Lecture 2.3

Debrief

Please submit the questions as different files with the names as given.

A function **must** `return`. This is how a function passes it's result to the rest of the program. Otherwise, all your work making something is lost. e.g. if I wanted you to use `divisors` in a larger program.

Python can loop over any iterable. e.g.

```
thetas = np.linspace(0,np.pi/2,101)
for theta in thetas:
    ...
```

Constructs such as the following are messy, limited, and strongly discouraged.

```
n = 0
while n < 100: # don't do this
    theta = (np.pi/200)*n
    ... # this method conflates making theta with iterating over theta
    n += 1 # you also don't have a list of thetas later for plotting
```

If you want to number them as you go, use `enumerate`:

```
vals = np.zeros(len(thetas)) # len(thing) given the length of a list, array, tuple, etc
for n,theta in enumerate(thetas):
    vals[n] = thetas**2 # or whatever
```

LaTeX in labels: (no assessment based on this; just for future reference)

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-2,2,1001)
y = x**2+np.sqrt(x)
plt.plot(x,y,label=r'equation $y=f(x)$')
plt.xlabel(r'$x$') # note the r before the start of the strings
plt.ylabel(r'$y$')
plt.title(r'A graph of $y=x^2+\sqrt{x}$')
plt.legend()
```

Loading and saving

Get data from a file and save it to a file

```
import numpy as np
np.savetxt # save an array to a text file
np.loadtxt # load a correctly formatted file into a numpy array
np.genfromtxt # more flexible way to load text files into arrays
```

The most common use is to load a CSV file i.e. a text file formatted as values, to be interpreted as numbers, separated by a comma. This is the simplest spreadsheet-like format.

```
def f(x): return x**2
x = np.linspace(-2,2,101)
y = f(x)
A = np.zeros((len(x),2)) # make an empty array with 101 rows and 2 columns
A[:,0] = x               # fill the zeroth column with x values
A[:,1] = y               # and filll the first column with y values
np.savetxt("data.csv", A, delimiter=",")      # save 2D array to data.csv
data = np.loadtxt("data.csv", delimiter=",") # load 2D array of data
```

Vectors

numpy arrays can be used to make something like a vector

```
import numpy as np
x = np.array([1,0,0])
y = np.array([0,1,0])
z = np.array([0,0,1])

r = x + 2*y - z
r
```

```
[ 1  2 -1]
```

They even do dot and cross products:

```
r.dot(x+2*y)
np.cross(x,y)
```

```
[0 0 1]
```

Matrices

We've met arrays. Arrays can be used to represent a matrix.
And you can do a dot product of an array with a vector. Here's the rotation matrix:

```
def R(theta):
    from numpy import array, cos, sin
    return array([[cos(theta), sin(theta)],[-sin(theta),cos(theta)]])

x = np.array([1,0])
R(np.pi/4).dot(x)
```

```
[ 0.70710678 -0.70710678]
```

Notice that I don't have to assign the value returned by R to something and then use it; I can use it in-line.
Higher dimensional arrays are possible. Make either literally (which is tedious and not usually necessary) or with `np.zeros`. e.g. here's a literal $2 \times 2 \times 2$ array:

```
A = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])
A.shape
```

```
(2, 2, 2)
```

Meet other linear algebra functions later: `from scipy.linalg import inv,det`

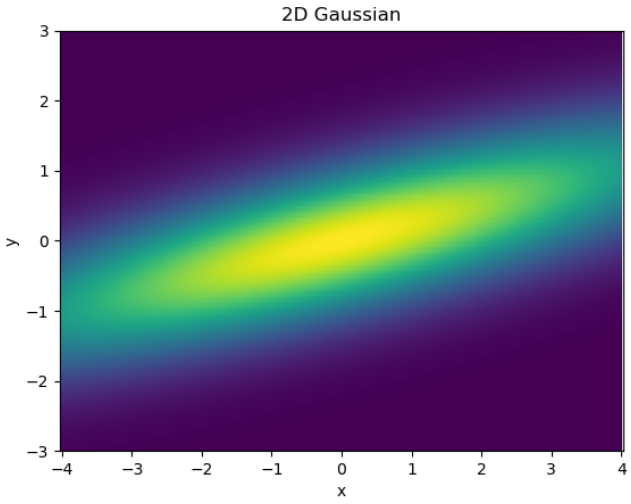
Plotting and fitting with uncertainties

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

def f(x,a,b): return a*x**3 + b*x**2

x = np.linspace(-1,1,101)
sigmay = np.ones(len(x))*0.5 # make an array same length as x containing 0.5s
noise = np.random.randn(len(x))*sigmay # Gaussian white noise
y = f(x,3,-2) + noise

plt.errorbar(x,y,sigmay,label='Data') # like plt.plot but with error bars
guess = [1,-1]
opt,cov = curve_fit(f,x,y,p0=guess,sigma=sigmay)
plt.plot(x,f(x,*opt),label='Best fit')
plt.legend()
```

