# Week 2: Computing in python I, Some useful packages and tools

MSc/MRes CMEE 2014-15


Samraat Pawar

Imperial College London


October 16, 2014

# A FEW THINGS FIRST

- Any issues with version control and pushing your Week 1? – DO NOT clone `cmee2014masterepo` into `cmeecoursework`
- If you don't have `ipdb` (let me know if you still don't get `ipdb`):

```
sudo apt-get install python-ipdb
```

- Let's discuss the `dictionary.py` and yesterday's Regex exercises in the afternoon
- We will do a quick recap of what we did in the whole week tomorrow morning (back in CPB!)

# USEFUL PYTHON PACKAGES

- These are always available as standard libraries (just require `import` from within python or ipython):
  - `io`: file input-output with `*.csv`, `*.txt`, etc.
  - `subprocess`: to run other programs, including multiple ones at the same time, including operating system-dependent functionality
  - `sqlite3`: for manipulating and querying `sqlite` databases
  - `math`: for mathematical functions

- These you will have to install in terminal using `sudo apt-get install python-packagename` (as you did for `ipdb` earlier)):
  - `scipy`: for scientific computing
  - `matplotlib`: for plotting (very matlab-like, requires `scipy`) (all packaged in `pylab`)
  - `scrapy`: for writing web spiders that crawl web sites and extract data from them
  - `beautifulsoup`: for parsing HTML and XML (can do what `scrapy` does)
  - `biopython`: for biological computation, including bioinformatics

# USING python TO PATCH TOGETHER WORK-FLOWS, I

- You can use python to build an automated work-flow that involves multiple applications (R, LaTeX, bash, etc.)
- For example, you could, in theory, write a single Python script to generate and update your MSc/MRes dissertation, tables, plots, and all!
- the subprocess modules will enable you to do this
- Let's try – first launch ipython, then cd to your python code directory, and type:

```
import subprocess
subprocess.os.system("geany boilerplate.py")
subprocess.os.system("gedit ../Data/TestOaksData.csv")
subprocess.os.system("python boilerplate.py") # A bit silly!
```

- Easy as pie!

# USING `python` TO PATCH TOGETHER WORK-FLOWS, II

- Similarly, to compile your Latex document (using `pdflatex` in this case):

```
subprocess.os.system("pdflatex yourlatexdoc.tex")
```

- You can also do this (instead of using *subprocess.os*):

```
subprocess.Popen("geany boilerplate.py", shell=True).wait()
```

- You can also run `R` in a similar way (More on this in Python Week 2)

- Why am I telling you all this now? – Because of the next two weeks (esp. GIS)

# USING python TO PATCH TOGETHER WORK-FLOWS, III

- You can also use `subprocess.os` to make your code OS (Linux, Windows, Mac) independent
- For example to assign paths:

```
subprocess.os.path.join('directory', 'subdirectory', 'file')
```

- The result would be appropriately different on Windows!
- Note that in all cases you can "catch" the output of `subprocess` so that you can then use the output within your python script:

```
MyPath = subprocess.os.path.join('directory', 'subdirectory', 'file')
```

- Explore what `subprocess` can do by tabbing `subprocess.` (so also for submodules, e.g., `subprocess.os.` an then tab)

# NUMERICAL COMPUTING IN `Python`

- The python package/library `scipy` can help you do serious number crunching:
    - Linear algebra
    - Numerical integration
    - Fourier transforms
    - Interpolation
    - Special functions (Incomplete Gamma, Bessel, etc.)
    - Random numbers and statistical functions
- Let's have a quick look at `scipy`
- In the following, we will use the `array` class in `scipy` for data manipulations and calculations (I mentioned this on the first day)
- Scipy arrays are similar in some respects to python lists, but are more naturally multidimensional, homogeneous in type, and allow efficient manipulations

# NUMERICAL COMPUTING IN Python

```
In [1]: import scipy

In [2]: a = scipy.array(range(5))

In [3]: a
Out[3]: array([0, 1, 2, 3, 4])

In [4]: a = scipy.array(range(5), dtype = float)

In [5]: a
Out[5]: array([ 0.,  1.,  2.,  3.,  4.])

In [6]: a.dtype
Out[6]: dtype('float64')

In [7]: x = scipy.arange(5)

In [8]: x
Out[8]: array([0, 1, 2, 3, 4])

In [9]: x = scipy.arange(5.)

In [10]: x
Out[10]: array([ 0.,  1.,  2.,  3.,  4.])
```

