

Review of Yesterday

1) Functions – write a function to square inputs a and b and return their sum

```
In [5]: def add_squares(a, b):  
        return a**2 + b**2  
  
        add_squares(3,4)
```

Out[5]: 25

2) Strings – Combined strings "hello" and " world", convert to capitals and print

```
In [6]: s = "hello" + " " + "world"  
        print(s.upper())  
  
        HELLO WORLD
```

3) Files – Open a plain text file (created with e.g. notepad) and print in Python

```
In [7]: #Code to create file  
import os  
os.system("echo 'some text in file' > ./test.txt")  
#code to create file  
  
with open('./test.txt') as f:  
    filestr = f.read()  
  
print(filestr)  
  
some text in file
```

4) Lists – Create a list with 1,2 and 3, add an extra entry 4 and iterate through the list and print the contents

```
In [8]: l = [1,2,3]  
        l = l + [4]  
        for i in l:  
            print(i)  
  
        1  
        2  
        3  
        4
```

5) Dictionary – Create a shape_sides dictionary d with keys "triangle", "square" and "pentagon" and values 3, 4 and 5 respectively. Iterate and print all items

```
In [9]: d = {"triangle":3, "square":4, "pentagon":5}
        for key, value in d.items():
            print(key, value)

('pentagon', 5)
('square', 4)
('triangle', 3)
```

6) Numpy arrays – Import the numpy module, create an numpy arrays of values from 1 to 5 and add one to each entry.

```
In [10]: import numpy as np
        x = np.array(range(1,6))
        x = x +1
        print(x)

[2 3 4 5 6]
```

7) Create a module containing a function which adds two numbers a and b, returning thier sum. import into a script and print output

```
In [11]: %%writefile add_module.py
        # ^ NOT PYTHON ^
        #%%writefile is jupyter notebook cell magic
        #to create a file Numbers.py

        def add(a, b):
            return a + b

Writing add_module.py
```

```
In [12]: from add_module import add
        add(4.,5.)
```

Out[12]: 9.0

8) Classes – Create a class called number which takes an input x in its constructor and stores it (self.x = x). Add a method to square the (self.x) value and return

```
In [13]: class number():
        def __init__(self, x):
            self.x = x
        def square(self):
            return self.x**2

n = number(5)
another = number(26)
print(n.square(), another.square())

(25, 676)
```

Introduction to Numpy and Plotting

1) Create a numpy array `x=np.array([1,4,7])`. Get the mean and standard deviation. Add one to each value in the array and get the new mean and standard deviation.

```
In [14]: import numpy as np
x = np.array([1,4,7])
print(x, x.mean(), x.std())
x = x + 1
print(x, x.mean(), x.std())

(array([1, 4, 7]), 4.0, 2.4494897427831779)
(array([2, 5, 8]), 5.0, 2.4494897427831779)
```

2) Create array `x = np.array([[1,2,3],[4,5,6],[7,8,9]])`, use array slicing to get array `([1, 2, 3])` and array `([2, 5, 8])` and add them together.

```
In [15]: x = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(x)
print(x[0,:],x[:,1],x[0,:]+x[:,1])

[[1 2 3]
 [4 5 6]
 [7 8 9]]
(array([1, 2, 3]), array([2, 5, 8]), array([ 3,  7, 11]))
```

3) Setup a 3 by 3 identity matrix "I" (ones on the diagonal, zeros off diagonal). Create a 3 by 3 array of random numbers r. Check `np.dot(I,r)` is as expected

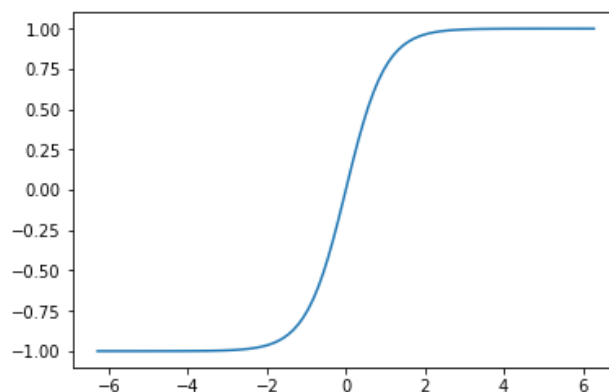
```
In [16]: import numpy as np
I = np.array([[1,0,0],[0,1,0],[0,0,1]])
r = np.random.random([3,3])
# Check dot product of r with Identity matrix
# is the same as r
print(r == np.dot(I,r))

[[ True  True  True]
 [ True  True  True]
 [ True  True  True]]
```

4) Plot a tanh function in the range -2 pi to 2 pi using linspace and matplotlib plot.

