Sayyed Mohsen Vazirizade

23398312

smvazirizade@email.arizona.edu

INFO 521

Introduction to Machine Learning

Assignment 1

Problem 1

If you haven't already, setup your programming environment! Python is recommended:

http://w3.sista.arizona.edu/~clayton/courses/ml/python_setup.html

If python is new to you, work through the short python tutorial:

http://w3.sista.arizona.edu/~clayton/courses/ml/tutorial.html

Problem 2

Exercise 1.1 from FCMA p.35

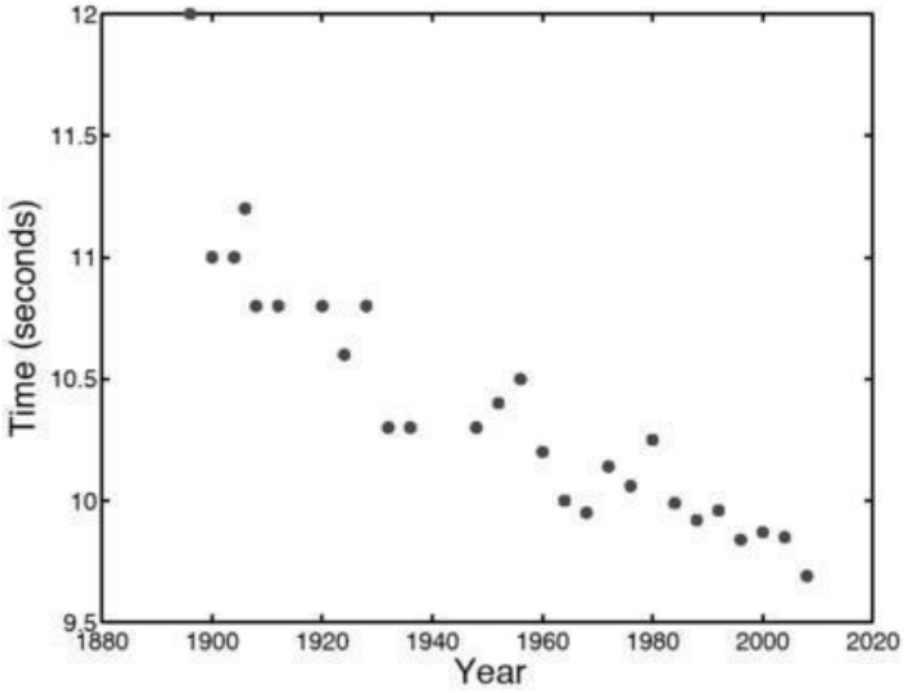


Figure 1 Reproduction of figure 1.1, Olympic men's 100m data

By examining Figure 1.1 [from p. 2 of FCMA, reproduced here], estimate (by hand / in your head) the kind of values we should expect for w_0 (y-intercept) and w_1 (slope) as parameters of a line a to the data (e.g., High? Low? Positive? Negative?). (No computer or calculator calculation is needed here {just estimate!)

If we draw an estimated line which shows the relation between y and x:

$$y = a \times x + b$$

Or in this case we can define it as:

$$time = w_1 \times year + w_0$$

Can observe the trend of the points and the relation between the two sets of data. This line, showing the trend, is shown in the following figure in orange. As shown in the following figure, time (time record) is inversely proportional to year; which means the value of record reduces during the time. Based on this the value of slope, w_1, should be negative. Furthermore, at a rough guess, when x is 0, y is 11.5 seconds.

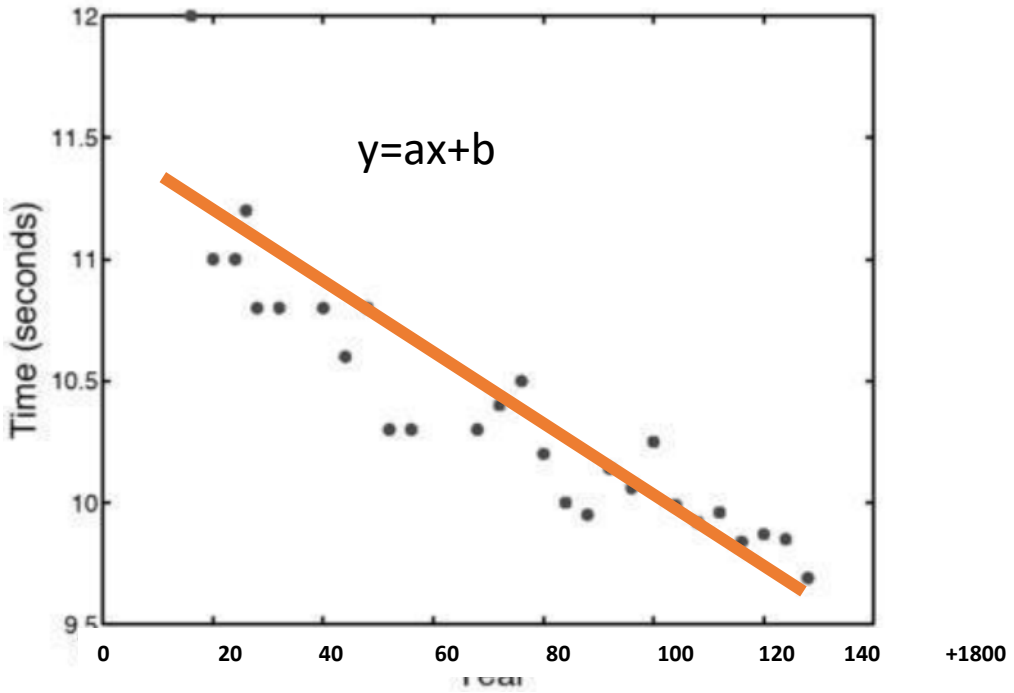


Figure 2Plotted data and the trend line

$$y = a \times x + b \Rightarrow 11.5 = a \times 0 + b \Rightarrow b = 11.5$$

$$a = -\frac{11.5 - 9.5}{130} \Rightarrow a \approx -0.0154$$

So we have:

$$y = -0.0154 \times x + 11.5$$

However, we have to change *a* and *b* to w_0 and w_1

$$\begin{cases} y = time \\ x + 1880 = year \end{cases} and\ y = -0.0154\ x + 11.5 \Rightarrow\ time = -0.0154\ year + 40.45$$

The following table provide an estimation about the precision of the calculated line

Table 1 calculated value based on the estimated line

year	Estimated time
1880	11.4980

1900	11.1900
1920	10.8820
1940	10.5740
1960	10.2660
1980	9.9580
2000	9.6500
2020	9.3420

Thus, the final answers are as follow:

$$w_0 = 40.45$$

$$w_1 = -0.0154$$

$$Equation: \ time = -0.0154\, year + 40.45$$

Problem 3

NOTE: The following three exercises (3, 4 and 5) review basic linear algebra concepts.

