Scientific Computing with Python
Quick Reference

Starting up

To start `ipython` with `pylab`:

```
$ ipython --pylab
```

To exit: `^D` (Ctrl-d)

To break from loops: `^C` (Ctrl-c)

Simple plotting

Creating a linear array:

```
x = linspace(0, 2*pi, 50)
```

Plotting two variables:

```
plot(x, sin(x))
```

Plotting two lists of equal length x, y:

```
plot(x, y)
```

Plots with colors:

`plot(x, sin(x), 'b')` gives a blue line

Line style and markers:

`plot(x, sin(x), '--')` gives a dashed line

`'.'` – a point marker, `'o'` – a circle marker

Labels:

`xlabel('x')` and `ylabel('sin(x)')`

Title (`pylab` accepts TEX in any text expression):

```
title(r'$\sigma$ vs. $\sin(\sigma)$')
```

Legend:

```
legend(['sin(x)'], loc=center)
legend(['sin(x)', 'cos(x)'])
```

If `loc` is not specified, `best` is used

Annotate:

```
annotate('annotation string', xy=(1.5, 1))
```

Saving figures:

```
savefig('sinusoids.png')
```

Axes lengths:

Get the axes lengths:

```
xmin, xmax = xlim()
ymin, ymax = ylim()
```

Set the axes lengths:

```
xlim(-2*pi, 2*pi)
ylim(-1.2, 1.2)
```

Miscellaneous:

`clf()` to clear the plot area

`close()` to close the figure

IPython tips

`TAB` completes commands

History: Up, down arrows or (`Ctrl-p`/`Ctrl-n`)

Search: `Ctrl-r` and start typing

`Ctrl-a`: go to start of line

`Ctrl-e`: go to end of line

`Ctrl-k`: kill to end of line

Help: `object?`

Source Code: `object??`

Time execution of expression/statement: `%timeit`

Saving and Running scripts

`%hist` returns history of commands used.

To save a set of lines, say 14-18, 20, 22, to `sample.py`

```
%save sample.py 14-18 20 22
```

To run `sample.py`

```
%run -i sample.py
```

Reading from files

`filename.txt` is a file with float data. Using loadtxt:

`X = loadtxt('filename.txt')`

X is an array with all the data from `filename.txt`

`X,Y = loadtxt('filename.txt', unpack=True)`

X,Y contain each column of the data

`X = loadtxt('filename.txt', delimiter=';')`

when ';' delmits the columns of data

Statistical operations

`mean`, `median`, `std`

NumPy Arrays

Fixed size: `arr.size`

Homogeneous: `arr.dtype`

Extent along each dimension: `arr.shape`

Bytes per element: `arr.itemsize`

Array Creation

`C = array([[11,12,13], [21,22,23], [31,32,33]])`

`C.shape` shape— rows & cols

`C.dtype` data type

`B = ones_like(C)` array of ones; same shape, dtype as C
similarly `zeros_like, empty_like`

`A = ones((3,2))` array of ones of shape (3,2)
similarly `zeros, empty`

`I = identity(3)` identity matrix of size 3x3

`x, y = mgrid[0:3, 0:5]` mesh-grid of size 3x5

`x, y = ogrid[0:3, 0:5]` open mesh-grid of size 3x5

Accessing & Changing elements

`C[1, 2]` gets third element of second row
Note: Indexing starts from 0.

`C[1]` gets the second row

`C[1,:]` same as above (':' implies all columns)

`C[:,1]` gets the second column (':' implies all rows)

`C[0:2,:]` or `C[:2,:]` gets $1^{st}, 2^{nd}$ rows; all cols

`C[1:3,:]` or `C[1:,:]` gets $2^{nd}, 3^{rd}$ rows; all cols

`C[0:3:2,:]` or `C[::2,:]` gets $1^{st}, 3^{rd}$ rows; all cols

Matrix Operations

For matrices A and B of equal shapes:

`A.T` transpose

`sum(A)` sum of all elements

`A+B` addition

`A*B` element wise product

`dot(A, B)` Matrix multiplication

`inv(A)` inverse, `det(A)` determinant

`eig(A)` eigen values and vectors

`norm(A)` norm

`svd(A)` singular value decomposition

Least Square Fit

To get the least square fit of L vs. tsq:

`A = array([L, ones_like(L)])`

`A = A.T` vandermonde matrix

`result = lstsq(A, tsq)`

`coef = result[0]` coefficients

`Tline = coef[0]*L + coef[1]`

Solving Linear Equations

`A = array([[3,2,-1], [2,-2,4], [-1, 0.5, -1]])`
coefficient array

`b = array([1, -2, 0])` constant array

`x = solve(A, b)` the required solution

Checking the solution:

`Ax = dot(A,x)` matrix multiplication of A and x

`allclose(Ax, b)` check the closeness of Ax, b

Roots of Polynomials

`coeffs = [1, 6, 13]` coefficients in descending order

`roots(coeffs)` returns complex roots of the polynomial

Roots of non-linear equations

`from scipy.optimize import fsolve`
`fsolve` is not in `pylab`
we import from `scipy.optimize`

We wish to find the roots of $f(x) = sin(x) + cos(x)^2$

```
def f(x):
    return sin(x)+cos(x)**2
```

`fsolve(f, 0)`
arguments are function name and initial estimate

ODE

To solve the ODE below:
$\frac{dy}{dt} = ky(L-y)$, L = 25000, k = 0.00003, y(0) = 250

```
def f(y, t):
    k, L = 0.00003, 25000
    return k*y*(L-y)
```

`t = linspace(0, 12, 60)` time over which to solve ODE
`y0 = 250` initial conditions
`from scipy.integrate import odeint`
`y = odeint(f, y0, t)`

FFT

`t = linspace(0, 2*pi, 500)`
`y = sin(4*pi*t)` a sinusoidal signal
`f = fft(y)`
`freq = fftfreq(500, t[1] - t[0])`
`plot(freq[:250], abs(f)[:250])`

Random Numbers

`from numpy import random`
`random.random`: Uniform deviates in $[0, 1)$
`random.normal`: Random samples – Gaussian dist.
`random.normal`: Random samples – Gaussian dist.
`x = random.random(size=1000)`
`x,y = random.normal(size=(2,1000))`

Record Arrays

```
typ = [('id', int), ('x', float)]
rec = numpy.zeros(10, dtype=typ)
rec['id'] = range(10)
rec['x'] = random.random(size=10)
```

`data = csv2rec('data.csv')` from csv to record array

Basic image processing

`from pylab import imread, imshow, colorbar`
`a = imread('lena.png')`: a is a NumPy array
`imshow(a)`
`colorbar()`

3D plotting with Mayavi's `mlab`

`from mayavi import mlab` Ready to Go!
`mlab.test_<TAB>` Test Functions
`mlab.points3d(x, y, z)` Plot points in 3D
`mlab.contour3d(x*x*0.5 + y*y + z*z*2)` 3D contours
`mlab.gcf` get current figure
`mlab.savefig` save current figure
`mlab.figure` switch figure or new figure
`mlab.axes` create axes
`mlab.outline` create outline
`mlab.title` add title
`mlab.xlabel, ylabel, zlabel` labels
`mlab.colorbar` Add colorbar
`mlab.scalarbar` Add colorbar for scalar color mapping
`mlab.vectorbar` Add colorbar for vector color mapping
`mlab.show` show figure (standalone scripts)