Week 7 - Lecture 13

Istvan Albert
Huck Institutes for the Life Sciences
Setup check for the Mac

```python
# if this runs your setup is fine
#
# otherwise you need to troubleshoot
#
import pylab
```
Plotting – Troubleshooting for Mac

• There may be two different python versions installed on a Mac one has plotting the other does not. *(default python and Enthought python)*

• Komodo ➔ Preferences ➔ Environment ➔ PATH

• **Double click** PATH ➔ Click dropdown arrow ➔ Click green + icon

• Navigate to this path then click OK:

  /Library/Frameworks/Python.framework/Versions/4.3.0/bin

• Press **Up Arrow** icon to make this path first entry

• **See next page for screenshots**
Mac only! - Komodo Preferences

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOME</td>
<td>/Users/ialbert</td>
</tr>
<tr>
<td>LOGNAME</td>
<td>ialbert</td>
</tr>
<tr>
<td>PATH</td>
<td>/usr/bin:/bin:/usr/sbin:/sbin</td>
</tr>
<tr>
<td>SECURITYSESSIONID</td>
<td>b99af0</td>
</tr>
<tr>
<td>SHELL</td>
<td>/bin/bash</td>
</tr>
</tbody>
</table>

User Environment Variables (override defaults)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
</table>
| PATH | /Library/Frameworks/Python.framework/Versions/Current/bin/
Mac only! – Update path

Navigate to ➔ Macintosh HD ➔

/Library/Frameworks/Python.framework/Versions/4.3.0/bin

Test with ➔ write: **import pylab**
Generating random numbers

• Allows us to create baseline values

• We can shuffle datasets, split them into random groups

• Compare against random samples
Commonly used functions

```python
import random

# random number between 0 and 1
print random.random()

# random integer between the limits
# including the two limits!
print random.randint(1, 100)

# random number from a normal (gaussian)
# distribution of mean=1,
# and standard deviation=10)
print random.gauss(mu=1, sigma=10)
```

Command Output

```
0.68521959858
21
9.13325179721
```
Generating random numbers

```python
import random

# seed allows you to generate the
# same sequence of random numbers repeatedly
random.seed(39929)

def generate(x):
    return random.random()

data = range(10)
values = map(generate, data)

print values[:3]
```

Output:

```
[0.59585819492243042, 0.93760022126109865, 0.036526830134548383]
```
Pigeonhole principle

Occurs when distributing a larger set over a smaller one

10 pigeons over 9 boxes $\rightarrow$ some box will end up with more than one pigeon

Other “pigeonhole” problems are a lot less obvious $\rightarrow$ for example randomizing entries
Proposed randomization of a list

Take a list of numbers (of size N)

1. Go over each element and swap it with any other element

   Total possible end configurations: N!
   Total visited configurations: N^N

   \[ N^N > N! \]

   it is a pigeonhole problem

   We can never evenly distribute N^N over N! possible outcomes

   The incorrect algorithm described above is widely circulated.

Paper: How We Learned to Cheat at Online Poker: A Study in Software Security
Python does it correctly

```python
import random

data = range(10)

print data

# in place shuffling (randomization)
# does not return anything
random.shuffle(data)

print data
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[6, 4, 3, 7, 5, 2, 0, 1, 9, 8]
```
Unique random elements

```python
import random

# create the data
data = range(10)

# randomize
random.shuffle(data)

# picking the first 5 of a randomized order
# is the same as picking 5 at random
# from the original data
print data[:5]
```

There are more efficient solutions for long lists
Plotting! Yay!

