Python Programming for Machine Learning

summer term 2013

Day 1: Python Basics and Numpy Intro

Daniel Bartz

2.4.-4.4.2013

What is Scientific Computing?

Using computers to analyze and solve scientific problems!

- analyze data
- numerical simulations
- develope prototypes
- visualize results
- ...

Tools for Scientific Computing: MATLAB

Advantages:

- very easy to use
- comfortable IDE
- widely used
- many toolboxes available
- interpreted – easy to play around
- ...

Disadvantages:

- expensive
- you need a licence for every toolbox
- not a full programming language
- not really object-oriented
- ...

Tools for Scientific Computing: C++ (or Fortran, Java...)

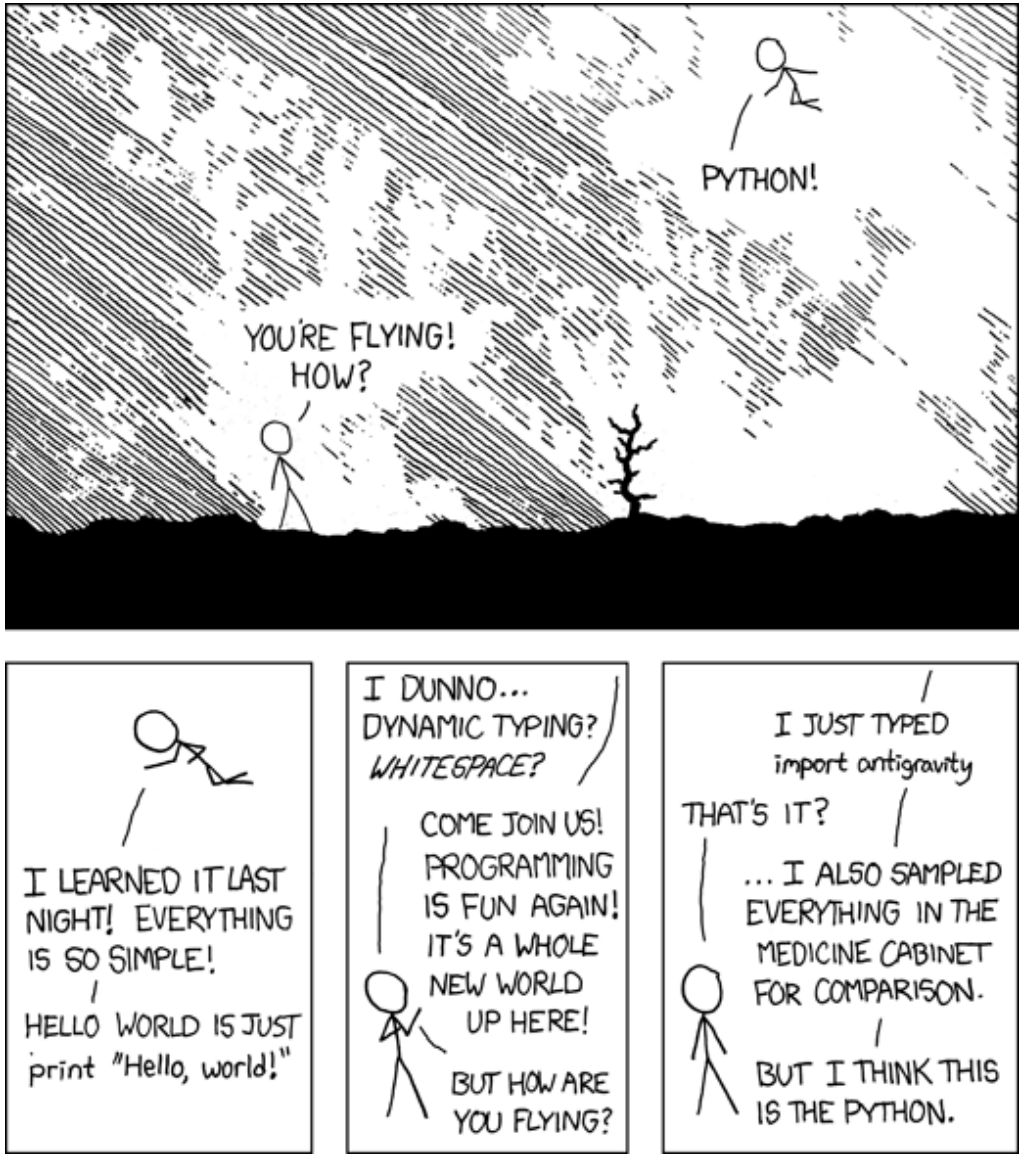
Advantages:

- Fast
- full programming language
- freely available
- ...

Disadvantages:

- less intuitive
- compiled – not as easy to play around
- ...

Tools for Scientific Computing: Python



http://xkcd.com/353/

Tools for Scientific Computing: Python

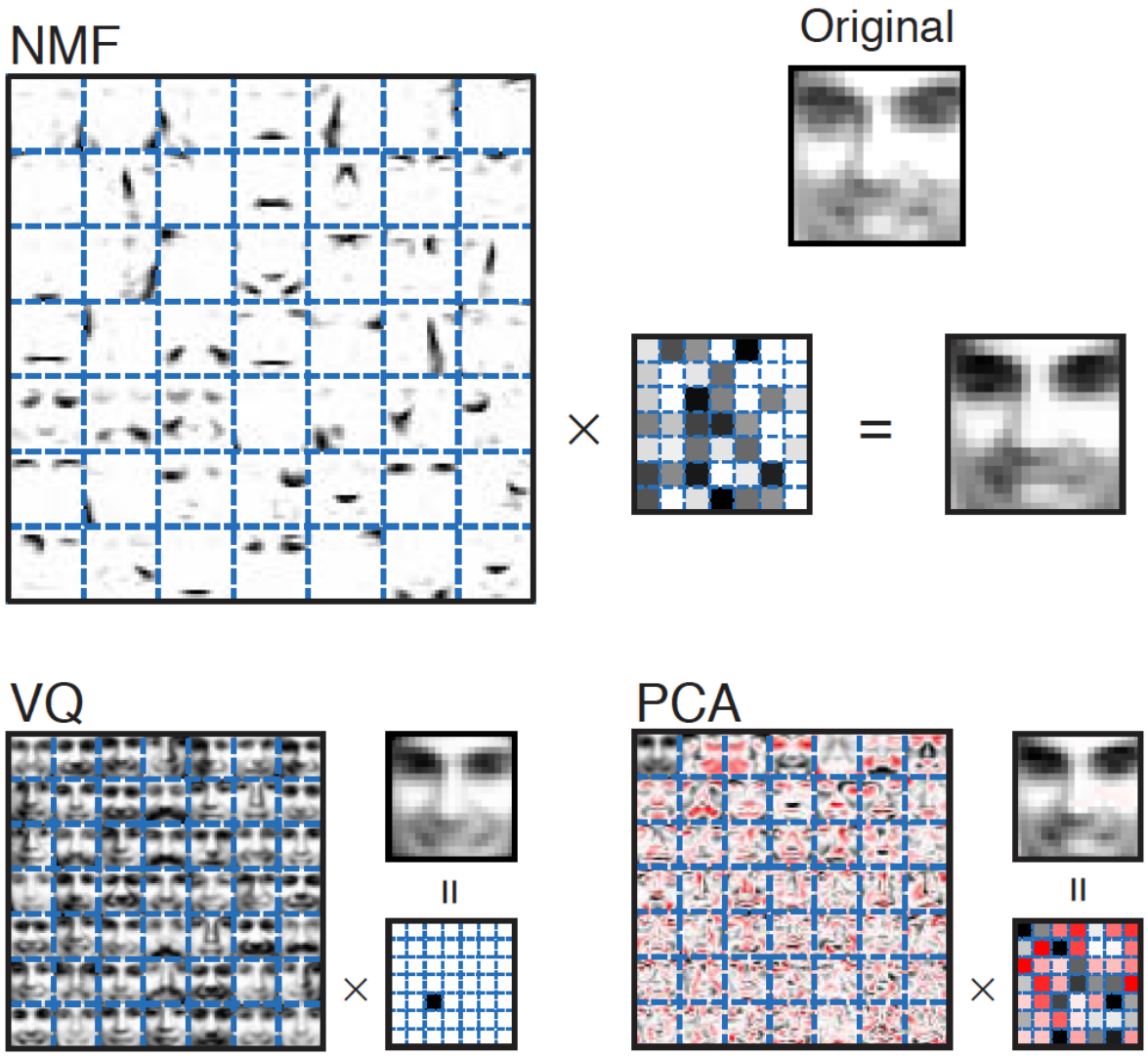
Advantages:

- Faster than Matlab
- full object-oriented programming language
- open source
- many free libraries available
- growing community
- interpreted – easy to play around
- if you can do it in Matlab, you can do it in Python!
- ...

Disadvantages:

- MATLAB may still be a bit easier and have a more comfortable IDE
- C++ may be a bit faster

Course Topics: Science...



Machine Learning: Non-negative matrix factorization

Course Topics: ... and Computing!

topics:

- the IPython shell
- Python basics
- matplotlib/pylab: visualizing results
- numpy: efficient numerics
- scipy: many routines needed for science

(hopefully the IRB gets scipy to run tomorrow or Thursday...)

IPython – python as a pocket calculator

- `print 'Hello, world!'`
- `1+2; 1/3; 3./5; 4*5; a = 3%2; a = sin(5); c = a`
- `import math; a = math.sin(5)`
- tab completion: explore packages
- ?: get help
- system commands:
  `ls, mkdir test, cd sctest, cd .., rm -r sctest, cd sctest`
- magic commands:
  `%lsmagic, %run wc.py, %timeit math.sin(12), %debug, %reload_ext`
- and much more...

Python basics I - variable types

- standard variable types:

```
a = 1                                          # integers
b = 1.                                         # float
text = ''ceci n'est pas un text''              # string
triple = (1,2,'hey!')                          # tuple

list1 = [1,2.,'hallo']                         # list
set1 = set([1, 2, 3, 3, 'A','A','a'])          #set
dict1 = {'Python': 'fast', 'Matlab': 'slow'}
                                               #dictionary
```

- indexing:

```
text = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
print text[0]                          first element
print text[:4]                         first four elements
print text[:6:2]          every second element, until sixth
print list1[-1]                        last element
```

Python basics II – scripts and functions

- writing a script:
  just put some code in a file!
- defining a function:

```
def a_useless_function(n):
      if random.random() >= 0.5:
            return n
      else:
            return math.sin(n)
```

- `run wc.py`

```
import useless as ul
ul.a_useless_function(1)
import more_useless as mul
mul.a_useless_function(1)
```

Python basics III – mutable and immutable objects

immutable objects:

- int, float, tuple, string
- `a = 3; b = 'imnutable'; c = (1,4)`
- `b[2] = 'm', c[0] = 0` # error!

mutable objects:

- list, set, dict
- `b = ['n','u','t','a','b','l','e'];`
  `c = set([1,2,3])`
- `b[0] = 'm', c.remove(1)` # valid!
- **BE CAREFUL:**
- `b = ['n','u','t','a','b','l','e']; c = b;`
  `c[0] = 'm'` $\rightarrow$ `b = ?`
- `b = ['n','u','t','a','b','l','e']; c = list(b);`
  `c[0] = 'm'` $\rightarrow$ `b = ?`
- `c = set([1,2,3]); d = c.remove(1)` $\rightarrow$ `d = ?`

Python basics IV – double assign, assert

double assign:

- `a = 3; b = 5` vs. `a, b = 3, 5`
  `a = b; b = a` vs. `a, b = b, a`
  $\rightarrow$ `b = ?, a = ?`
- splitting up tuples and lists:
  `a = (1, 2); b, c = a`
  `a = [1, 2]; b, c = a`
- "triple assign":
  `left, up, down right = dirs = ('<=','U','D','=>')`