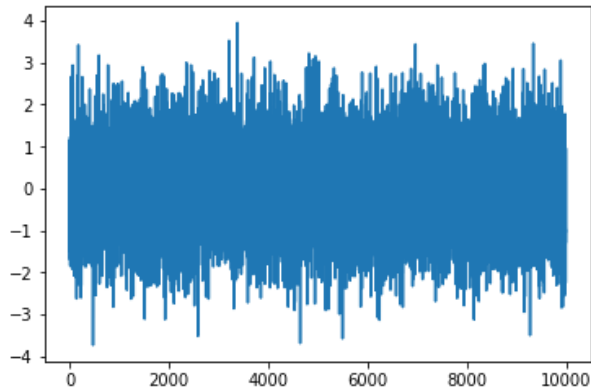
```
In [17]: import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-2.*np.pi, 2*np.pi, 100)
plt.plot(x, np.tanh(x))
plt.show()
```



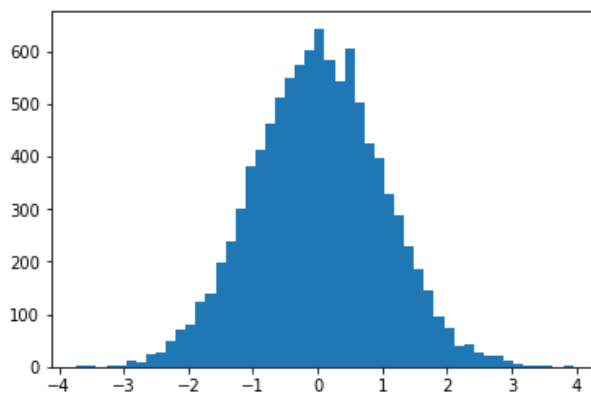
5) Create a 1D array of 10,000 normally distributed random numbers t . Plot as a time history and zoom in to see the detail.

```
In [18]: #Here time history was confusing, should mean just plot  
#values. To zoom, you will need to plot in python or  
#ipython which will open up a panel  
# Also, the command  
##matplotlib notebook  
# should allow interactive plots  
t = np.random.randn(10000)  
plt.plot(t)  
plt.show()
```



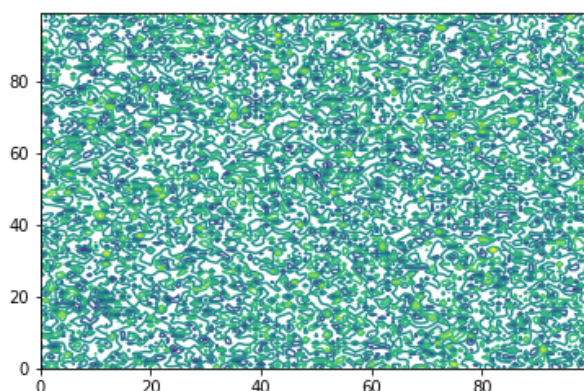
6) Plot a histogram of the array t from question 4) with 50 bins.

```
In [19]: plt.hist(t, 50)  
plt.show()
```



7) Convert array t to a 2D array using `field=t.reshape(100,100)` and plot using contour.

```
In [20]: plt.contour(t.reshape(100,100))  
plt.show()
```

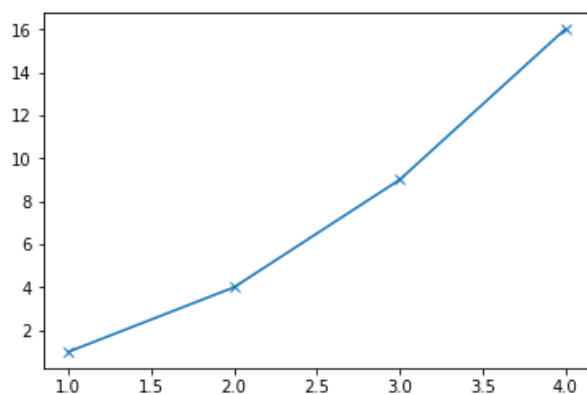


Loading data from Files and Plotting

1) Setup a 2D matrix $z = \text{np.array}([1,2,3,4],[1,4,9,16])$. Use array slicing (or a loop) to get two arrays $a=[1,2,3,4]$ and $b=[1,4,9,16]$ and plot them against each other.

```
In [21]: z = np.array([[1,2,3,4],[1,4,9,16]])  
print(z)  
a = z[0,:]  
b = z[1,:]  
plt.plot(a, b, '-x')  
plt.show()
```

```
[[ 1  2  3  4]  
 [ 1  4  9 16]]
```

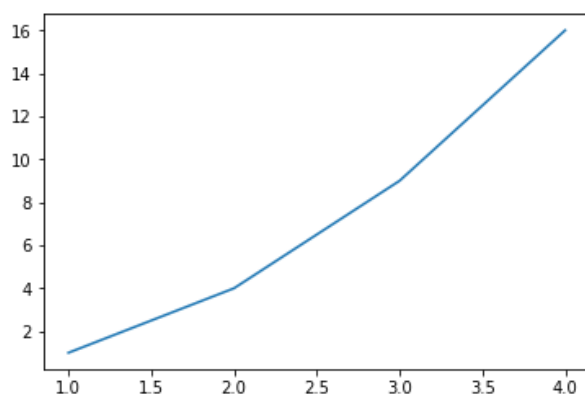


2) Create a csv file (using excel or a text editor) to give two columns containing the data from part 1), read into Python and plot one column against the other

```
In [22]: #Note, should be done in a text editor
#or excel and saved as csv. We use
#os.system here so notebook is self contained.
#String here has to specified as
#a raw string (r letter before) and we
#need double slash for newline character \n
#to allow text escape.
csvfile = r"1, 1 \n 2, 4 \n 3, 9 \n 4, 16 \n"
import os
cmd = r"echo " + csvfile + r" > out.csv"
os.system(cmd)

#Read and plot
import numpy as np
data = np.genfromtxt('out.csv', delimiter=',')
print(data)
plt.plot(data[:,0], data[:,1])
plt.show()
```