- If you haven’t **seen** your data, you don’t know your data

- **matplotlib** ➔ matlab “clone” in python

- Search for **matplotlib** then look at the **gallery of plots** to see what is possible

**Note:** he gallery is not sorted by difficulty level, many plots explore advanced functionality
Creating a plot

```python
import pylab

def square(x):
    return x*x

data = range(-10, 10)
values = map(square, data)

pylab.plot(values)

# this keeps the plot open
pylab.show()
```
Adding more details

```python
import pylab

def square(value):
    return value*value

x = range(-10, 10)
y = map(square, x)

pylab.title('Squared values plot')
pylab.xlabel('Index')
pylab.ylabel('Values')
pylab.plot(y)
pylab.show()
```
Program output

Squared values plot

Values

Index
pylab.plot() - quite powerful

- It takes dozens of parameters (all have default values)

- Output is formatted via declarations like this:

  `pylab.plot(values, "bo-", lw=2)`
```python
def square(value):
    return value * value

def equation(value):
    return value ** 2 + 5 * value - 6

x = range(-10, 10)
y1 = map(square, x)
y2 = map(equation, x)

pylab.plot(x, x, 'r*-')
pylab.plot(x, y1, 'bo')
pylab.plot(x, y2, 'go--')
pylab.show()
```
Color specification

The following color abbreviations are supported:

<table>
<thead>
<tr>
<th>character</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘b’</td>
<td>blue</td>
</tr>
<tr>
<td>‘g’</td>
<td>green</td>
</tr>
<tr>
<td>‘r’</td>
<td>red</td>
</tr>
<tr>
<td>‘c’</td>
<td>cyan</td>
</tr>
<tr>
<td>‘m’</td>
<td>magenta</td>
</tr>
<tr>
<td>‘y’</td>
<td>yellow</td>
</tr>
<tr>
<td>‘k’</td>
<td>black</td>
</tr>
<tr>
<td>‘w’</td>
<td>white</td>
</tr>
</tbody>
</table>

In addition, you can specify colors in many weird and wonderful ways, including full names (‘green’), hex strings (‘#008000’), RGBA tuples ((0,1,0,1)) or grayscale intensities as a string (‘0.8’). Of these, the string specifications can be used in place of a `fmt` group, but the tuple forms can be used only as `kwargs`.
## Symbol and line specification

<table>
<thead>
<tr>
<th>character</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>' - '</td>
<td>solid line style</td>
</tr>
<tr>
<td>' -- '</td>
<td>dashed line style</td>
</tr>
<tr>
<td>' --- '</td>
<td>dash-dot line style</td>
</tr>
<tr>
<td>' : '</td>
<td>dotted line style</td>
</tr>
<tr>
<td>' , '</td>
<td>point marker</td>
</tr>
<tr>
<td>' ; '</td>
<td>pixel marker</td>
</tr>
<tr>
<td>' o '</td>
<td>circle marker</td>
</tr>
<tr>
<td>' V '</td>
<td>triangle_down marker</td>
</tr>
<tr>
<td>' ^ '</td>
<td>triangle_up marker</td>
</tr>
<tr>
<td>' &lt; '</td>
<td>triangle_left marker</td>
</tr>
<tr>
<td>' &gt; '</td>
<td>triangle_right marker</td>
</tr>
<tr>
<td>' 1 '</td>
<td>tri_down marker</td>
</tr>
<tr>
<td>' 2 '</td>
<td>tri_up marker</td>
</tr>
<tr>
<td>' 3 '</td>
<td>tri_left marker</td>
</tr>
<tr>
<td>' 4 '</td>
<td>tri_right marker</td>
</tr>
<tr>
<td>' 5 '</td>
<td>square marker</td>
</tr>
<tr>
<td>' p '</td>
<td>pentagon marker</td>
</tr>
<tr>
<td>' # '</td>
<td>star marker</td>
</tr>
<tr>
<td>' h '</td>
<td>hexagon1 marker</td>
</tr>
<tr>
<td>' H '</td>
<td>hexagon2 marker</td>
</tr>
<tr>
<td>' + '</td>
<td>plus marker</td>
</tr>
<tr>
<td>' x '</td>
<td>x marker</td>
</tr>
<tr>
<td>' D '</td>
<td>diamond marker</td>
</tr>
<tr>
<td>' d '</td>
<td>thin_diamond marker</td>
</tr>
<tr>
<td>' l '</td>
<td>vline marker</td>
</tr>
<tr>
<td>' _ '</td>
<td>hline marker</td>
</tr>
</tbody>
</table>

**linestyle** or *ls*

[`'-'` | `'.'` | `'-.'` | `':.'` | `'None'` | `' '` | `'` ] and any drawstyle in combination with a linestyle, e.g. ‘steps-’.
Histograms

```python
import pylab, random

N = 100
mean, sigma = 10, 2

def gauss(x):
    return random.gauss(mu=mean, sigma=sigma)

values = map(gauss, range(N))

pylab.hist(values)

#pylab.hist(values, bins=20, fc='r')
pylab.show()
```
Histogram output
Generate the histogram shown on the next page. The data → random gaussian distribution. Mean=100 Standard deviation \( \sigma = 10 \).