Notation conventions:

• Script variables, such as x_{n2} and w_1 represent scalar values

• Lowercase bold-face variables, such as $\mathbf{x}_n$ and w, represent vectors

• Uppercase bold-face variables, such as **X**, represent n (rows) × m (columns) matrices

• Note that because all indexes in the following are either a value between 0, 1, ..., 9, or a scalar, n, I am representing multiple dimension indexes without a comma, as it is unambiguous; e.g., x_{32} is the element scalar value of X at row 3, column 2. When we have to refer to specific index values greater than 9, we'll use commas, such as $x_{32;3}$ is the scalar value in the 32nd row and 3rd column.

• `T' in expressions like w^T indicates the transpose operator.

• Unless stated otherwise, we will assume that all vectors without the transpose, T, are column vectors, so w^T is a row vector.

• It is sometimes convenient to express an example vector as a bracketed list of elements (e.g., in a sentence): [x1; x2; … ; xn]. In general I am going to try to be careful about the orientation of vectors, so the previous example would be a row vector. To make it a column vector, I'll add a transpose: [x1; x2; :::; xn]^T.

$$w^T X^T X w = w_0^2 \left(\sum_{n=0}^{N} w_{n1}^2 \right) + 2 w_0 w_1 \left(\sum_{n=0}^{N} x_{n1} x_{n2} \right) + w_1^2 \left(\sum_{n=0}^{N} w_{n2}^2 \right)$$

where

$$w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ \vdots & \vdots \\ x_{N1} & x_{N2} \end{bmatrix}$$

The formula for calculation of matrix multiplication is as follow:

$$\mathbf{X^T X} \equiv c_{ij} = \sum_{k=1}^{m} a_{ik}\, c_{kj}$$

or

$$\mathbf{X^T X} = \begin{bmatrix} x_{11} & x_{21} & x_{31} & \dots & x_{N1} \\ x_{12} & x_{22} & x_{32} & \dots & x_{N2} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ \vdots & \vdots \\ x_{N1} & x_{N2} \end{bmatrix}$$

For the first step we calculate:

$$\mathbf{X^T X} = \begin{bmatrix} x_{11}^2 + x_{21}^2 + x_{31}^2 + \dots + x_{N1}^2 & x_{11}x_{12} + x_{22}x_{22} + x_{31}x_{32} + \dots + x_{N1}x_{N2} \\ x_{11}x_{12} + x_{21}x_{22} + x_{31}x_{32} + \dots + x_{N1}x_{N2} & x_{12}^2 + x_{22}^2 + x_{32}^2 + \dots + x_{N2}^2 \end{bmatrix}$$

then calculate the multiplication of the previous step and w:

$$\mathbf{X^T X\, w} = \begin{bmatrix} x_{11}^2 + x_{21}^2 + x_{31}^2 + \dots + x_{N1}^2 & x_{11}x_{12} + x_{22}x_{22} + x_{31}x_{32} + \dots + x_{N1}x_{N2} \\ x_{11}x_{12} + x_{21}x_{22} + x_{31}x_{32} + \dots + x_{N1}x_{N2} & x_{12}^2 + x_{22}^2 + x_{32}^2 + \dots + x_{N2}^2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

$$\mathbf{X^T X\, w} = \begin{bmatrix} (x_{11}^2 + x_{21}^2 + x_{31}^2 + \dots + x_{N1}^2)w_0 + (x_{11}x_{12} + x_{22}x_{22} + x_{31}x_{32} + \dots + x_{N1}x_{N2})w_1 \\ (x_{11}x_{12} + x_{22}x_{22} + x_{31}x_{32} + \dots + x_{N1}x_{N2})w_0 + (x_{11}^2 + x_{21}^2 + x_{31}^2 + \dots + x_{N1}^2)w_1 \end{bmatrix}$$

as the last step we premultiply the previous step by w^T

$$w^T\, \mathbf{X^T X\, w} = [w_0 \quad w_1] \begin{bmatrix} (x_{11}^2 + x_{21}^2 + x_{31}^2 + \dots + x_{N1}^2)w_0 + (x_{11}x_{12} + x_{22}x_{22} + x_{31}x_{32} + \dots + x_{N1}x_{N2})w_1 \\ (x_{11}x_{12} + x_{22}x_{22} + x_{31}x_{32} + \dots + x_{N1}x_{N2})w_0 + (x_{11}^2 + x_{21}^2 + x_{31}^2 + \dots + x_{N1}^2)w_1 \end{bmatrix}$$

$$w^T\, \mathbf{X^T X\, w} = \begin{bmatrix} (x_{11}^2 + x_{21}^2 + x_{31}^2 + \dots + x_{N1}^2)w_0^2 + (x_{11}x_{12} + x_{22}x_{22} + x_{31}x_{32} + \dots + x_{N1}x_{N2})w_0w_1 \\ (x_{11}x_{12} + x_{22}x_{22} + x_{31}x_{32} + \dots + x_{N1}x_{N2})w_0w_1 + (x_{11}^2 + x_{21}^2 + x_{31}^2 + \dots + x_{N1}^2)w_1^2 \end{bmatrix}$$

$$\begin{aligned} w^T\, \mathbf{X^T X}\, w = [&(x_{11}^2 + x_{21}^2 + x_{31}^2 + \dots + x_{N1}^2)w_0^2 + (x_{11}x_{12} + x_{22}x_{22} + x_{31}x_{32} + \dots + x_{N1}x_{N2})w_0w_1 \\ &+ (x_{11}x_{12} + x_{22}x_{22} + x_{31}x_{32} + \dots + x_{N1}x_{N2})w_0w_1 \\ &+ (x_{11}^2 + x_{21}^2 + x_{31}^2 + \dots + x_{N1}^2)w_1^2] \end{aligned}$$

then, we can simplify the final answer as:

$$w^T\, X^T\, X\, w = w_0^2 \left(\sum_{n=0}^{N} w_{n1}^2 \right) + 2\, w_0 w_1 \left(\sum_{n=0}^{N} x_{n1}\, x_{n2} \right) + w_1^2 \left(\sum_{n=0}^{N} w_{n2}^2 \right)$$

Problem 4

Using w and **X** as defined in the previous exercise, show that $(\mathbf{X}w)^T = w^T\mathbf{X}^T$ by multiplying out both sides.