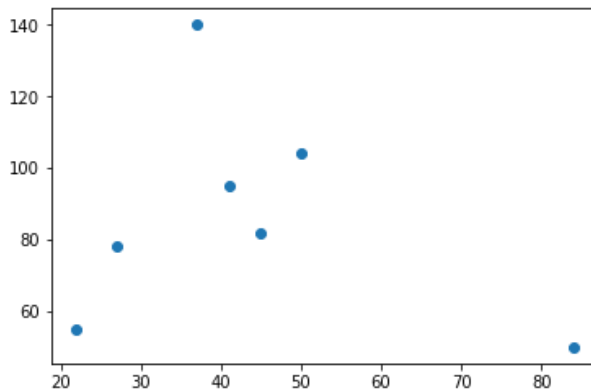
```
[[ 1.  1.]
 [ 2.  4.]
 [ 3.  9.]
 [ 4. 16.]]
```



3) Open a spreadsheet (e.g. the sample in the examples folder) using either pandas or convert to csv and use `genfromtxt` and `plot` (e.g. age against weight).

```
In [23]: #You may need to install this and update xlrd
#which is easiest using e.g. pip install --upgrade xlrd
import pandas
import matplotlib.pyplot as plt
data = pandas.read_excel("./sample_spreadsheet.xlsx")
print(data)
plt.plot(data['Age'], data['Weight'], 'o')
plt.show()
```

	Name	Age	Weight
0	Joe Bloggs	27	78
1	John Dow	41	95
2	Jane Doe	22	55
3	Gary Jones	50	104
4	Michael Hunt	45	82
5	James Brown	37	140
6	Jessica Green	84	50



4) Use Pickle to dump list [4,6,7] and string "hello". Load in a new script/session

```
In [24]: l = [4,6,7]
s = "hello"
import pickle
f = open('out.p', 'w')
pickle.dump([l, s], f)
f.close()
fnew = open('out.p', 'r')
plist = pickle.load(fnew)
print(l, s, plist)
for i in plist:
    print(i)
```

([4, 6, 7], 'hello', [[4, 6, 7], 'hello'])
[4, 6, 7]
hello

5) Write a function to read a csv file as a string and convert to numbers stored in a numpy array. What options do you need to make it more general (e.g. to skip header lines). Use on question 3) and 4) above.

```

In [33]: import numpy as np

#There are many ways to do this and making it general
#is not simple. This example is to emphasise the design and
#usefulness of numpy genfromtxt.
#Here we consider a file of the form:
#x,      y
#1.0, 1.0
#2.0, 4.0
#3.0, 9.0
#4.0, 16.0
#5.0, 25.0
#6.0, 36.0
#Note, should be created in a text editor
#or excel and saved as csv. We use
#os.system here so notebook is self contained.
#String here has to specified as
#a raw string (r letter before) and we
#need double slash for newline character \n
#to allow text escape.
csvfile = r"x, y\n1.0, 1.0\n2.0, 4.0\n3.0, 9.0\n4.0, 16.0\n5.0, 25.0\n6.0, 36.0"
import os
cmd = r"echo " + csvfile + r" > file.csv"
os.system(cmd)

def read_csv(filename):
    with open(filename) as f:
        l = []
        for line in f:
            cols = line.replace("\n", "").split(",")
            l.append(cols)
        #As we know first element is a header, we skip
        #this and convert to a numpy array of floats
        #using dtype keyword
        return np.array(l[1:], dtype="d")

#Read and print
x = read_csv("file.csv")
print(x)

#We add the keyword skip_header to
#allow the user to specify how many
#non-number rows there are and specify
#its default value as 1.
def read_csv(filename, skip_header=1):
    with open(filename) as f:
        l = []
        for line in f:
            cols = line.replace("\n", "").split(",")
            l.append(cols)
        return np.array(l[skip_header:], dtype="d")

#Read with top four lines skipped and print
x = read_csv("file.csv", skip_header=4)
print("4 skipped lines", x)

[[ 1.  1.]
 [ 2.  4.]
 [ 3.  9.]
 [ 4. 16.]
 [ 5. 25.]
 [ 6. 36.]]
('4 skipped lines', array([[ 4., 16.],
 [ 5., 25.],
 [ 6., 36.])))