Vertical lines are at mean-\( \sigma \), mean and at mean+\( \sigma \) positions.

How many datapoints did you need to get a smooth distribution (approximation)?

What percent of your generated values are no more than one standard deviation (\( \sigma \)) away from the mean (\( |\text{mean}-x|<\sigma \))?
Homework: desired output
Week 7 - Lecture 14

Istvan Albert
Huck Institutes for the Life Sciences
About Flowcharts (from xkcd)

A Guide to Understanding Flow Charts
Presented in Flow Chart Form

START

Do you understand flow charts?

Yes

Good

No

Okay, you see the line labeled "yes"?

Yes

Yes

...and you can see the ones labeled "no"?

Yes

Yes

Let's go drink.

6 drinks

Hey, I should try installing FreeBSD!

No

Screw it.

But you just followed them twice!

No

But you see the ones labeled "no"?

Yes

Wait, what?

Yes

I hate you.

No

Listen.
We say no to flowcharts

• One of the mistakes of early programming practice

• Still widely practiced in some course work

• They are an illusion ➔ any non-trivial problem degrades into an complicated set of lines that is actually a lot more difficult to understand
# Tuple unpacking

```python
# unpacks right side into left side
# the number of elements must match on each side
a, b, c = ('Joe', 12, [1, 2, 3])

print(a)
print(b)
print(c)
```

```
Joe
12
[1, 2, 3]
```
Tuple references

```python
row = ( 'Joe', 12, [1, 2, 3] )

print row[0], row[-1]

a, b, c = row

print a, c
```

```
Joe [1, 2, 3]
Joe [1, 2, 3]
```
Parallel iteration

• Accessing multiple rows of different lists at the same time

• For example:

  we have foreground, background → subtract background from foreground

• We’ll look at two methods: one that assumes both lists of the same lengths, the other truncates to shortest
Parallel iteration: map

# two parameter function
def add(x, y):
    return x + y

data1 = [1, 2, 3]
data2 = [10, 20, 30]

# map gets two datasets
result = map(add, data1, data2)

print result

[11, 22, 33]

Two parameters for the function, two datasets passed to map
map: transforms one or more datasets

```python
help(map)
```

Help on built-in function map in module __builtins__:

```python
map(...)
map(function, sequence[, sequence, ...]) -> list

Return a list of the results of applying the function to the items of
the argument sequence(s). If more than one sequence is given, the
function is called with an argument list consisting of the corresponding
item of each sequence, substituting None for missing values when not all
sequences have the same length. If the function is None, return a list of
the items of the sequence (or a list of tuples if more than one sequence).
```
Map: uneven data sizes

```python
def merge(x, y):
    return x, y

data1 = [1, 2, 3]
data2 = [10, 20, 30, 40]

# map gets two datasets
result = map(merge, data1, data2)

print result
```

```
[(1, 10), (2, 20), (3, 30), (None, 40)]
```
Parallel iteration: zip

```python
a = [1, 2, 3]
b = [10, 20, 30]
c = zip(a, b)
print(c)
```

```
[(1, 10), (2, 20), (3, 30)]
```
Parallel iteration: zip (uneven lengths)

```
a = [ 1, 2, 3]
b = [10, 20, 30, 40]
c = zip(a,b)
print c
```

Result stays the same! Truncates to shortest data
Starting point – this should work

```python
# renamed the week4 module to this name
import bmmmb

rows = bmmmb.read_tabular('short-expression.txt')

# background and foreground for channel 1
bg = bmmmb.get_column(rows, 'CH1B_MEAN')
fg = bmmmb.get_column(rows, 'CH1I_MEAN')

# averages
print bmmmb.avg(bg)
print bmmmb.avg(fg)
```