$$\mathbf{Xw} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ \vdots & \vdots \\ x_{N1} & x_{N2} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} x_{11}w_0 + x_{12}w_1 \\ x_{21}w_0 + x_{22}w_1 \\ x_{31}w_0 + x_{32}w_1 \\ \vdots \\ x_{N1}w_0 + x_{N2}w_1 \end{bmatrix}$$

$$(\mathbf{Xw})^T = \begin{bmatrix} x_{11}w_0 + x_{12}w_1 \\ x_{21}w_0 + x_{22}w_1 \\ x_{31}w_0 + x_{32}w_1 \\ \vdots \\ x_{N1}w_0 + x_{N2}w_1 \end{bmatrix}^T$$

$$= [x_{11}w_0 + x_{12}w_1 \quad x_{21}w_0 + x_{22}w_1 \quad x_{31}w_0 + x_{32}w_1 \quad \cdots \quad x_{N1}w_0 + x_{N2}w_1]$$

$$\mathbf{w^T X^T} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}^T \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ \vdots & \vdots \\ x_{N1} & x_{N2} \end{bmatrix}^T = [w_0 \quad w_1] \begin{bmatrix} x_{11} & x_{21} & x_{31} & \dots & x_{N1} \\ x_{12} & x_{22} & x_{32} & \dots & x_{N2} \end{bmatrix}$$

$$\mathbf{w^T X^T} = [x_{11}w_0 + x_{12}w_1 \quad x_{21}w_0 + x_{22}w_1 \quad x_{31}w_0 + x_{32}w_1 \quad \cdots \quad x_{N1}w_0 + x_{N2}w_1]$$

Problem 5

When multiplying a scalar by a vector (or matrix), we multiply each element of the vector (or matrix) by that scalar. For $x_n = [x_{n1}, x_{n2}]^T$, $t = [t_1, \ldots, t_N]^T$, $w = [w_0, w_1]^T$, and

$$\mathbf{X} = \begin{bmatrix} \boldsymbol{x_1^T} \\ \boldsymbol{x_2^T} \\ \boldsymbol{x_3^T} \\ \vdots \\ \boldsymbol{x_N^T} \end{bmatrix}$$

show that

$$\sum_n \mathrm{x}_n \, t_n = \mathrm{X^T t}$$

and

$$\sum_n \mathrm{x}_n \, \mathrm{x}_n^T \, w = \mathrm{X^T X \, w}$$

In the fist step we calculate:

$$\mathrm{X^T t} = \begin{bmatrix} \boldsymbol{x_1^T} \\ \boldsymbol{x_2^T} \\ \boldsymbol{x_3^T} \\ \vdots \\ \boldsymbol{x_N^T} \end{bmatrix}^{\boldsymbol{T}} [t_{\mathbf{1}} \quad \ldots \quad t_N]^{\boldsymbol{T}} = [\boldsymbol{x_1^T} \quad \ldots \quad \boldsymbol{x_N^T}][t_{\mathbf{1}} \quad \ldots \quad t_N]^{\boldsymbol{T}} = \begin{bmatrix} \boldsymbol{x_{11}} & \ldots & \boldsymbol{x_{N1}} \\ \boldsymbol{x_{12}} & \ldots & \boldsymbol{x_{N2}} \end{bmatrix} \begin{bmatrix} t_{\mathbf{1}} \\ \vdots \\ t_N \end{bmatrix}$$

thus we have:

$$\mathrm{X^T t} = \begin{bmatrix} x_{\mathbf{11}} t_{\mathbf{1}} + \ldots + x_{N\mathbf{1}} t_N \\ x_{\mathbf{12}} t_{\mathbf{1}} + \ldots + x_{N\mathbf{2}} t_N \end{bmatrix}$$

now we have to change this one to a summation form:

$$\mathrm{X^T t} = \begin{bmatrix} x_{11} t_1 + \ldots + x_{N1} t_N \\ x_{12} t_1 + \ldots + x_{N2} t_N \end{bmatrix} = \begin{bmatrix} x_{\mathbf{11}} \\ x_{\mathbf{12}} \end{bmatrix} t_{\mathbf{1}} + \cdots + \begin{bmatrix} x_{N\mathbf{1}} \\ x_{N\mathbf{2}} \end{bmatrix} t_N = \mathrm{x}_1 t_{\mathbf{1}} + \cdots + \mathrm{x}_N t_N = \sum_n \mathrm{x}_n \, t_n = \sum_{n=1}^{N} \mathrm{x}_n t_n$$

For the second one we start from the both sides and show they meet each other.

The right side:

$$\mathrm{X^T X\,w} = \begin{bmatrix} x_{11} & x_{21} & x_{31} & \dots & x_{N1} \\ x_{12} & x_{22} & x_{32} & \dots & x_{N2} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ \vdots & \vdots \\ x_{N1} & x_{N2} \end{bmatrix} w =$$

$$\begin{bmatrix} x_{11}^2 + x_{21}^2 + x_{31}^2 + \dots + x_{N1}^2 & x_{11}x_{12} + x_{22}x_{22} + x_{31}x_{32} + \dots + x_{N1}x_{N2} \\ x_{11}x_{12} + x_{21}x_{22} + x_{31}x_{32} + \dots + x_{N1}x_{N2} & x_{12}^2 + x_{22}^2 + x_{32}^2 + \dots + x_{N2}^2 \end{bmatrix} \mathrm{w} =$$

$$\begin{bmatrix} x_{11}^2 + x_{21}^2 + x_{31}^2 + \dots + x_{N1}^2 & x_{11}x_{12} + x_{22}x_{22} + x_{31}x_{32} + \dots + x_{N1}x_{N2} \\ x_{11}x_{12} + x_{21}x_{22} + x_{31}x_{32} + \dots + x_{N1}x_{N2} & x_{12}^2 + x_{22}^2 + x_{32}^2 + \dots + x_{N2}^2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \Rightarrow$$

$$\mathbf{X^T X\,w} = \begin{bmatrix} (x_{11}^2 + x_{21}^2 + x_{31}^2 + \cdots + x_{N1}^2)w_0 + (x_{11}x_{12} + x_{22}x_{22} + x_{31}x_{32} + \cdots + x_{N1}x_{N2})w_1 \\ (x_{11}x_{12} + x_{22}x_{22} + x_{31}x_{32} + \cdots + x_{N1}x_{N2})w_0 + (x_{11}^2 + x_{21}^2 + x_{31}^2 + \cdots + x_{N1}^2)w_1 \end{bmatrix}$$

The left side:

$$\sum_n \mathrm{x}_n\,\mathrm{x}_n^T\,w = \sum_n \begin{bmatrix} \boldsymbol{x_{n1}} \\ \boldsymbol{x_{n2}} \end{bmatrix} [\boldsymbol{x_{n1}} \quad \boldsymbol{x_{n2}}]\,w = \sum_n \begin{bmatrix} x_{n1}^2 & x_{n1}x_{n2} \\ x_{n1}x_{n2} & x_{n2}^2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \sum_n \begin{bmatrix} (x_{n1}^2)w_0 + (x_{n1}x_{n2})w_1 \\ (x_{n1}x_{n2})w_0 + (x_{n2}^2)w_1 \end{bmatrix} \Rightarrow$$

$$\sum_n \mathrm{x}_n\,\mathrm{x}_n^T\,w = \begin{bmatrix} (x_{11}^2 + x_{21}^2 + x_{31}^2 + \cdots + x_{N1}^2)w_0 + (x_{11}x_{12} + x_{22}x_{22} + x_{31}x_{32} + \cdots + x_{N1}x_{N2})w_1 \\ (x_{11}x_{12} + x_{22}x_{22} + x_{31}x_{32} + \cdots + x_{N1}x_{N2})w_0 + (x_{11}^2 + x_{21}^2 + x_{31}^2 + \cdots + x_{N1}^2)w_1 \end{bmatrix}$$

Thus we can see:

$$\sum_n \mathrm{x}_n\,\mathrm{x}_n^T\,w = \mathrm{X^T X\,w}$$

Problem 6

The following script shows all the required parts in the problem statement line by line. Each line has a full description as a text and including the comments.