```

6) Add names option to take the title on the top column and create a dictionary of arrays

```
In [34]: #Again this is not simple. Again to show how useful
#genfromtxt can be (which should be used)
#Here we consider a file of the form:
#x,      y
#1.0, 1.0
#2.0, 4.0
#3.0, 9.0
#4.0, 16.0
#5.0, 25.0
#6.0, 36.0

#We add the keyword title to take top row after
#skipped header lines which now have default
#value of zero as first line is title
def read_csv(filename, skip_header=0, title=None):
    with open(filename) as f:
        d = {}
        for line in f:
            # If we specify title=True, take first line
            # and create a dictionary with the
            # values on the first line as keys
            if title: #same as title=True
                cols = line.replace("\n","").split(",")
                for k in cols:
                    d[k] = []
                #set title to False as we have this
                title=False
                continue
            cols = line.replace("\n","").split(",")
            #There is a key per column so loop over keys
            #and use enumerate to get corresponding column
            for i, key in enumerate(d.keys()):
                d[key].append(cols[i])

        #Convert lists to numpy arrays
        for key in d.keys():
            d[key] = np.array(d[key], dtype="d")
        return d

print(read_csv("file.csv", skip_header=0, title=True))

{'x': array([ 1.,  2.,  3.,  4.,  5.,  6.]), 'y': array([ 1.,  4.,  9.,
16., 25., 36.])}
```

Using Python as glue: Filesystems, subprocess and ctype

1) Use glob to get files in the current directory, loop through list and print all files. Create a list of only the python scripts (i.e. files which end with .py).

```
In [35]: import glob
files = glob.glob(".*")
#Print all files
pylist = []
for f in files:
    print(f)
    if ".py" in f:
        pylist.append(f)

print(pylist)

./out.csv
./test.txt~
./Day2_Hands_on_sessions.ipynb
./out.p
./add_module.pyc
./file.csv~
./Solutions_for_2D_plotting_hands_on
./sample_spreadsheet.xlsx
./file.csv
./test.txt
./add_module.py
['./add_module.pyc', './add_module.py']
```

2) Use os to create a folder and change directory to it. Use a loop to create filename0 to filename10 (see hands on 3 yesterday) each file containing the number 0 to 10 respectivly (note os.system("echo 5 > filename5") creates a 5 in filename5

```
In [36]: import os
folder= "some_folder"
try:
    os.mkdir(folder)
except OSError:
    pass #We need this for if the file exists
os.chdir(folder)
for i in range(11):
    cmd = "echo " + str(i) + " > filename" + str(i)
    print(cmd)
    os.system(cmd)

echo 0 > filename0
echo 1 > filename1
echo 2 > filename2
echo 3 > filename3
echo 4 > filename4
echo 5 > filename5
echo 6 > filename6
echo 7 > filename7
echo 8 > filename8
echo 9 > filename9
echo 10 > filename10
```

3) Read the contents of files filename0, ..., filename10 either using Python open or subprocess ("cat filename0" in linux/mac, "type filename10" in windows)

```
In [37]: files = glob.glob("filename*")
         files.sort()
         for file in files:
             f = open(file)
             print(f.read())
```

0

1

10

2

3

4

5

6

7

8

9

4) Use subprocess instead of os.system in 2), read using 3) and check

```
In [38]: import subprocess as sp
         folder= "some_folder"
         os.mkdir(folder)
         os.chdir(folder)
         for i in range(11):
             cmd = "echo " + str(i) + " > filename" + str(i)
             out = sp.Popen(cmd, shell=True)
             out.wait()

         for i in range(11):
             print(sp.check_output("cat filename" + str(i), shell=True))
```

0

1

2

3

4

5

6

7

8

9

10

Advanced

5) Compile a low level code (C, Fortran or other) and run using subprocess returning the output to Python

```
In [39]: #None of this is Python specific, just creating  
#the C and Fortran code here  
import os  
fcode = """  
program hello  
    print*, "Hello Fortran"  
end program  
"""  
  
#Use write syntax here, a 'w' keyword  
open('hello.f90','w').write(fcode)  
#Call compiler to get executable  
os.system('gfortran hello.f90 -o fhello')  
ccode = """  
#include <stdio.h>  
int main(void){  
    printf("Hello C ");  
    return 0;  
}  
"""  
  
#Use write syntax here, a 'w' keyword  
open('hello.cpp','w').write(ccode)  
#Call compiler to get executable  
os.system('gcc hello.cpp -o chello')  
  
#This is the solution  
import subprocess as sp  
print(sp.check_output("./fhello"))  
print(sp.check_output("./chello"))
```

