Homework 02

Numpy Introduction

1a) Create two numpy arrays (a and b). a should be all integers between 10-19 (inclusive), and b should be ten evenly spaced numbers between 1-7. Print all the results below:

i) Square all the elements in both arrays (element-wise)
ii) Add both the squared arrays (e.g., [1,2] + [3,4] = [4,6])
iii) Sum the elements with even indices of the added array.
iv) Take the square root of the added array (element-wise square root)__

In [16]:

```
import numpy as np
a = np.array(range(10,20))
b = np.linspace(1,7,10)

sq_a = np.array([i**2 for i in a])
sq_b = np.array([i**2 for i in b])

print ("Square of elements of array 'a' = ", sq_a[:])
print ("square of elements of array 'b' = ", sq_b[:])

sq_ab = sq_a+sq_b
print ("The sum of corresponding elements of arrays a and b: ", sq_ab)

sum_even = 0

for i in range(0,len(sq_ab)):
    if i % 2==0:
        sum_even+=sq_ab[i]

print("The sum of all elements in the even indices of the added array: ", sum_even)

sqrt_ab = [i**0.5 for i in sq_ab]
print ("the sqrt of the added array is ", sqrt_ab)
```

```
Square of elements of array 'a' =  [100 121 144 169 196 225 256 289 324 36
1]
square of elements of array 'b' =  [  1.           2.77777778   5.44444444
   9.          13.44444444
  18.77777778  25.          32.11111111  40.11111111  49.        ]
The sum of corresponding elements of arrays a and b:  [ 101.          123.
77777778  149.44444444  178.          209.44444444
  243.77777778  281.          321.11111111  364.11111111  410.        ]
The sum of all elements in the even indices of the added array:  1105.0
the sqrt of the added array is  [10.04987562112089, 11.125546178852424, 1
2.224747213928167, 13.341664064126334, 14.472195564061607, 15.613384571507
158, 16.763054614240211, 17.919573407620817, 19.081695708482279, 20.248456
731316587]
```

1b) Append b to a, reshape the appended array so that it is a 5x4, 2d array and store the results in a variable called m. Print m.

In [2]:

```
appended_array = np.array([a,b])
m = np.reshape(appended_array,(5,4))
print (m)
```

```
[[ 10.          11.          12.          13.        ]
 [ 14.          15.          16.          17.        ]
 [ 18.          19.           1.           1.66666667]
 [  2.33333333   3.           3.66666667   4.33333333]
 [  5.           5.66666667   6.33333333   7.        ]]
```

1c) Extract the second and the third column of the m matrix. Store the resulting 5x2 matrix in a new variable called m2. Print m2.

In [3]:

```
m1 = m[:,1:3]
m2 = np.reshape(m1,(5,2))
print (m2)
```

```
[[ 11.          12.        ]
 [ 15.          16.        ]
 [ 19.           1.        ]
 [  3.           3.66666667]
 [  5.66666667   6.33333333]]
```

1d) Take the dot product of m2 and m store the results in a matrix called m3. Print m3. Note that Dot product of two matrices A.B =A^TB

In [4]:

```
m2 = m2.T
m3 = np.dot(m2,m)
m3
```

Out[4]:

```
array([[ 697.33333333,  748.11111111,  437.88888889,  482.33333333],
       [ 402.22222222,  437.88888889,  454.55555556,  489.88888889]])
```

1e) Round the m3 matrix to two decimal points. Store the result in place and print the new m3.

In [5]:

```
m3 = (np.around(m3,2))
m3
```

Out[5]:

```
array([[ 697.33,  748.11,  437.89,  482.33],
       [ 402.22,  437.89,  454.56,  489.89]])
```

1f) Sort the m3 array so that the highest value is at the top left, the next highest value to the right of the highest, and the lowest value is at the bottom right. Print the sorted m3 array.

In [6]:

```
sorted_array = -np.sort(-m3)

sorted_array
```

Out[6]:

```
array([[ 748.11,  697.33,  482.33,  437.89],
       [ 489.89,  454.56,  437.89,  402.22]])
```

NumPy and Masks

2a) create an array called 'f' where the values are sin(x) for x from 0 to pi with 100 values in f

- print f
- use a 'mask' and print an array that is True when f >= 1/2 and False when f < 1/2
- create and print an array sequence that has only those values where f>= 1/2