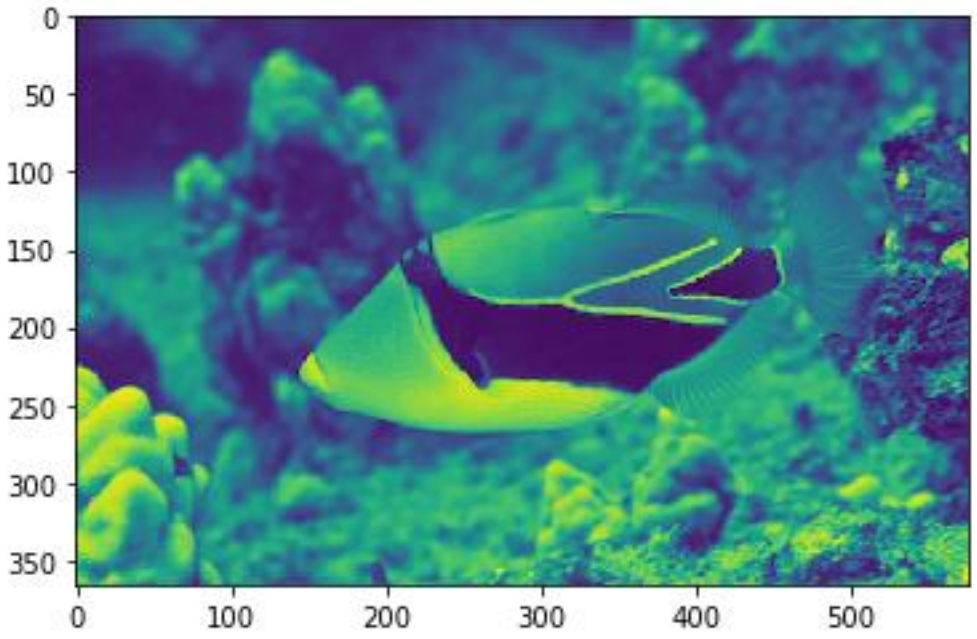
```
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 28 10:51:45 2017

@author: smvaz
"""

#calling numpy and matplotlib.pyplot
import numpy as np
import matplotlib.pyplot as plt
#calling the resource data and showing of interest values
Data=np.loadtxt('data/humu-2.txt')
print('Type of Data is %s ' % type(Data))
print('Total size of Data is %i ' %Data.size)
Size1,Size2=Data.shape
print('The number of columns and rows are %i and %i, respetively' % (Size1, Size2))
#Rescaling according the problem statement and printing the values
Min, Max =Data.min(), Data.max()
print('The min and max vaules are %f and %f, respetively' % (Min, Max))
DataScaled=(Data-Min)/(Max-Min)
SizeScaled1,SizeScaled2=DataScaled.shape
print('The number of columns and rows are %i and %i, respetively' % (SizeScaled1, SizeScaled2))
MinScaled, MaxScaled =DataScaled.min(), DataScaled.max()
print('The min and max vaules of DataScaled are %f and %f, respetively' % (MinScaled, MaxScaled))
#verify the resaults
print('Do they have the same number of elements? %s'  %(Data.size==DataScaled.size))
print('Do they have the same number of rows and columns, respectively? %s %s'  % (Size1==SizeScaled1,Size2==SizeScaled2))

#Producing to uniformly random data with the same dimensions of previous part.
Radnom1=np.random.random((Size1,Size2))
np.savetxt('data/Radnom1.txt', Radnom1)
Radnom2=np.random.random((Size1,Size2))
np.savetxt('data/Radnom2.txt', Radnom2)
#printing
print('original shape')
plt.figure()
plt.imshow(Data)
plt.show()

print('Gray Scale')
plt.figure()
plt.imshow(Data, cmap='gray')
plt.show()

print('Radnom1')
DataRandom1=np.loadtxt('data/Radnom1.txt')
plt.figure()
plt.imshow(DataRandom1)
plt.show()

print('Radnom2')
DataRandom2=np.loadtxt('data/Radnom2.txt')
plt.figure()
```

```
plt.imshow(DataRandom2)
plt.show()

print('Difference between Radnom2 and Radnom1')
plt.figure()
plt.imshow(Radnom2-Radnom1)
plt.show()

plt.close()
#(For fun: Anyone know what `humu' is? Give the full name!)
print('The very last part asks about what Humu means. This is the short form of Humuhumunukunukuapua which is a name of a fish species, the one you can see above')
```

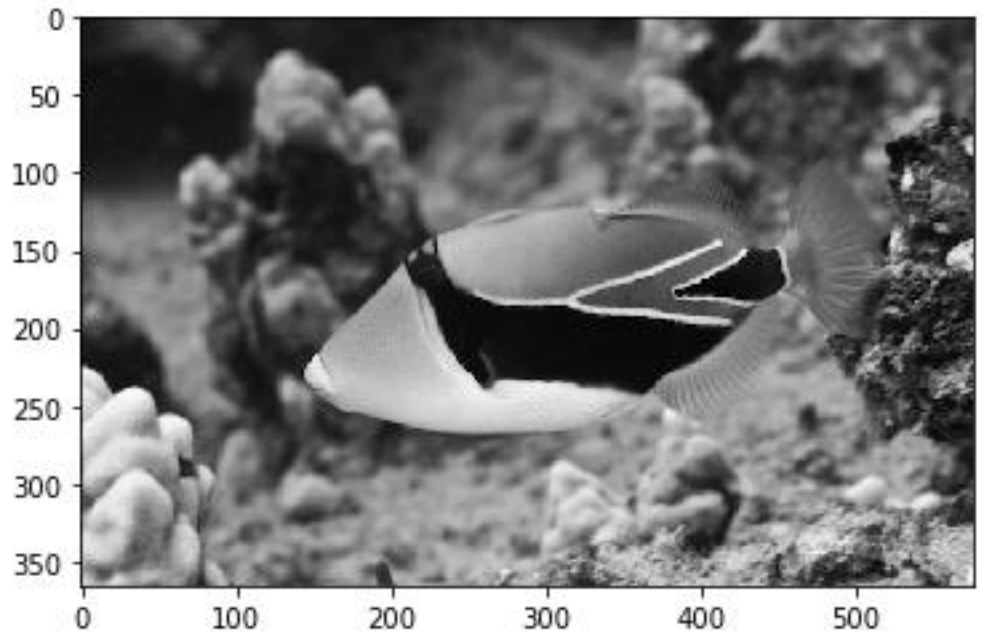
And there is the output in console window:

Note: Because I prefer to copy the answer without any revision and pristine, the figures caption are above the figures and they are not numbered.

