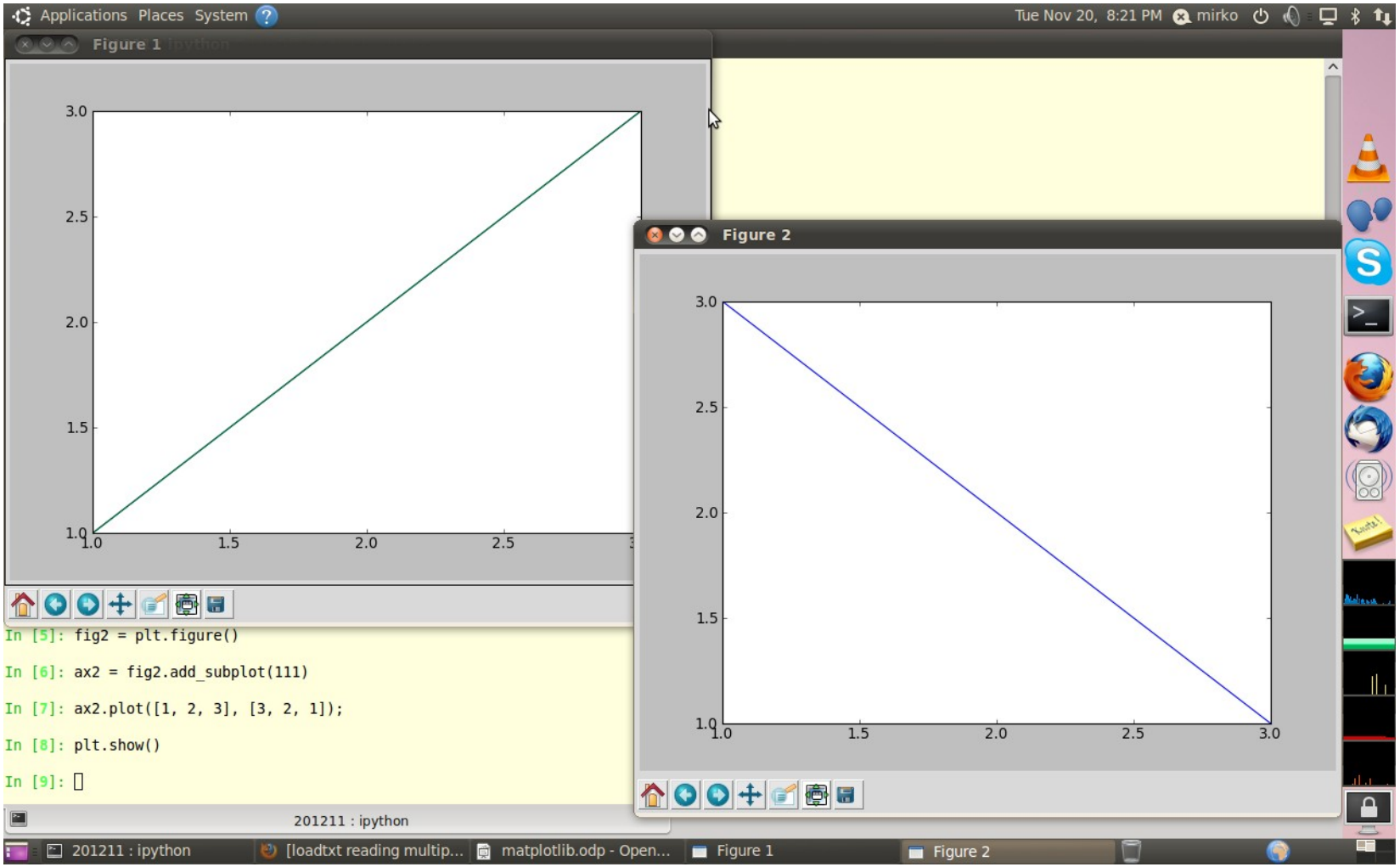


Object-oriented interface

- Multiple figures are allowed

```
>>> import matplotlib.pyplot as plt
>>> fig1 = plt.figure()
>>> ax1 = fig1.add_subplot(111)
>>> ax1.plot([1, 2, 3], [1, 2, 3]);
>>> fig2 = plt.figure()
>>> ax2 = fig2.add_subplot(111)
>>> ax2.plot([1, 2, 3], [3, 2, 1]);
>>> plt.show()
```

Object-oriented interface



Other examples

http://matplotlib.sourceforge.net/gallery.html

To summarize

- This was a very brief introduction to Matplotlib.
- We don't need to cover everything; just go with your needs
- It can be used interactively, *a la* Matlab, or Object-oriented
- It can be fully integrated in a Python program;
  - ➔ your analysis code can be integrated with a plot tool, tailored to the application needs

Bibliography

http://matplotlib.sourceforge.net/contents.html

Matplotlib for Python developers (Sandro Tosi, Packt Publishing Ltd., 2009)