Python basics V – list comprehensions

two ways to generate a list `L = [1,3,5,7,9]`:

- for-loop:

```
L = []
for i in range(10):
    if i%2 == 0:
        L.append(i+1)
```

- list comprehension:

```
L = [i+1 for i in range(10) if i%2 == 0]
```

Python basics VI – style

style:

- indentation: 4 whitespace instead of TAB
- every function should have a header
- variables should have descriptive names
- be consistent!
- more:
  http://www.python.org/dev/peps/pep-0008/#introduction

assertions:

- use assertions to test your code:

```
assert(     np.all( im2vec(vec2im(A) == A )     )
assert( fibo(4) == 3 )
```

Python basics VII – more

things you might like to know

- generators
- generator functions
- lambda functions
- profiling
- debugging
- ...

Python basics and IPython

Exercises I

Numpy: arrays

numpy

- `v = array([1,2,3])` # generate a vector
- `m = array([[1,2,3 ],[4,5,6]])` # generate a matrix
- `m.shape` # obtain the shape of an array
- `m = ones([5,7])` # generate a matrix of ones
- `m = zeros([3,4])` # generate a matrix of zeros
- `n = ones_like(m)` # same as ones(m.shape)
- `m = eye(3)` # identity matrix
- `m = rand(3,2)` # uniform random numbers in [0,1]
- `m = randn(3,4,6)` # normal random numbers
- `assert( m[1,2,3] == m[1][2][3])` # accessing elements
- `m[:,0:2,1]` # accessing slices

Numpy: array operations

numpy

- `a = rand(2,3)` # generate a matrix
- `b = 2 * a` # scalar multiplication
- `b = a ** 2` # **2 on each element
- `b = sin(a)` # applied on each element
- `b = 2 + a` # addition of a scalar
- `c = a + b; c = a ** b; ...` # elementwise operations
- `b2 = transpose(b)` # transpose the matrix
- `c = dot(a,b2)` # matrix multiplication
- `c = a.dot(b.transpose())` # equivalent formulation

Numpy: important functions

numpy

- `A = array([[1,2,3 ],[4,5,6]])` # generate a matrix
- `s = sum(A)` # sum of elements
- `m = mean(A)` # mean of elements
- `s_row = sum(A,0)` # sum over rows (first index=0)
- `s_row = sum(A,0,keepdims=True)` # keeps dimensionality
- `m_col = mean(A,1)` # mean over columnns (second index=1)
- `v = var(A)` # variance
- `sdev = std(A)` # standard deviation
- `maximum = amax(A)` # maximum