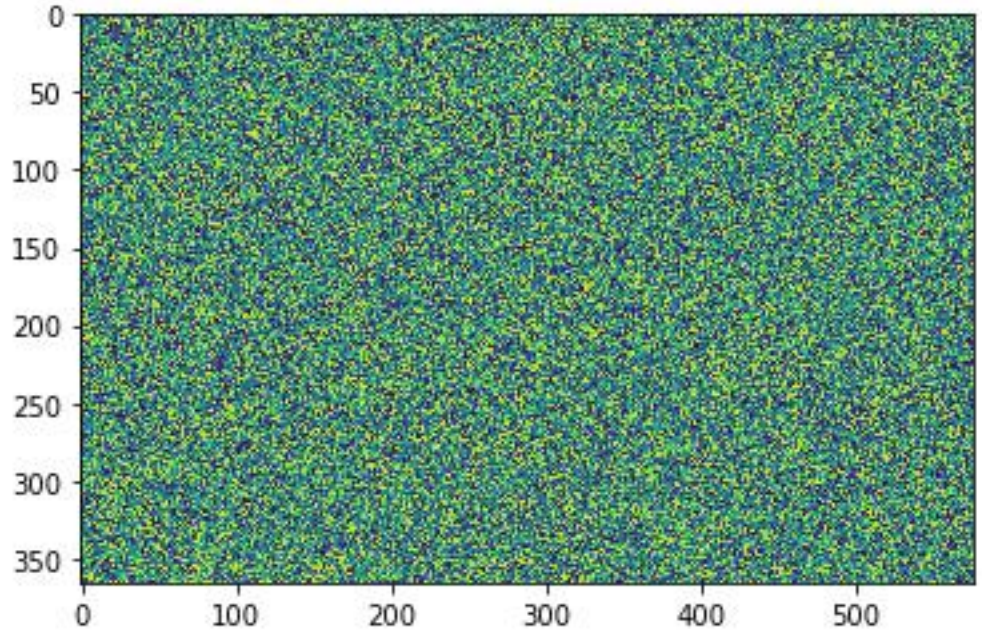
```
runfile('C:/Users/smvaz/Desktop/HW1/assignment_6/hw1.py', wdir='C:/Users/smvaz/Desktop/HW1/assignment_6')
Type of Data is <class 'numpy.ndarray'>
Total size of Data is 210816
The number of columns and rows are 366 and 576, respetively
The min and max vaules are 2.830000 and 15.250000, respetively
The number of columns and rows are 366 and 576, respetively
The min and max vaules of DataScaled are 0.000000 and 1.000000, respetively
Do they have the same number of elements? True
Do they have the same number of rows and columns, respectively? True True

original shape
```



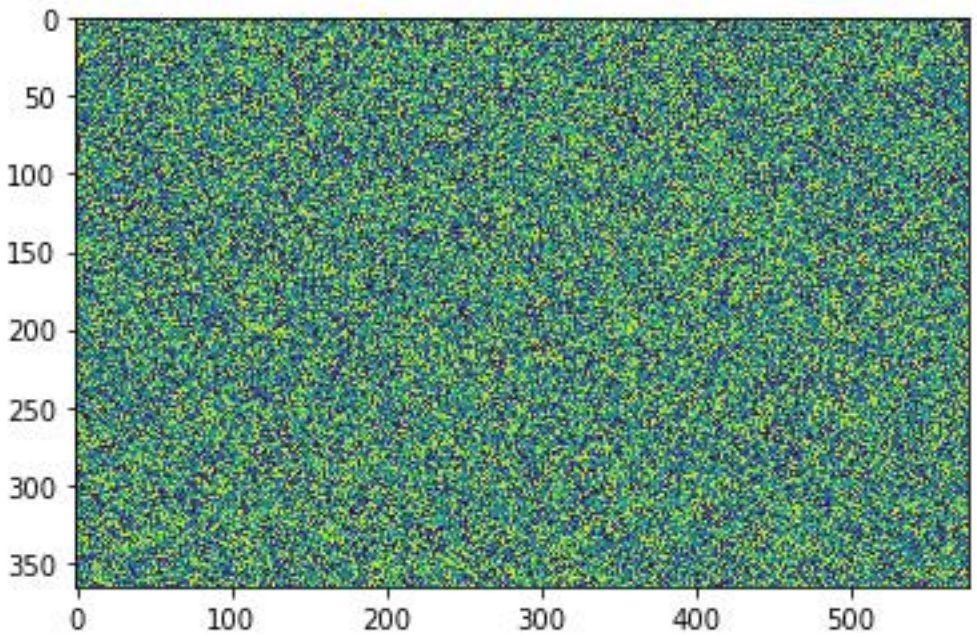
Gray Scale



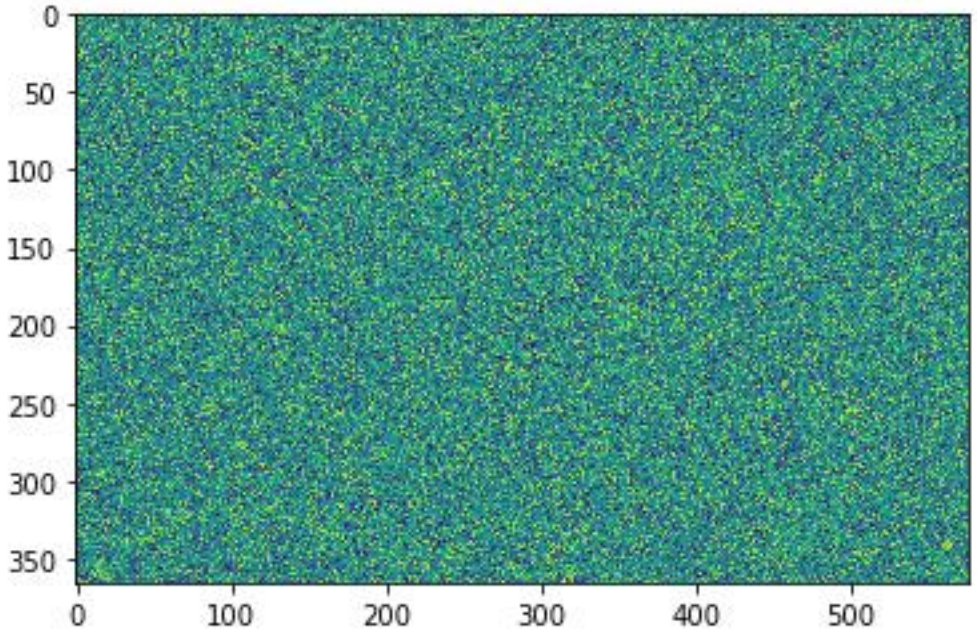
Radnom1



Radnom2



Difference between Radnom2 and Radnom1



The very last part asks about what Humu means. This is the short form of Humuhumunukunukuapua which is a name of a fish species, the one you can see above

Problem 7 and 8

We can change it to a function which is callable by using def command. The name of this function is help(exercise6)

I created a segment in the beginning as help by using “”” and “””.