In [23]:

```
import numpy.ma as ma

f1 = np.linspace(np.sin(0),np.sin(np.pi/2),50)
#print ("f1 is", f1)
f2 = np.linspace(np.sin(np.pi/2 + 0.01),np.sin(np.pi),50)
#print ("f2 is", f2)

f = np.array([f1,f2])
f = np.reshape(f,(1,100))

mask = f>=0.5
print("The array sequence that is true when f>=1/2 and false otherwise is: \n", mask,
"\n\n")

x = ma.masked_less_equal(f,0.5)
print(" The array with elements of f>=1/2 : \n", x.compressed())
```

```
The array sequence that is true when f>=1/2 and false otherwise is:
 [[False False False False False False False False False False False False
  False False False False False False False False False False False False
  False  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True  True  True  True  True  True  True  True  True  True
   True  True  True False False False False False False False False False
  False False False False False False False False False False False False
  False False False False]]


 The array with elements of f>=1/2 :
 [ 0.51020408  0.53061224  0.55102041  0.57142857  0.59183673  0.6122449
  0.63265306  0.65306122  0.67346939  0.69387755  0.71428571  0.73469388
  0.75510204  0.7755102   0.79591837  0.81632653  0.83673469  0.85714286
  0.87755102  0.89795918  0.91836735  0.93877551  0.95918367  0.97959184
  1.          0.99995     0.97954286  0.95913571  0.93872857  0.91832143
  0.89791429  0.87750714  0.8571      0.83669286  0.81628571  0.79587857
  0.77547143  0.75506429  0.73465714  0.71425     0.69384286  0.67343571
  0.65302857  0.63262143  0.61221429  0.59180714  0.5714      0.55099286
  0.53058571  0.51017857]
```

NumPy and 2 Variable Prediction

Let 'x' be the number of miles a person drives per day and 'y' be the dollars spent on buying car fuel (per day).

We have created 2 numpy arrays each of size 100 that represent x and y.
x (number of miles) ranges from 1 to 10 with a uniform noise of (0,1/2)
y (money spent in dollars) will be from 1 to 20 with a uniform noise (0,1)

In [8]:

```
# seed the random number generator with a fixed value
import numpy as np
np.random.seed(500)

x=np.linspace(1,10,100)+ np.random.uniform(low=0,high=.5,size=100)
y=np.linspace(1,20,100)+ np.random.uniform(low=0,high=1,size=100)
print ('x = ',x)
print ('y= ',y)
```

```
x =  [  1.34683976   1.12176759   1.51512398   1.55233174   1.40619168
   1.65075498   1.79399331   1.80243817   1.89844195   2.00100023
   2.3344038    2.22424872   2.24914511   2.36268477   2.49808849
   2.8212704    2.68452475   2.68229427   3.09511169   2.95703884
   3.09047742   3.2544361    3.41541904   3.40886375   3.50672677
   3.74960644   3.64861355   3.7721462    3.56368566   4.01092701
   4.15630694   4.06088549   4.02517179   4.25169402   4.15897504
   4.26835333   4.32520644   4.48563164   4.78490721   4.84614839
   4.96698768   5.18754259   5.29582013   5.32097781   5.0674106
   5.47601124   5.46852704   5.64537452   5.49642807   5.89755027
   5.68548923   5.76276141   5.94613234   6.18135713   5.96522091
   6.0275473    6.54290191   6.4991329    6.74003765   6.81809807
   6.50611821   6.91538752   7.01250925   6.89905417   7.31314433
   7.20472297   7.1043621    7.48199528   7.58957227   7.61744354
   7.6991707    7.85436822   8.03510784   7.80787781   8.22410224
   7.99366248   8.40581097   8.28913792   8.45971515   8.54227144
   8.6906456    8.61856507   8.83489887   8.66309658   8.94837987
   9.20890222   8.9614749    8.92608294   9.13231416   9.55889896
   9.61488451   9.54252979   9.42015491   9.90952569  10.00659591
  10.02504265  10.07330937   9.93489915  10.0892334   10.36509991]
y=  [  1.6635012    2.0214592    2.10816052   2.26016496   1.96287558
   2.9554635    3.02881887   3.33565296   2.75465779   3.4250107
   3.39670148   3.39377767   3.78503343   4.38293049   4.32963586
   4.03925039   4.73691868   4.30098399   4.8416329    4.78175957
   4.99765787   5.31746817   5.76844671   5.93723749   5.72811642
   6.70973615   6.68143367   6.57482731   7.17737603   7.54863252
   7.30221419   7.3202573    7.78023884   7.91133365   8.2765417
   8.69203281   8.78219865   8.45897546   8.89094715   8.81719921
   8.87106971   9.66192562   9.4020625    9.85990783   9.60359778
  10.07386266  10.6957995   10.66721916  11.18256285  10.57431836
  11.46744716  10.94398916  11.26445259  12.09754828  12.11988037
  12.121557    12.17613693  12.43750193  13.00912372  12.86407194
  13.24640866  12.76120085  13.11723062  14.07841099  14.19821707
  14.27289001  14.30624942  14.63060835  14.2770918   15.0744923
  14.45261619  15.11897313  15.2378667   15.27203124  15.32491892
  16.01095271  15.71250558  16.29488506  16.70618934  16.56555394
  16.42379457  17.18144744  17.13813976  17.69613625  17.37763019
  17.90942839  17.90343733  18.01951169  18.35727914  18.16841269
  18.61813748  18.66062754  18.81217983  19.44995194  19.7213867
  19.71966726  19.78961904  19.64385088  20.69719809  20.07974319]
```

