Tutorial MATH 1MP3 – March 7, 2017

The following tutorial goes over the basics of the numpy library, covering arrays in depth and all the little things you'll want to do with them. Try running the code presented (with print statements as necessary) and answering all the little questions sprinkled throughout the file.

The goal of this tutorial is to get you acquainted with all the many little tricks in numpy for manipulating arrays (beyond just base-python methods) and all the special functions for creating special arrays. That way you've at least *once* seen how to do it, so you have a resource to flip back to.

At the end of the tutorial is one question (similar in size to previous tutorials) to test these concepts.

Import numpy
Before you do anything you should import numpy

```
import numpy as np
```

Working with Arrays
Making an array in numpy is pretty straight forward.

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

There are also many convenient numpy functions for making special arrays, try the following bits of code out and see what the result is.

```
a1 = np.zeros((2,3,4))

a2 = np.random.random((2,2))

a3 = np.empty((3,2))

a4 = np.ones((3,3,3))

a5 = np.full((2,2),7)

a6 = np.arange(10,25,5)

a7 = np.linspace(0,2,9)

a8 = np.eye(7)
```

What is the difference between arange and linspace?

Download the following text file: http://beastman.ca/math1mp/lab7_data.txt and open it in your text reader to see what's inside.

Run the following code

```
x, y, z = np.loadtxt('lab7_data.txt', skiprows=1, unpack=True)
```

and

```
A = np.loadtxt('lab7_data.txt', skiprows=1, unpack=False)
```

It's maybe easy to see that the skiprows argument tells numpy to ignore the first row of the text file (why do we want to do that?).

Based on the difference between the two commands, what does unpack=True do?

Now try the following snippet

```
A1 = np.eye(2)
A2 = np.array([[1,2],[3,4]])

print(A1*A2)
```

What do you get? How does this differ from if you multiplied the two matrices together? See if you can find a suitable function to do matrix multiplication in numpy, hint: use `print(dir(np))`.

Now we'll cover saving a numpy array to a file. Which may be useful if you do a bunch of complicated operations on a matrix in numpy and want to be able to access it later.

```
B = np.array([np.linspace(0,5,7),np.linspace(5,10,7)])
C = np.matmul(B,B.transpose())

np.savetxt('test.txt', C, delimiter=",")
```

Any guesses what the delimiter argument does? Open up test.txt in a text editor and see what the result was.

Now there are lots of way to visualise your numpy arrays.

```
print(C.ndim)
print(C.size)
print(C.shape)

print(C.itemsize) # prints the length of one array element in bytes
# i.e. how many bytes it takes to store any one element in C
print(C.nbytes) # prints the total bytes that C is using
```

Broadcasting

Now numpy does addition and subtraction of arrays in a way that you expect when the arrays are the same shape.

```
x = np.ones((2,3))
print(x.shape)
print(x)

y = np.random.random((2,3))
print(y.shape)
```

```
print(y)

print(x-y)
```

But numpy is also clever about what it does when the shapes are *not* the same.

```
x = np.ones((3,4))
print(x.shape)
print(x)

y = np.arange(4)
print(y.shape)
print(y)

print= (x-y)
```

This is what is meant by numpy *broadcasting*. What exactly is numpy doing here?

What about the following:

```
x  = np.ones((3,4))
print(x.shape)

y = np.random.random((5,4))
print(y.shape)

print(x+y)
```

What is the output? What went wrong?

Numpy can do broadcasting when one of the dimensions is 1 or when the arrays have the same size. In this example, neither is true. So numpy doesn't know what to do. Whereas in the last example, we had a dimension that was 1, and things worked easier.

Some intersting special numpy functions

We've seen that standard operations (+, -, *, /, or %) all work "element-wise" on numpy arrays. You can also do more complicated functions on numpy arrays.

For instance numpy as the following functions:

- np.sqrt
- np.exp
- np.sin
- np.cos
- np.log

It's worth noting that np.log refers to the *natural* log (also called ln in most classes) and np.exp to the exponential function.
Try the following

```
I = np.eye(2)
E = np.exp(I)
print(E)

print(np.log(E))
```

Knowing (from above) that np.random.random returns numbers uniformly distributed between 0 and 1, try to puzzle out what the following code does.

```
b = 5
a = 1

A = np.random.random((2,2)) * (b-a) + a
```

Numpy also has many different properties of arrays that come in handy, try running the following to see what each function does.

```
print(A)
print(A.sum())
print(A.min())
print(A.max())
print(A.max(axis=0))

print(A.mean())

print(np.std(A))
```

Getting Help

We've already seen the dir() and help() functions for getting help before. For instance, in a shell run:

```
help(np.std)
```

to perfectly understand what the last function did. You can also do a search based on keywords (it looks in docstrings).

```
np.lookfor("Standard Deviation")
```

Reshaping Arrays

The following is nice for generating matrices from arrays, or vice versa.

```
x = np.arange(0,5,0.1)
print(x)
print(x.shape)
print(x.size)

y = x.reshape(5,10)
```

```
print(y)
print(y.shape)
print(y.size)
```

Notice that y.size and x.size are the same. So you can take any array and make it into a matrix of any shape, as long as the same number of elements are in the result. i.e. if you take an array of size L you can use reshape to reshape it into any matrix of shape (M, N) as long as M*N=L.

Just to see an example where we do this wrong consider

```
y = x.reshape(5,5)
```

Now, we've seen how we can reshape arrays into matrices, we can also flatten matrices into arrays.

```
z = y.ravel() # think "unravel"
print(z)
print(x)
print(np.array_equal(z,x))
```

Final Question

1. Write a function called `random_matrix_fun(n)` that:
   - Takes an argument n for dimension.
   - Generates a matrix with (n+1) columns and n rows (call it B) full of random values between -5 and 5
   - Use Python splicing to seperate the matrix B into a (nxn) matrix A and a vector x
   - Compute `b=Ax` using matrix vector multiplication and print out b
   - Find an appropriate function to calculate the eigenvalues and eigenvectors of A
     - Recall from a Linear algebra course: if λ is an eigenvalue of A, and **x** is an eigenvector of A then: A**x** = λ**x.** Also an nxn matrix has n eigenvalues and n eigenvectors.
     - If w is a list of eigenvalues and X is a matrix of eigenvectors (where the columns are the vectors, numpy returns like this), then A X[:,i] is equal to w[i] X[:,i] (where multiplication is matrix multiplication).
   - Verify that the identity A**x** = λ**x** is true for each eigenvalue in A
     - have a print statement that says “All eigenvalues are correct” or “At leat one Eigenvalue is incorrect” depending on the result.
     - Remember – you’re probably dealing with floating points! Things could be “equal” but `X==Y` might return `False` (see Lab 5 for more help).

       You might want to do something like this:
       to test if array `X` is equal to array `Y` try
       `sum(abs(X-Y)) < 10**(-5) # or some other tolerance`
       instead of `X == Y`

       Let’s break down what’s going on:
       - `X-Y` does subtraction ELEMENT-WISE. So if the arrays were perfectly equal, each entry would be 0.
       - `abs(X-Y)` then takes these differences and makes them all positive.

        - `sum(abs(X-Y))` now adds all these positive differences up. If they were equal, this would be 0. For floating point, we expect this to be less than some tolerance, here I chose `10**(-5)` arbitrarily, it's small enough.
    - Return a python-base list with the matrix A, the vector b (that you printed earlier in the function), and a list of all the eigenvalues of A.
        - i.e. your output should be something like: [A,b, [λ_1,λ_2,...,λ_n]].