It can be called by help(exercise6).

The user should pay attention to the fact that there is an indent in each line after def command and when there is no ident it means the function definition finishes there.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 28 10:51:45 2017

@author: smvaz
"""
def exercise6(infile, outfile):
    '''
    This fucntion is written on 8/28/2017
    The fist argument is the exact address and format of the input file, which is an image the user wants to draw
    The second argument is the exact address for saving the output file
    '''
    import numpy as np
    import matplotlib.pyplot as plt
    Data=np.loadtxt(infile)
    print('Type of Data is %s ' % type(Data))
    print('Total size of Data is %i ' %Data.size)
    Size1,Size2=Data.shape
    print('The number of columns and rows are %i and %i, respetively' % (Size1, Size2))
    Min, Max =Data.min(), Data.max()
    print('The min and max vaules are %f and %f, respetively' % (Min, Max))
    DataScaled=(Data-Min)/(Max-Min)
    SizeScaled1,SizeScaled2=DataScaled.shape
    print('The number of columns and rows are %i and %i, respetively' % (SizeScaled1, SizeScaled2))
    MinScaled, MaxScaled =DataScaled.min(), DataScaled.max()
    print('The min and max vaules of DataScaled are %f and %f, respetively' % (MinScaled, MaxScaled))

    print('Do they have the same number of elements? %s'  %(Data.size==DataScaled.size))
    print('Do they have the same number of rows and columns, respectively? %s %s'  % (Size1==SizeScaled1,Size2==SizeScaled2))

    #Producing to uniformly random data with the same dimensions of previous part.
    Radnom1=np.random.random((Size1,Size2))
    np.savetxt(outfile, Radnom1)


    print('original shape')
    plt.figure()
    plt.imshow(Data)
    plt.show()

    print('Gray Scale')
    plt.figure()
    plt.imshow(Data, cmap='gray')
    plt.show()

    print('outfile')
    DataRandom1=np.loadtxt(outfile)
    plt.figure()
```

```
    plt.imshow(DataRandom1)
    plt.show()


    plt.close()

    print('The very last part asks about what Humu means. This is the short form of Humuhumunukunukuapua which is a name of a fish species, the one you can see above')


help(exercise6)
exercise6('data/humu-2.txt','data/random1.txt')
```

Problem 9

Here we define a function, exercise7(itterations), itterations =1000 in this example. We can have two different approach in this problem.

1- We can set the seed number to a specific value, 8 for example, in the beginning of each simulation or
2- We can set it to a specific value for the very first time and then omit it.

In the former we encounter the same results for each try; however, the latter shows various values. The script related to this function is:

```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 29 10:35:48 2017

@author: smvaz
"""

def exercise7(itterations):
    """
    This is a function for generating two sets of N random dices and counting the number of double six.
    """
    import numpy as np
    RandomAll=[0]*itterations
    for j in range(0,itterations):
     RandomAll[j]=[np.random.randint(1,7) for i in range(0,2)]
    i=0
    for j in range(0,itterations):
        if RandomAll[j][1]==RandomAll[j][0]==6:
            i=1+i
    print(i)
    #print(RandomAll)

for j in range(0,10):
 np.random.seed(seed=8)
 help(exercise7)
 exercise7(1000)
```

we encounter the same results for each try; however, the latter shows various values. The script related to this function is:

Based on the two mentioned scenarios, the number of double-six is as the following table.

Table 2Comparsion between setting seed number to 8 in the loop and out of the loop

Try	1	2	3	4	5	6	7	8	9	10
1st Scenario	26	26	26	26	26	26	26	26	26	26
2nd Scenario	26	27	21	26	21	31	27	45	26	21

Furthermore, if we reset the seed number to 8 and run all the process from scratch again the results are exactly the same.

We use seed number in order to be able generate a number of random number; however, by virtue of using a certain value for seed number we can repeat those random numbers. Sometimes we need random number; nevertheless, we need that experiment repeatable. In this regard, seed number comes in handy.

Problem 10 a

This question is not very clear. Although the whole procedure remains the same but in detail that might be some differences. The phrase three-dimensional is a little bit confusing. In some references a vector with row, columns, and depth is called 3D; however, we assume the question ask a function to generate two column vectors (3,1). Even though it is possible to consider the dimensions of the vector as an input for the function. Furthermore, the seed number is defined at the beginning of the function; thus, the answer for these two vectors is not the same. Here is the script:

```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 29 10:35:48 2017

@author: smvaz
"""
import numpy as np
def exercise10(Seed):
    """
    This is a function for generating two sets of 3D random numbers.
    """
    import numpy as np
    np.random.seed(seed=Seed)
    RandomAll1=np.random.rand(3)
#    print('The first 3D random verctor is \n %s' %RandomAll1)
    RandomAll2=np.random.rand(3)
#    print('The second 3D random verctor is \n %s' %RandomAll2)
    return(RandomAll1,RandomAll2)


#this is just a flag
print('mohsen')
a,b=exercise10(5)
print('The first 3D random verctor is \n %s' %a)
print('The second 3D random verctor is \n %s' %b)
```

In the function above, the function produces the both vectors. It uses seed number 5 for the fist one and the next seed for the second one; and here are the results:

```
mohsen
The first 3D random verctor is
 [ 0.22199317  0.87073231  0.20671916]
The second 3D random verctor is
 [ 0.91861091  0.48841119  0.61174386]
```

Another approach is defining a function as it produces just one vector each time, which means we have to call it twice.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 29 10:35:48 2017

@author: smvaz
"""
import numpy as np
def exercise10(Seed):
    """
    This is a function for generating two sets of 3D random numbers.
    """
```

```
    import numpy as np
    np.random.seed(seed=Seed)
    RandomAll1=np.random.rand(3)
#    print('The first 3D random verctor is \n %s' %RandomAll1)
    RandomAll2=np.random.rand(3)
#    print('The second 3D random verctor is \n %s' %RandomAll2)
    return(RandomAll1,RandomAll2)


#this is just a flag
print('mohsen')
a,b=exercise10(5)
print('The first 3D random verctor is \n %s' %a)
print('The second 3D random verctor is \n %s' %b)
```

the results of the second approach is:

```
mohsen
The first 3D random verctor is
 [ 0.22199317  0.87073231  0.20671916]
The second 3D random verctor is
 [ 0.22199317  0.87073231  0.20671916]
```

As you can see because both of them use seed=5 the answer for both is the same.