59.3058823529
356.105882353
Computing the actual signal

```python
# renamed the week4 module to this name
import bmmb
rows = bmmb.read_tabular('short-expression.txt')

# background and foreground for channel 1
bg = bmmb.get_column(rows, 'CH1B_MEAN')
fg = bmmb.get_column(rows, 'CH1I_MEAN')

# subtract the background from the foreground

def subtract(fval, bval):
    return fval - bval

# actual signal
values = map(subtract, fg, bg)
print values[:5]
```

```
[133.0, 191.0, 201.0, 146.0, 73.0]
```
Back to plotting
matplotlib is incredibly powerful

Try the pylab tutorial for look up various details on plotting:

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

Every customization you can think of is probably possible.

Does not mean everything will be necessarily easy to accomplish.
```python
import pylab

def s1(x):
    return 2*x*x + 3*x + 10

def s2(x):
    return 4*x*x + 5*x + 6

x = range(-10, 10)
y1 = map(s1, x)
y2 = map(s2, x)

# marker size=ms, linewidth=lw
pylab.plot(x, y1, 'bo-', lw=2, ms=10)
pylab.plot(x, y2, 'r^-', lw=2, ms=10)
pylab.show()
```
Which line is what equation?
Adding a legend

```
# marker size=ms, linewidth=lw

p1 = pylab.plot(x,y1, 'bo-', lw=2, ms=10)
p2 = pylab.plot(x,y2, 'r^-', lw=2, ms=10)

text1 = "2*x**2 + 3*x + 10"
text2 = "4*x**2 + 5*x + 6"

pylab.legend([p1, p2], [text1, text2])
pylab.show()
```

Save the return values of the plots, assign them text labels
Output with a legend – could be better
p1 = pylab.plot(x, y1, 'bo-', lw=2, ms=10)
p2 = pylab.plot(x, y2, 'r^-', lw=2, ms=10)

# latex formatting parameters
text1 = "$2x^2+3x+10$"
text2 = "$4x^2+5x+6$"

pylab.legend([p1, p2], [text1, text2], loc="upper center")
pylab.show()
Publication quality output

Figure 1

Equations:
1. $2x^2 + 3x + 10$
2. $4x^2 + 5x + 6$
Read the documentation

Try the pylab tutorial for look up various details on plotting:

• [http://matplotlib.sourceforge.net/contents.html](http://matplotlib.sourceforge.net/contents.html)

• There are always slight oddities: for example histograms and legends don’t play well together

• Everything can be done, but some operations can be more complicated
Multi-plots

• Last plotting related concept that we need

• Arrange multiple plots on one “canvas”

• The `subplot()` command uses: rows, cols, figure number

**Convenience:** use 2, 1, 1 or 211 as subplot parameters to specify the rows, columns and figure
Multiplots: rows, cols, fignum

```python
pylab.subplot(211)
p1 = pylab.plot(x, y1, 'bo-', lw=2, ms=10)
pylab.subplot(212)
p2 = pylab.plot(x, y2, 'r^-', lw=2, ms=10)
```
```python
pylab.subplot(2, 1, 1)
p1 = pylab.plot(x, y1, 'bo-', lw=2, ms=10)

# you may split each row into multiple columns
pylab.subplot(2, 3, 4)
p2 = pylab.plot(x, y2, 'r^-', lw=2, ms=10)

pylab.subplot(2, 3, 5)
p3 = pylab.plot(x, y2, 'r^-', lw=2, ms=10)

pylab.subplot(2, 3, 6)
p3 = pylab.plot(x, y2, 'r^-', lw=2, ms=10)
```
Homework 2/2

• How close can you get to making the plot shown on the next page

• You may start out with a random dataset, then later add the realistic datasets when the plots themselves show up correctly. The plot uses the short-expression.txt dataset.

• Does not have to be identical, but should have the information that is presented. You could make them nicer as well.

• `pylab.savefig('filename.png')` saves the figure into a png picture. Call it before the `show()`