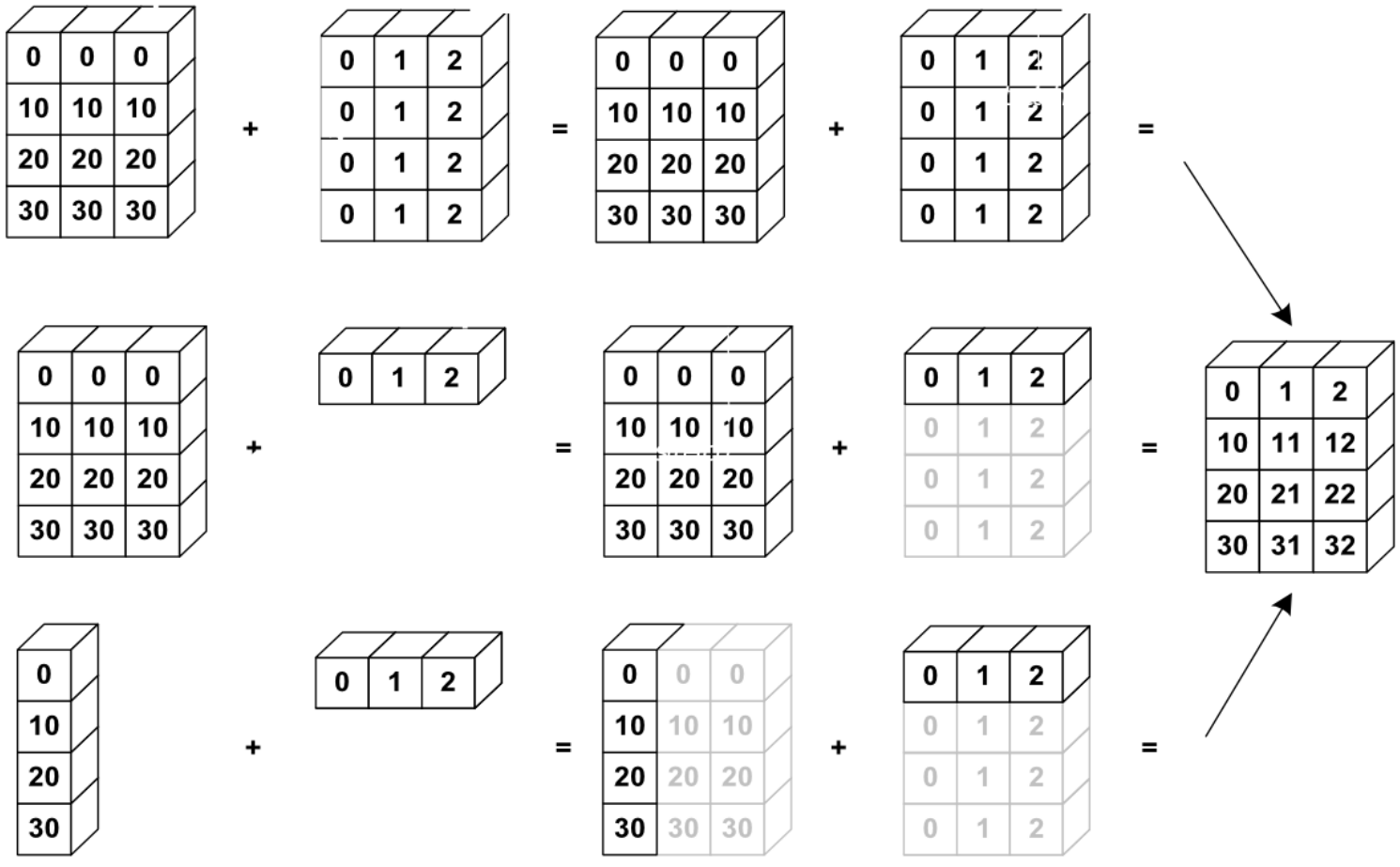
Numpy: reshaping arrays

- `a = np.arange(6)` # generate a 1D-object (row)
- `b = a[:,np.newaxis]` # make it 2D (column)
- `c = a + 10*b` # add row and column (broadcasting)

- `mat3D = rand(3,4,5)` # generate a 3D-matrix
- `mat3D = mat3D.transpose(2,0,1)` # exchange dimensions

- `mat2D = mat3D.reshape(15,4)` # reshape a matrix
- `mat4D = mat2D.reshape(5,3,2,2)` # reshape a matrix
- `mat2D = mat3D.reshape(4,-1)` # -1: infer size automatically

Numpy: array broadcasting



Numpy: arrays are mutable!

BE CAREFUL:

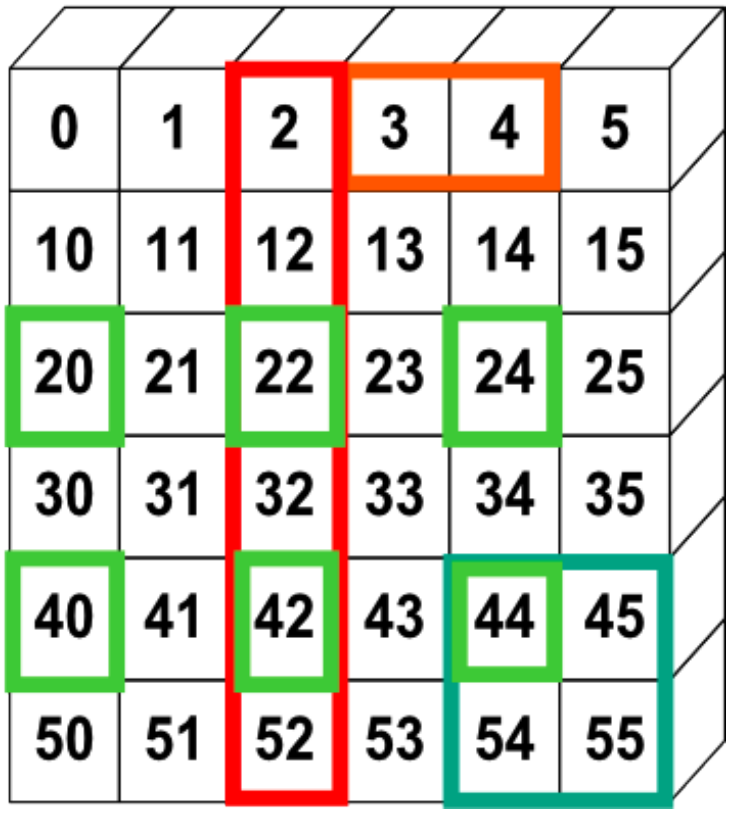
- `a = np.arange(6)` # generate a 1D-object (row)
- `b = a[:,np.newaxis]` # make it 2D (column)
- `c = a + 10*b` # add row and column (broadcasting)

- `col = c[:,0]; row = c[0,:]` # extract 1^{st} row & column
- `col[-1] = 99; row[-1] = 99` # modify row and column

- `c_trans = transpose(c)` # transpose c
- `c_trans[0,0] = 99` # modify the transpose

- `c_trans = transpose(c).copy()` # transpose c and copy
- `c_trans[-1,-1] = 99` # modify the transpose

Numpy: array indexing solution



http://scipy-lectures.github.com

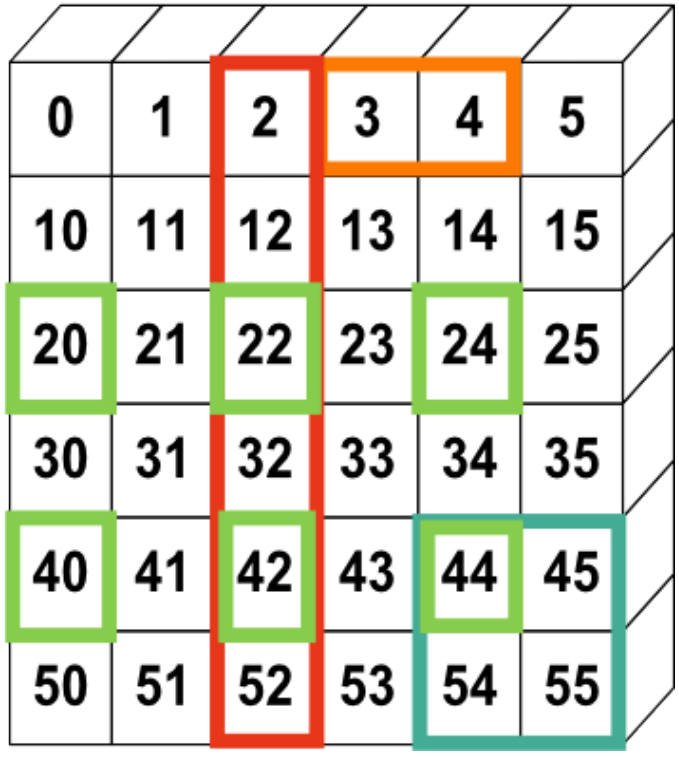
Numpy: array indexing solution

```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



http://scipy-lectures.github.com

Numpy basics

Exercises II