Plotting 2D arrays

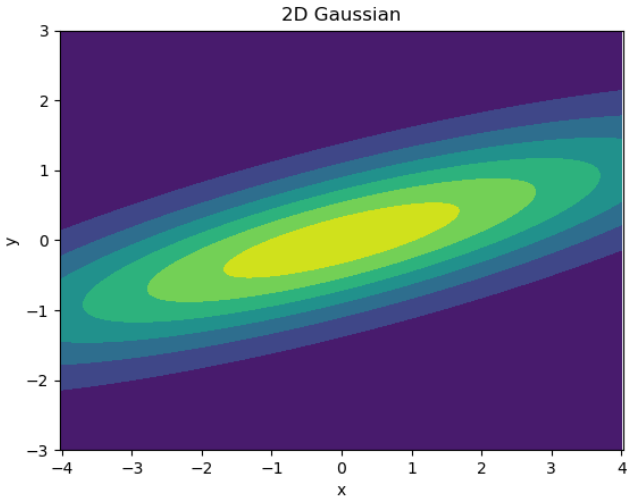
Often, we want to plot a 2D grid.

```
def f(x,y):
    return np.exp(-x**2/10-y**2+x*y/2)

x = np.linspace(-4,4,401)
y = np.linspace(-3,3,301)

N, M = len(x), len(y)
A = np.zeros((N,M))
for n,xi in enumerate(x):
    for m, yi in enumerate(y):
        A[n,m] = f(xi,yi)

plt.pcolormesh(x,y,A.transpose())
plt.axis('equal'); plt.xlabel('x'); plt.ylabel('y'); plt.title('2D Gaussian')
plt.savefig('figs/pcolormesh.png'); plt.close('all')
```



Also contour plots

```
plt.contourf(x,y,A.transpose())
plt.axis('equal'); plt.xlabel('x'); plt.ylabel('y'); plt.title('2D Gaussian')
plt.savefig('figs/contourf.png'); plt.close('all')
```



Array calculation in 2D [BEYOND]

I have advocated using arrays to speed up calculation e.g.

```
def g(x): return x**2
x = np.linspace(-1,1,1001)
y = g(x)
```

not

```
y = list()
for xi in x:
    y.append(g(xi))
```

or

```
y = [g(xi) for xi in x] # list comprehension is great, but arrays are faster
```

unless necessary because the function can't handle arrays.
Similarly, we can (usually) do the same trick with 2D arrays:

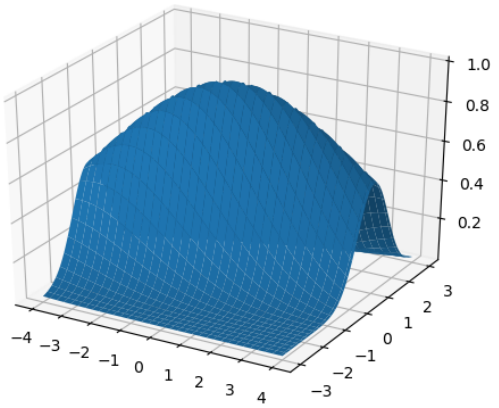
```
x = np.linspace(-4,4,101)
y = np.linspace(-3,3,101)
X,Y = np.meshgrid(x,y)
Z = f(X,Y) # this does the same as the two nested for loops on the previous page
```

Many other styles of plot are available. See Scipy Lectures or Matplotlib documentation.

3D plots [BEYOND]

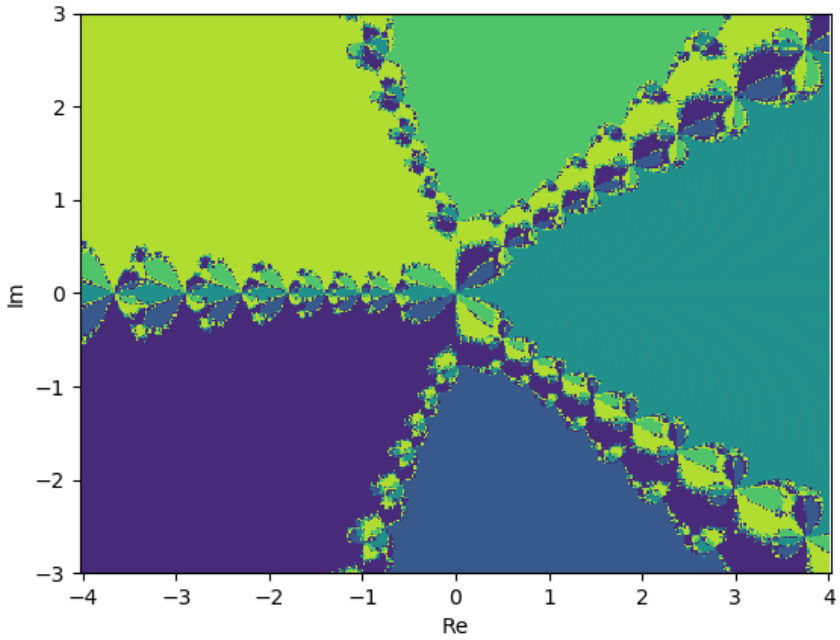
This is beyond the scope of the course. A minimal example is included here for reference. See matplotlib mplot3d tutorial.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.gca(projection='3d')

surf = ax.plot_surface(X, Y, Z)
plt.savefig("figs/plot3d.png"); plt.close('all')
```



Newton's fractal and illustrating `enumerate`

```
def f(z):
    return z**5 + z**3 - 1

x = np.linspace(-4,4,401)
y = np.linspace(-3,3,301)

N, M = len(x), len(y)
A = np.zeros((N,M))
for n,xi in enumerate(x):
    for m, yi in enumerate(y):
        z = xi + 1j*yi
        solution = newton(f, z)  # scipy's newton will use an alternative
                                 # method unless the derivative is given
                                 # (this demo uses the implementation from the notes)
        A[n,m] = np.angle(solution) # take the angle of this complex number
                                    # (for this example, angle is a good way to distinguish solutions

plt.pcolormesh(x,y,A.transpose())
plt.axis('equal'); plt.xlabel('Re'); plt.ylabel('Im')
plt.savefig('figs/newtonfractal.png'); plt.close('all')
```



Assignment 2 preview

Likely to involve:

- Loading real data
- Numerical integration of a curve based on real data
- Fitting theoretical curves to real data and extracting fit parameters
- Plotting 2D data
- Working with vectors and matrices