# NUMERICAL COMPUTING IN Python

```
In [11]: x.
x.T               x.conj            x.fill
x.nbytes          x.round           x.take
x.all             x.conjugate       x.flags
x.ndim            x.searchsorted    x.tofile
x.any             x.copy            x.flat
x.newbyteorder    x.setfield        x.tolist
x.argmax          x.ctypes          x.flatten
x.nonzero         x.setflags        x.tostring
x.argmin          x.cumprod         x.getfield
x.prod            x.shape           x.trace
x.argsort         x.cumsum          x.imag
x.ptp             x.size            x.transpose
x.astype          x.data            x.item
x.put             x.sort            x.var
x.base            x.diagonal        x.itemset
x.ravel           x.squeeze         x.view
x.byteswap        x.dot             x.itemsize
x.real            x.std
x.choose          x.dtype           x.max
x.repeat          x.strides
x.clip            x.dump            x.mean
x.reshape         x.sum
x.compress        x.dumps           x.min
x.resize          x.swapaxes

In [11]: x.tolist()
Out[11]: [0.0, 1.0, 2.0, 3.0, 4.0]

In [12]: x.shape
Out[12]: (5,)
```

```
In [14]: mat = scipy.array([[0, 1], [2, 3]])

In [16]: mat.shape
Out[16]: (2, 2)

In [17]: mat[1]
Out[17]: array([2, 3])

In [18]: mat[0,0]
Out[18]: 0

In [19]: mat.ravel() # flatten!
Out[19]: array([0, 1, 2, 3])

In [20]: scipy.ones((4,2))
Out[20]:
array([[ 1.,  1.],
       [ 1.,  1.],
       [ 1.,  1.],
       [ 1.,  1.]])

In [21]: scipy.zeros((4,2))
Out[21]:
array([[ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.]])
```

# NUMERICAL COMPUTING IN Python

```
In [22]: scipy.identity(4)
Out[22]:
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])

In [23]: m = scipy.identity(4)

In [24]: m.reshape((8, 2))
Out[24]:
array([[ 1.,  0.],
       [ 0.,  0.],
       [ 0.,  1.],
       [ 0.,  0.],
       [ 0.,  0.],
       [ 1.,  0.],
       [ 0.,  0.],
       [ 0.,  1.]])

In [25]: m.fill(16)

In [26]: m
Out[26]:
array([[ 16.,  16.,  16.,  16.],
       [ 16.,  16.,  16.,  16.],
       [ 16.,  16.,  16.,  16.],
       [ 16.,  16.,  16.,  16.]])
```

# NUMERICAL COMPUTING IN Python

- Let's perform some common operations on arrays:

```
In [1]: import scipy

In [2]: mm = scipy.arange(16)

In [3]: mm = mm.reshape(4,4)

In [4]: mm
Out[4]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])

# NOTE: Rows first

In [5]: mm.transpose()
Out[5]:
array([[ 0,  4,  8, 12],
       [ 1,  5,  9, 13],
       [ 2,  6, 10, 14],
       [ 3,  7, 11, 15]])

In [6]: mm + mm.transpose()
Out[6]:
array([[ 0,  5, 10, 15],
       [ 5, 10, 15, 20],
       [10, 15, 20, 25],
       [15, 20, 25, 30]])
```

# NUMERICAL COMPUTING IN Python

```
In [7]: mm - mm.transpose()
Out[7]:
array([[ 0, -3, -6, -9],
       [ 3,  0, -3, -6],
       [ 6,  3,  0, -3],
       [ 9,  6,  3,  0]])

In [8]: mm * mm.transpose()
## Elementwise!

Out[8]:
array([[  0,   4,  16,  36],
       [  4,  25,  54,  91],
       [ 16,  54, 100, 154],
       [ 36,  91, 154, 225]])

In [9]: mm / mm.transpose()
Warning: divide by zero encountered in divide

# Note the integer division
Out[9]:
array([[0, 0, 0, 0],
       [4, 1, 0, 0],
       [4, 1, 1, 0],
       [4, 1, 1, 1]])
```