Hello Fortran

Hello C

6) Write a simple c function which takes an float, subtracts 1.0 and returns. Write a ctypes wrapper in Python and call it. What do you notice about duck typing here?

```
In [40]: #Again c++ code should be written and compiled
#seperatly but included in this notebook for completeness
import os

ccode = """
extern "C" float subtract_one(float i)
{
    return i-1.0;
}
"""

#Use write syntax here, a 'w' keyword
open('test.cpp','w').write(ccode)
os.system('gcc -shared -o testlib.so -fPIC test.cpp')

#This is the actual code which uses ctypes
#to interface with Python
import numpy.ctypeslib as ctl
import ctypes
libname = 'testlib.so'
libdir = './'
lib=ctl.load_library(libname, libdir)
py_subtract_one = lib.subtract_one
py_subtract_one.argtypes = [ctypes.c_float]
#We also need to specify the return is a float here
#Previously this was the default of an integer
lib.subtract_one.restype = ctypes.c_float
#Try a float
print(py_subtract_one(5.5))
#try an integer
print(py_subtract_one(2))
#What about a string?
print(py_subtract_one("2"))
```

```
4.5
1.0
```

```
-----
ArgumentError                                Traceback (most recent call last)
<ipython-input-40-16f7b6418c35> in <module>()
      30 print(py_subtract_one(2))
      31 #What about a string?
--> 32 print(py_subtract_one("2"))
```

```
ArgumentError: argument 1: <type 'exceptions.TypeError'>: wrong type
```

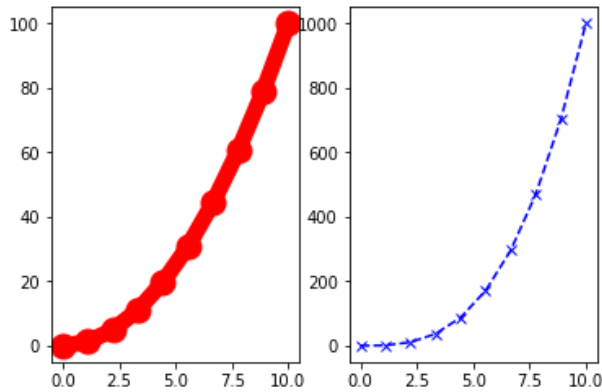
Notice that we've actually passed an int to the c code which, due to lack of C interface checking is passed in without complaint and is converted to a float by the subtraction of a float. However, a string is caught by python...

More Advanced Plotting

1) Create `x=np.linspace(0., 10.,1000)` and plot `x2` and `x3` on axes `ax[0]` and `ax[1]` from `plt.subplots(2,1)`. Change line colour, markers and size

```
In [41]: import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(0., 10.,10)
fig, ax = plt.subplots(1,2)
ax[0].plot(x, x**2, 'r-o', ms=15., lw=10)
ax[1].plot(x, x**3, 'b--x')
plt.show()
```



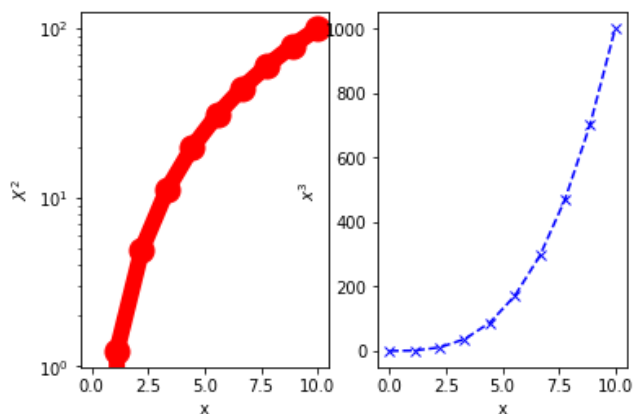
2) Change the y axes on 1) to logarithmic and label the x and y axes

```
In [42]: import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(0., 10.,10)
fig, ax = plt.subplots(1,2)
ax[0].plot(x, x**2, 'r-o', ms=15., lw=10)
ax[1].plot(x, x**3, 'b--x')
ax[0].set_yscale("log")
ax[0].set_xlabel("x")
ax[1].set_xlabel("x")

ax[0].set_ylabel("$X^2$")
ax[1].set_ylabel("$x^3$")

plt.show()
```

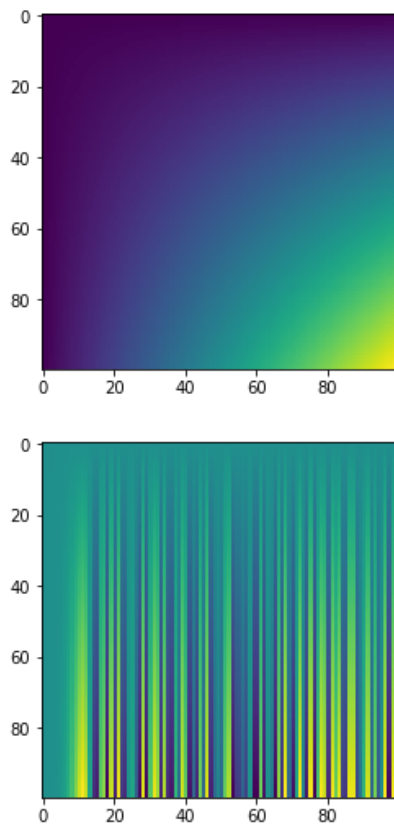


3) Create 2D data from 1D arrays y and z using $x = \text{np.outer}(y, z)$ and plot using imshow, contour or pcolormesh (try different 1D arrays)

```
In [43]: import numpy as np
import matplotlib.pyplot as plt

y = np.linspace(0,1.,100)
z = np.linspace(0,1.,100)
x=np.outer(y,z)
plt.imshow(x)
plt.show()

y = np.linspace(0,1.,100)
z = np.sin(np.linspace(0.,10,100)**4)
x=np.outer(y,z)
plt.imshow(x)
plt.show()
```