Problem 10 b

For this part, we just have to add these lines to the script:

```
print('a+b= \n %s' %(a+b))
print('a.*b= \n %s' %(np.multiply(a,b)))
print('transpose(a)*b= \n %s' %(np.dot(np.transpose(a),b)))
```

lines to the script:

the answer with the first approach is:

```
a+b=
 [ 1.14060408  1.35914349  0.81846302]
a.*b=
 [ 0.20392535  0.4252754   0.12645917]
transpose(a)*b=
 0.755659923856
```

and the second approach is

```
a+b=
 [ 0.44398634  1.74146461  0.41343831]
a.*b=
 [ 0.04928097  0.75817475  0.04273281]
transpose(a)*b=
 0.850188526216
```

the script

And typeset the output for each one is:

$$a + b = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 0.2220 \\ 0.8707 \\ 0.2067 \end{bmatrix} + \begin{bmatrix} 0.9186 \\ 0.4884 \\ 0.6117 \end{bmatrix} = \begin{bmatrix} 1.1406 \\ 1.3591 \\ 0.8185 \end{bmatrix}$$

$$a \circ b = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \circ \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 0.2220 \\ 0.8707 \\ 0.2067 \end{bmatrix} \circ \begin{bmatrix} 0.9186 \\ 0.4884 \\ 0.6117 \end{bmatrix} = \begin{bmatrix} 0.2039 \\ 0.4253 \\ 0.1265 \end{bmatrix}$$

$$a^T\ b = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}^T \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 0.2220 \\ 0.8707 \\ 0.2067 \end{bmatrix}^T \begin{bmatrix} 0.9186 \\ 0.4884 \\ 0.6117 \end{bmatrix} = [0.2220 \quad 0.8707 \quad 0.2067] \begin{bmatrix} 0.9186 \\ 0.4884 \\ 0.6117 \end{bmatrix}$$
$$= [0.7557]$$

And for the other one:

$$a + b = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 0.2220 \\ 0.8707 \\ 0.2067 \end{bmatrix} + \begin{bmatrix} 0.2220 \\ 0.8707 \\ 0.2067 \end{bmatrix} = \begin{bmatrix} 0.4440 \\ 1.7415 \\ 0.4134 \end{bmatrix}$$

$$a \circ b = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \circ \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 0.2220 \\ 0.8707 \\ 0.2067 \end{bmatrix} \circ \begin{bmatrix} 0.2220 \\ 0.8707 \\ 0.2067 \end{bmatrix} = \begin{bmatrix} 0.0493 \\ 0.7582 \\ 0.0427 \end{bmatrix}$$

$$a^T\ b = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}^T \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 0.2220 \\ 0.8707 \\ 0.2067 \end{bmatrix}^T \begin{bmatrix} 0.9186 \\ 0.4884 \\ 0.6117 \end{bmatrix} = [0.2220 \quad 0.8707 \quad 0.2067] \begin{bmatrix} 0.2220 \\ 0.8707 \\ 0.2067 \end{bmatrix}$$
$$= [0.8502]$$

Problem 10 C

The following lines should be added to the scrip.

```
np.random.seed(seed=2)
X=np.random.rand(3,3)
print('transpose(a)*X= \n %s' %(np.dot(np.transpose(a),X)))
print('transpose(a)*X*b= \n %s' %(np.dot(np.dot(np.transpose(a),X),b)))
print('X^-1= \n %s' % np.linalg.inv((X)))
```

As we discussed earlier, based on our assumption about b and a these are the results:

```
transpose(a)*X=
 [ 0.51814195  0.49979844  0.47159888]
transpose(a)*X*b=
 1.00857572251
X^-1=
 [[-1.20936675  5.11771977 -3.42333228]
 [-0.96691719  0.279414    1.46561347]
 [ 2.82418088 -4.07257903  2.64627411]]
```

Second approach:

```
transpose(a)*X=
 [ 0.51814195  0.49979844  0.47159888]
transpose(a)*X*b=
 0.647703148412
X^-1=
 [[-1.20936675  5.11771977 -3.42333228]
 [-0.96691719  0.279414    1.46561347]
```

[2.82418088 -4.07257903 2.64627411]]

Problem 11-a

The three figures below are created by the python file, plotlinear.py.