# NUMERICAL COMPUTING IN Python

```
In [10]: mm * scipy.pi
Out[10]:
array([[  0.      ,   3.14159,   6.28318531,   9.42477796],
       [ 12.566370,  15.70796,  18.84955592,  21.99114858],
       [ 25.132741,  28.27433,  31.41592654,  34.55751919],
       [ 37.699111,  40.84070,  43.98229715,  47.1238898 ]])

In [11]: mm.dot(mm) # MATRIX MULTIPLICATION
Out[11]:
array([[ 56,  62,  68,  74],
       [152, 174, 196, 218],
       [248, 286, 324, 362],
       [344, 398, 452, 506]])
```

- We can do a lot more (but won't!) by importing the `linalg` sub-package: `import scipy.linalg`
- Two other particularly useful `scipy` sub-packages are:
  - `scipy.integrate` (what will I need this for?)
  - `scipy.stats` (why not use `R` for this?)

Let's take a quick spin in `scipy.stats`!

```
In [18]: import scipy.stats

In [19]: scipy.stats.
scipy.stats.arcsine                  scipy.stats.lognorm
scipy.stats.bernoulli                scipy.stats.mannwhitneyu
scipy.stats.beta                     scipy.stats.maxwell
scipy.stats.binom                    scipy.stats.moment
scipy.stats.chi2                     scipy.stats.nanstd
scipy.stats.chisqprob                scipy.stats.nbinom
scipy.stats.circvar                  scipy.stats.norm
scipy.stats.expon                    scipy.stats.powerlaw
scipy.stats.gompertz                 scipy.stats.t
scipy.stats.kruskal                  scipy.stats.uniform

In [19]: scipy.stats.norm.rvs(size = 10) # 10 samples from
N(0,1)
Out[19]:
array([-0.951319, -1.997693,  1.518519, -0.975607,  0.8903,
       -0.171347, -0.964987, -0.192849,  1.303369,  0.6728])

In [20]: scipy.stats.norm.rvs(5, size = 10)
# change mean to 5
Out[20]:
array([ 6.079362,  4.736106,  3.127175,  5.620740,  5.98831,
        6.657388,  5.899766,  5.754475,  5.353463,  3.24320])
```

```
In [21]: scipy.stats.norm.rvs(5, 100, size = 10)
# change sd to 100
Out[21]:
array([ -57.886247,   12.620516,  104.654729,  -30.979751,
         41.775710,  -31.423377,  -31.003134,   80.537090,
          3.835486,  103.462095])

# Random integers between 0 and 10
In [23]: scipy.stats.randint.rvs(0, 10, size =7)
Out[23]: array([6, 6, 2, 0, 9, 8, 5])
```

# NUMERICAL COMPUTING IN `Python`

- Let's look at an example using `scipy.integrate`
- Create `LV1.py` in your `Week2/Code` directory

# READINGS AND RESOURCES

- `www.matplotlib.org/`
- For SciPy, the official documentation is best:
  `docs.scipy.org/doc/scipy/reference/`
  Read about the scipy modules you think will be important to you...
- Many illustrative examples at
  `http://wiki.scipy.org/Cookbook` (including Lotka-Volterra!)
- In general, good module-specific cookbooks are out there (e.g., `biopython`)

# PRACTICAL 2 I

- Get Pracs 0 and 1 sorted out!
- Don't forget to bring today's (functional) scripts under version control `LV1.py`

# Practical 2 II

- Convert `LV1.py` into another script called `LV2.py` that does the following:
  1. Take arguments for the four LV model parameters `r, a, m ,e` from the commandline

     ```
     LV2.py arg1 arg2 ... etc
     ```

  2. Runs the Lotka-Volterra model with prey density dependence $rx(1 - \frac{x}{K})$
  3. Saves the plot as `.pdf` in an external results directory (`Week2/Results`)
  4. The chosen parameter values should show in the plot (e.g., $r = 1, a = .5$, etc)

- You change time length `t` too

- Also include a script called `run_LV2.py` in `Code` that will run `LV2.py` with appropriate arguments

# PRACTICAL 2 III

- Extra credit if you also choose appropriate values for the paramaters such that both predator and prey persist under in model with prey density dependence
- Extra-extra credit if you can write a recursion version of the model in discrete time (what's this?) – it should do everything that `run_LV2.py` does
- Extra-extra-extra credit if you can write a recursion versiaon of the model in discrete time with a random gaussian fluctuation at each time-step (use `scipy.stats`)

# PRACTICAL 2 IV

- Complete the code *blackbirds.py* that you find in the master repo (necessary data file is also there)