Switching to python

Roland Memisevic

University of Toronto

Jan 16, 2007

Running Pylab

Why?

- Python is (i) free, (ii) a 'real' programming language, and (iii) gaining popularity.

What we need:

- Pylab = Python + Numpy + Matplotlib + Ipython (+ Scipy)
- (All there on the cluster machines.)

To run Pylab interactively:

- Type:

  ipython -pylab

Exploring Pylab

- Most Matlab commands directly available:
- randn, zeros, eye, exp, cos, svd, plot, scatter, load, help (!), ...
- An example:

```
d = randn(10,1)
plot(d)
```

- Another example:

```
x = arange(-5.0, 5.0, 0.001) #this is Pylab's equivalent of (-5 : 0.001 : 5);
y = cos(x)
plot(x,y)
```

- Most operations (indexing, slicing) are the same, *but*...

Some differences

- Indexing works with *square* brackets.
- Indexes start at 0.
- No distinction between matrices and higher dimensional 'tensors':
  - '*' is *elementwise* multiplication.
  - Matrix multiplication is a *function*:

```
A = randn(5,5)
B = randn(5,1)
dot(A,B)          #Pylab's equivalent of Matlab A*B
```

- 'Everything is an object'.
  - Properties of many objects are given as *attributes*:

```
a = randn(5,2)
a.shape           #size(a) in Matlab
a.T.shape         #transpose
a.mean(0)          #mean(a,1) in Matlab
```

More python-specific things

- A useful built-in data-structure is the '*list*':
  mylist = [1, 2, 'hello', 3]
- Functions accept 'keyword-arguments'. For example:
  plot(d, linewidth = 5)
- Code in an external file is called *module* and can be *import*ed.
- A few quirks exist and can be confusing. For example, multiple definitions of 'zeros'; some commands slightly different from Matlab version; matrix-class available that redefines '*'; ...

Functions, control structures

- Defining a function:

```
def timesfour(x):
    return 4*x
```

- Control structures:
  - if-then-else:

```
if s == "y":
    print('a')
else:
    print('b')
```

  - while:

```
a = 1.0
while a != 10.0 and s == "hello":
    a = a + 1.0
```

  - for-loops:

```
for i in [1,2,'x',3,4,'h',5]:
    print(i)
```

Broadcasting and *newaxis*

- Adding a 2×5 matrix to a 1×5 vector?
- In Matlab, we use repmat. In Pylab we could do this, too.
- But Pylab offers also a another, more convenient, solution:
- Numpy always tries to copy each axis in each array to make the sizes match.
- Example:

```
( randn(2,5) + randn(1,5) ).shape      # result is (2,5)
```

- How about a 2×5 matrix to a $1 \times 5 \times 3$ tensor?
- We first have to make the number of dimensions match to make this work.
- Solution: '*newaxis*'. Examples:

```
randn(2,5).shape          # result is (2,5)
randn(2,5) + randn(1,5,3)      #does not work!
randn(2,5)[:,:,newaxis].shape          # result is (2,5,1)
randn(2,5)[:,:,newaxis] + randn(1,5,3)       #works
```

Some examples

- Cascading operations.
- Writing a derivative function.
- PCA on the Iris data-set.

More useful Python concepts

- Functions are *call-by-reference*.
- Tuples.
- Packing and un-packing.
- Dictionaries.
- Classes.
- Iterators, generators.
- List comprehensions.

Links

Learning Python:

- *docs.python.org/tut/*
- *www.diveintopython.org/*

Getting the packages:

- For Python: *python.org*
- For Numpy and Scipy: *scipy.org*
- For Matplotlib: *matplotlib.sourceforge.net*

Matlab–Python cross-references (very useful):

- *mathesaurus.sourceforge.net/matlab-numpy.html*
- *scipy.org/NumPy_for_Matlab_Users*