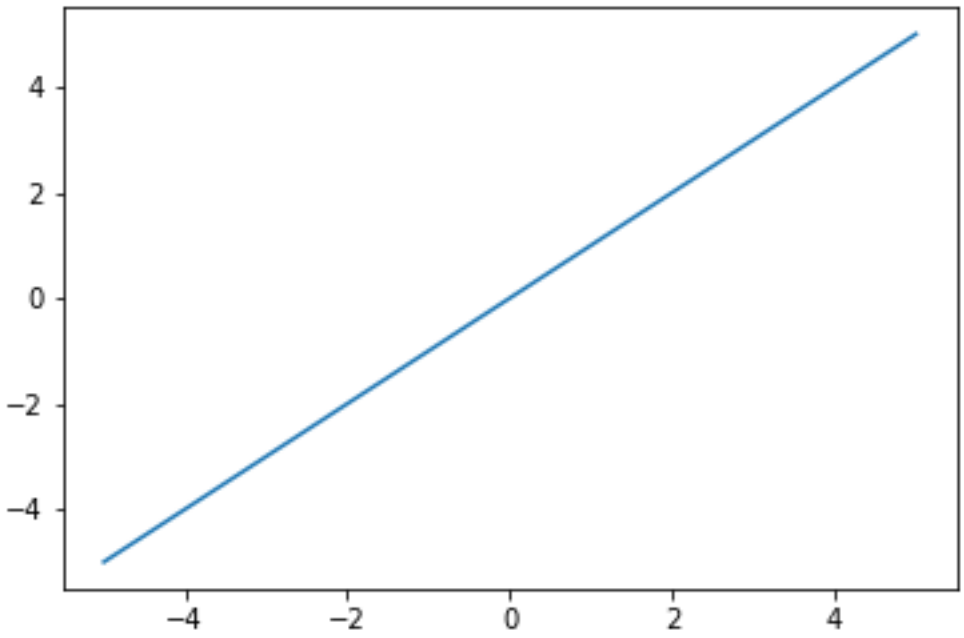


Figure 3 y = 0.0 + 1.0 x which means slope=1, intercept=0

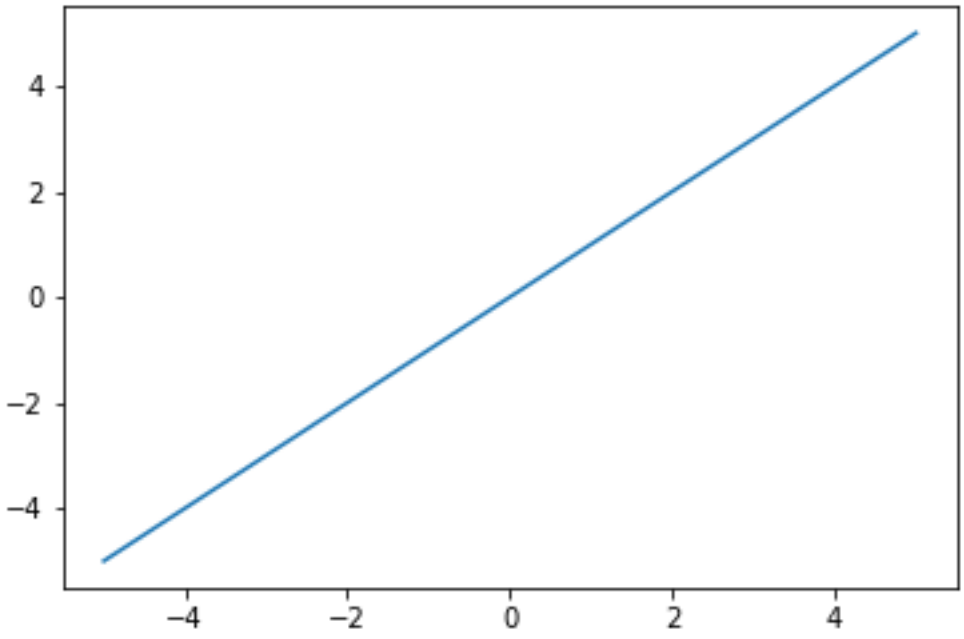


Figure 4 y = 1.0 + 1.0 x which means slope=1, intercept=1

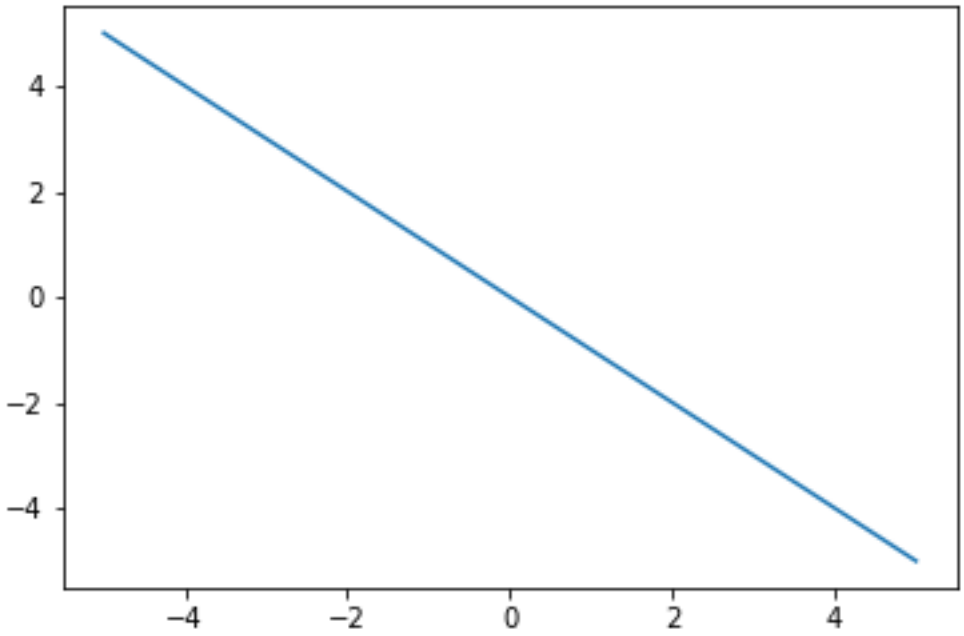


Figure 5 y = 0.0 - 1.0 x which means slope=1, intercept=0

Problem 11-b

Here is the script:

```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 29 22:06:35 2017

@author: smvaz
"""

# plotlinear.py

import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0.0, 10, 0.01)

plt.figure()
plt.plot(x, np.sin(2*np.pi*x))
plt.grid(True)
plt.xlabel('X')
plt.ylabel('sin(x)')
plt.axis('tight')
plt.title('Sine Function for x from 0.0 to 10.0')

plt.show()
```

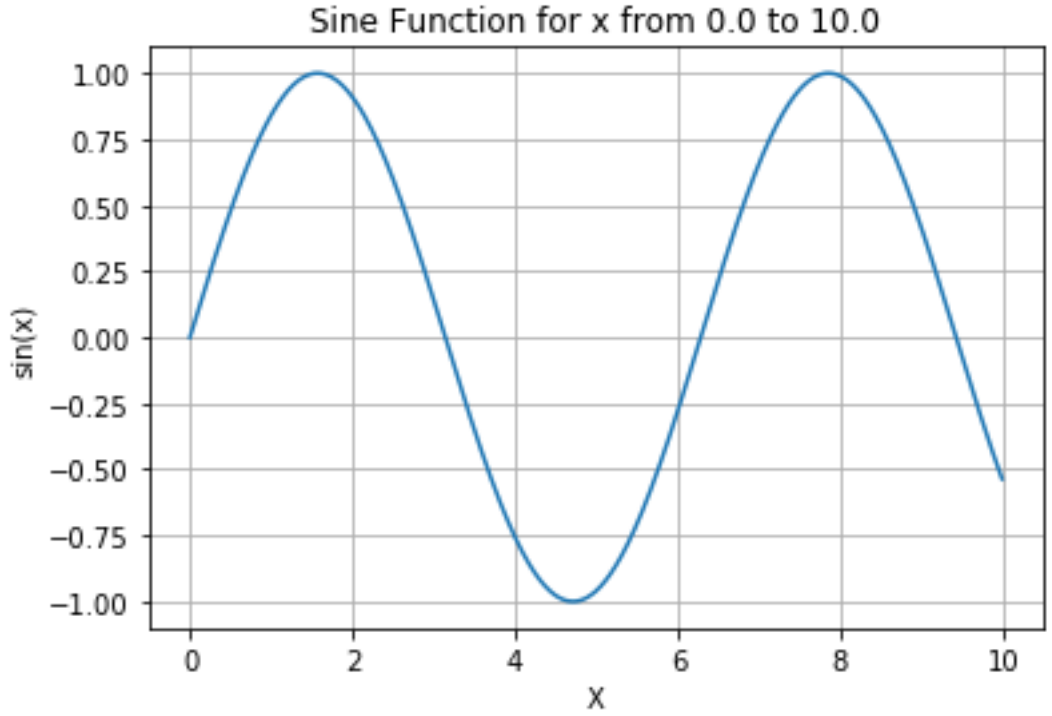


Figure 6The figure drown by Python