4) Fit an appropriate line to

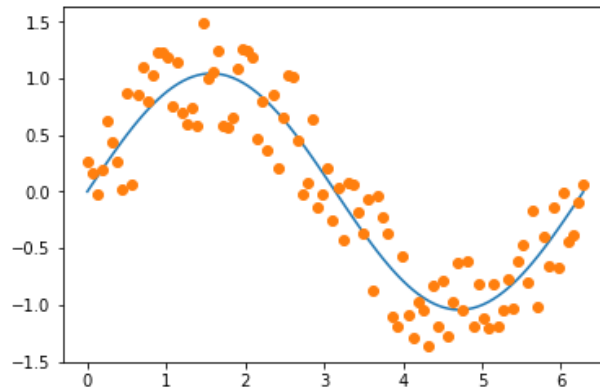
```
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x) + (2.*(np.random.random(100)-.5))
```

```
In [44]: import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x) + (np.random.random(100)-.5)

def fitfn(x, a, b):
    return a*np.sin(b*x)

params, cov = curve_fit(fitfn, x, y)
plt.plot(x, fitfn(x, params[0], params[1]))
plt.plot(x, y, 'o')
plt.show()
```



Advanced

5) Create fig, ax = plt.subplots(1,1), switch interactive mode on and plot ax.plot(np.sin(A*x)) for A in np.linspace(-5,5,100) using plt.pause(0.1) to redraw and plt.cla() to clear the axis (NOTE WON'T WORK IN NOTEBOOK, you WILL NEED TO SAVE FILES)

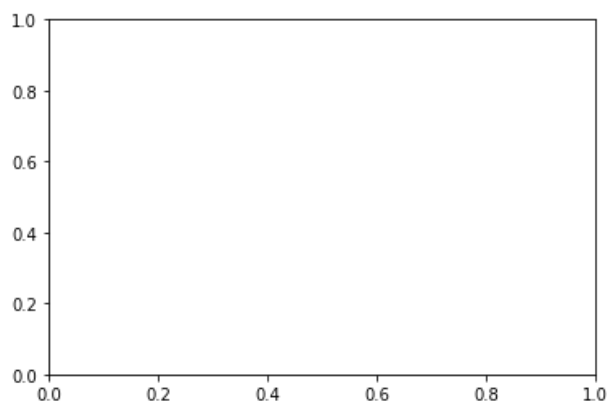
```
In [45]: import numpy as np
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1,1)
plt.ion() #interactive mode on
plt.show() #Show figure

x = np.linspace(0, 2*np.pi, 100)

for A in np.linspace(-5,5,100):
    print(A)
    ax.plot(np.sin(A*x))
    plt.pause(0.1)
    plt.cla()

plt.ioff()
```

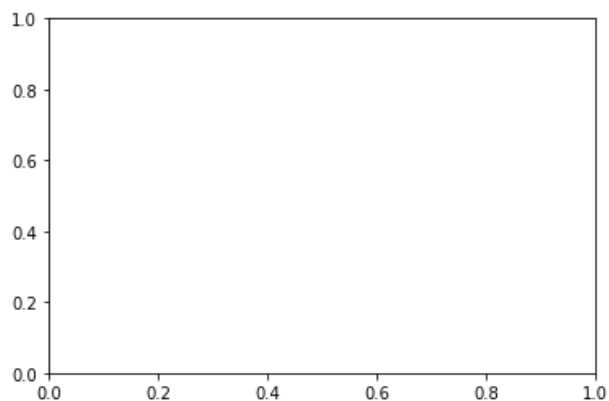


```
-5.0
-4.89898989899
-4.79797979798
-4.69696969697
-4.59595959596
-4.49494949495
-4.39393939394
-4.29292929293
-4.19191919192
-4.09090909091
-3.9898989899
-3.88888888889
-3.78787878788
-3.68686868687
-3.58585858586
-3.48484848485
-3.38383838384
-3.28282828283
```

```
-----
KeyboardInterrupt                                Traceback (most recent call last)
<ipython-input-45-028da9bc343c> in <module>()
      11     print(A)
      12     ax.plot(np.sin(A*x))
--> 13     plt.pause(0.1)
      14     plt.cla()
      15
```

```
/usr/local/lib/python2.7/dist-packages/matplotlib/pyplot.pyc in pause(interval)
1)
    303     # No on-screen figure is active, so sleep() is all we need.
    304     import time
--> 305     time.sleep(interval)
    306
    307
```

KeyboardInterrupt:



6) Run the slider example and adapt to plot $\sin(Ax^2)$ using function from number, num.square, with the value of A specified by the slider value.

```
In [46]: %%writefile Numbers.py
# ^ NOT PYTHON ^
%%writefile is jupyter notebook cell magic
#to create a file Numbers.py

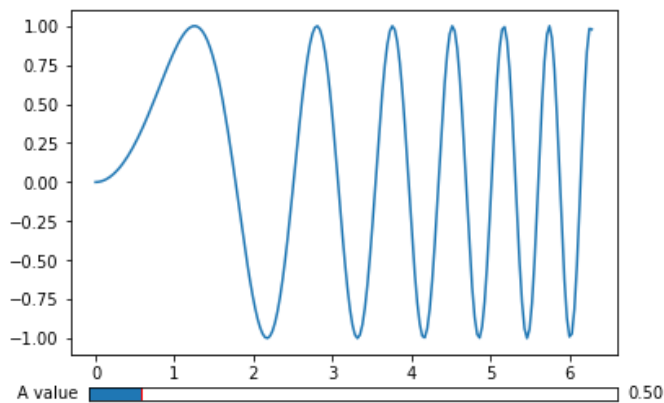
def square(a):
    return a**2

def cube(a):
    return a**3
```

Writing Numbers.py

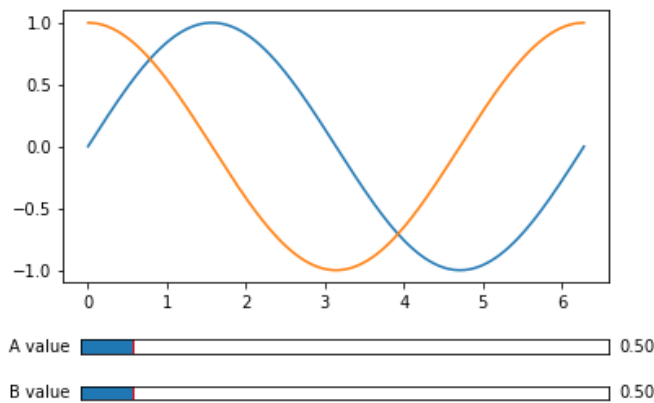
```
In [47]: import numpy as np
import matplotlib.pyplot as plt
import matplotlib.widgets as mw
import Numbers as num

#Setup initial plot of sine function
x = np.linspace(0, 2*np.pi, 200)
l, = plt.plot(x, np.sin(num.square(x)))
#Adjust figure to make room for slider
plt.subplots_adjust(bottom=0.15)
axslide = plt.axes([0.15, 0.05, 0.75, 0.03])
#Define function
def update(A):
    l.set_ydata(np.sin(A*num.square(x)))
    plt.draw()
#Bind update function to change in slider
s = mw.Slider(axslide, 'A value', 0., 5.)
s.on_changed(update)
plt.show()
```



7) Develop a slider example with both sine and cosine on the plot updated by slider. Adapt this to add a new slider for a second coefficient B for $\cos(Bx)$.

```
In [48]: import numpy as np
import matplotlib.pyplot as plt
import matplotlib.widgets as mw
import Numbers as num
#Setup initial plot of sine function
x = np.linspace(0, 2*np.pi, 200)
lsin, = plt.plot(x, np.sin(x))
lcos, = plt.plot(x, np.cos(x))
#Adjust figure to make room for slider
plt.subplots_adjust(bottom=0.3)
axslideA = plt.axes([0.15, 0.15, 0.75, 0.03])
axslideB = plt.axes([0.15, 0.05, 0.75, 0.03])
#Define function
def updateA(A):
    lsin.set_ydata(np.sin(A*x))
    plt.draw()
def updateB(B):
    lcos.set_ydata(np.cos(B*x))
    plt.draw()
#Bind update function to change in slider
sA = mw.Slider(axslideA, 'A value', 0., 5.)
sA.on_changed(updateA)
sB = mw.Slider(axslideB, 'B value', 0., 5.)
sB.on_changed(updateB)
plt.show()
```