3a) Find Expected value of x and the expected value of y

In [9]:

```
Ex = x.mean()
print(Ex)
Ey = y.mean()
print(Ey)
```

```
5.78253254159
11.0129816833
```

3b) Find variance of distributions of x and y

In [10]:

```
Vx = np.var(x)
Vx
```

Out[10]:

```
7.0333275294758497
```

In [11]:

```
Vy = np.var(y)
Vy
```

Out[11]:

```
30.113903575509635
```

3c) Find co-variance of x and y.

In [12]:

```
Cov_xy = np.cov(x,y)
print("The co variance is: ", Cov_xy[0,1])
```

```
The co variance is:  14.6577438328
```

3d) Assuming that number of dollars spent in car fuel is only dependant on the miles driven, by a linear relationship.
Write code that uses a linear predictor to calculate a predicted value of y for each x ie y_predicted = f(x) = y0+mx.

In [13]:

```
m = Cov_xy[1,0]/Vx
c = Ey - m*Ex

print ("The predicted model is given by Y_pred = X * ", m, " + ", c)
```

```
The predicted model is given by Y_pred = X *  2.08404112724  +  -1.0380539
5295
```

3e) Predict y for each value in x, pur the error into an array called y_error

In [19]:

```
y_error = []
for i in range(len(x)):
    y_pred = m*x[i] + c
    error = y_pred - y[i]
    y_error.append(error)

y_error = np.array(y_error)
print("The array y_error is: ", y_error)
```

```
The array y_error is:  [  1.05314309e-01  -7.21703366e-01   1.13662110e-02
  -6.30957167e-02
  -7.03682516e-02  -5.53276173e-01  -3.28116980e-01  -6.17351628e-01
   1.63719369e-01  -2.92897867e-01   4.30238098e-01   2.03594191e-01
  -1.35776473e-01  -4.97052216e-01  -1.61570667e-01   8.02339188e-01
  -1.80312658e-01   2.50973643e-01   5.70653214e-01   3.42777034e-01
   4.04970232e-01   4.26856544e-01   3.11373081e-01   1.28920801e-01
   5.41992427e-01   6.65439261e-02  -1.15626927e-01   2.48426563e-01
  -7.88562495e-01  -2.27749619e-01   3.21646464e-01   1.04741121e-01
  -4.29669229e-01  -8.86824064e-02  -6.47120628e-01  -8.34662873e-01
  -8.06344483e-01  -1.48788597e-01   4.29423037e-02   2.44319388e-01
   4.42282956e-01   1.11072540e-01   5.96590507e-01   1.91174825e-01
  -8.09596391e-02   3.00316025e-01  -3.37218195e-01   5.99195710e-02
  -7.65834652e-01   6.78364995e-01  -6.56707732e-01   2.77886742e-02
   8.94778085e-02  -2.53399759e-01  -7.26168620e-01  -5.97954474e-01
   4.21485798e-01   6.89043727e-02  -6.62014456e-04   3.07070891e-01
  -7.25444679e-01   6.12697206e-01   4.59073099e-01  -7.38552302e-01
   4.62253673e-03  -2.96004972e-01  -5.38520575e-01  -7.58764193e-02
   5.01835004e-01  -2.37480633e-01   5.54718230e-01   2.11799323e-01
   4.69574546e-01  -3.81467141e-02   7.76394438e-01  -3.89885301e-01
   7.67496225e-01  -5.80346879e-02  -1.13849001e-01   1.98837099e-01
   6.49814339e-01  -2.58057329e-01   2.36098875e-01  -6.79940645e-01
   2.33107534e-01   2.44248626e-01  -2.65409018e-01  -4.55241691e-01
  -3.63214807e-01   7.14671927e-01   3.81623320e-01   1.88343033e-01
  -2.18243517e-01   1.63853198e-01   9.47167582e-02   1.34879969e-01
   1.65518016e-01   2.28335961e-02  -7.08874698e-01   4.83497362e-01]
```

3f) Write code that calculates the root mean square error(RMSE), that is root of average of y-error squared

In [20]:

```
y_err_sq = [i**2 for i in y_error]
y_err_sq = np.array(y_err_sq)
avg_y = y_err_sq.mean()
rmse = avg_y**0.5
print("The root mean square is : ", rmse)
```

```
The root mean square is :  0.421318661997
```