Test Driven Development

1) Use test driven development (i.e. write the tests first) to design functions which returns the square and the cube of input values in a file called Numbers.py.

```
In [49]: import unittest

def square(a):
    pass
def cube(a):
    pass

#Note these test have been changed slightly from the
#examples to check integer is equal
#and almost equal if float (finite precision never
#exactly equal)
class test_square(unittest.TestCase):
    def test_float(self):
        self.assertAlmostEqual(square(2.), 4.)
    def test_int(self):
        self.assertEqual(square(2), 4)

class test_cube(unittest.TestCase):
    def test_float(self):
        self.assertAlmostEqual(cube(2.), 8.)
    def test_int(self):
        self.assertEqual(cube(2), 8)

#NOTE - We add argv=arg-is-ignored to avoid
#an error in jupyter notebooks
#but not needed in general
unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```

EFEF
=====
ERROR: test_float (__main__.test_cube)
-----
Traceback (most recent call last):
  File "<ipython-input-49-a106d1e12bc3>", line 21, in test_float
    self.assertAlmostEqual(cube(2.), 8.)
  File "/usr/lib/python2.7/unittest/case.py", line 552, in assertAlmostEqual
    if round(abs(second-first), places) == 0:
TypeError: unsupported operand type(s) for -: 'float' and 'NoneType'

=====
ERROR: test_float (__main__.test_square)
-----
Traceback (most recent call last):
  File "<ipython-input-49-a106d1e12bc3>", line 15, in test_float
    self.assertAlmostEqual(square(2.), 4.)
  File "/usr/lib/python2.7/unittest/case.py", line 552, in assertAlmostEqual
    if round(abs(second-first), places) == 0:
TypeError: unsupported operand type(s) for -: 'float' and 'NoneType'

=====
FAIL: test_int (__main__.test_cube)
-----
Traceback (most recent call last):
  File "<ipython-input-49-a106d1e12bc3>", line 23, in test_int
    self.assertEqual(cube(2), 8)
AssertionError: None != 8

=====
FAIL: test_int (__main__.test_square)
-----
Traceback (most recent call last):
  File "<ipython-input-49-a106d1e12bc3>", line 17, in test_int
    self.assertEqual(square(2), 4)
AssertionError: None != 4

-----
Ran 4 tests in 0.027s

FAILED (failures=2, errors=2)
Out[49]: <unittest.main.TestProgram at 0x6245510>

```

Not we want to write a function which passes these tests

```
In [50]: def square(a):
          return a**2

          def cube(a):
              return a**3

          #Note these test have been changed slightly from the
          #examples to check integer is equal
          #and almost equal if float (finite precision never
          #exactly equal)
          class test_square(unittest.TestCase):
              def test_float(self):
                  self.assertAlmostEqual(square(2.), 4.)
              def test_int(self):
                  self.assertEqual(square(2), 4)

          class test_cube(unittest.TestCase):
              def test_float(self):
                  self.assertAlmostEqual(cube(2.), 8.)
              def test_int(self):
                  self.assertEqual(cube(2), 8)

          #NOTE - We add argv=arg-is-ignored to avoid
          #an error in jupyter notebooks
          #but not needed in general
          unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
....
```

```
-----
Ran 4 tests in 0.005s
```

```
OK
```

```
Out[50]: <unittest.main.TestProgram at 0x5943410>
```

2) Refactor numbers.py to a class with constructor to take input a, stores it as self.a = a and change the functions square and cube to act on self.a.

```
In [51]: class Number():
        def __init__(self, a):
            self.a = a
        def square(self):
            return self.a**2
        def cube(self):
            return self.a**3

        #Note we also have to refactor the tests
        #this should have been done first for TDD
        class test_number(unittest.TestCase):
            def test_float(self):
                n = Number(2.)
                self.assertEqual(n.square(), 4.)
                self.assertEqual(n.cube(), 8.)
            def test_int(self):
                n = Number(2)
                self.assertEqual(n.square(), 4)
                self.assertEqual(n.cube(), 8)

        unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
.....
-----
Ran 6 tests in 0.007s
```

OK

Out[51]: <unittest.main.TestProgram at 0x6227650>

3) In numbers.py, add if **name** == "**main**": and move the tests inside this. Use import numbers as nb in a new script, instantiate n = nb.number(5.) and use the n.square() and n.cube() methods.

```
In [52]: %%writefile Number_class.py
        # ^ NOT PYTHON ^
        #the above is jupyter notebook cell magic
        #to create a file Numbers.py
        class Number():
            def __init__(self, a):
                self.a = a
            def square(self):
                return self.a**2
            def cube(self):
                return self.a**3

        #The name == main statement means this code will be run if
        # the script is called using python numbers.py but will NOT
        #be run if we import numbers
        if __name__ == "__main__":

            #Note we also have to refactor the tests
            #this should have been done first for TDD
            class test_number(unittest.TestCase):
                def test_float(self):
                    n = Number(2.)
                    self.assertEqual(n.square(), 4.)
                    self.assertEqual(n.cube(), 8.)
                def test_int(self):
                    n = Number(2)
                    self.assertEqual(n.square(), 4)
                    self.assertEqual(n.cube(), 8)

            unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

Writing Number_class.py

```
In [53]: import Number_class as nb  
  
n = nb.Number(5.)  
print(n.square(), n.cube())  
  
(25.0, 125.0)
```

Advanced

4) Refactor the three functions from the previous hands-on: `get_files`, `read_header` and `read_data` into a single class `postproc`. Write a constructor for the `postproc` class to take `foldername`, `header` and `filename` and get `self.header` and `self.files`. 5) Try steps 4) to 5) for a different input data format by creating a new class. Make `postproc` a base class and using Python inheritance syntax, e.g. `class postproc_binary(postproc):`, create a range of different data readers

Solution

A full set of scripts and a copy of the data should have been included with these solutions, demonstrating this refactoring as well as plotting and using a slider to view the data. Please email edward.smith05@imperial.ac.uk if you have any questions about these solutions (or any other aspect of the course).

Also check out open-source `pyDataView` which is based on this type of data analysis, collaborators are always welcome and it may be useful for your project.