Homework 03

NAME: Grant Pemberton

STUDENT ID: 3034347047

Numpy Introduction

1a) Create two numpy arrays (a and b). a should be all integers between 25-34 (inclusive), and b should be ten evenly spaced numbers between 1-6. Print all the results below:

i) Cube (i.e. raise to the power of 3) all the elements in both arrays (element-wise)
ii) Add both the cubed 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 [25]:
```
import numpy as np

#a is a numpy array with integers 25-34 inclusive

a = np.arange(25,35)
print ("array a: ", a)

#b is an array of ten evenly spaced numbers between 1 and 6

b = np.linspace(1,6,num=10)
print ("array b: ", b)

#then print
#i) element wise cube of both

cubeA = np.power(a,3)
print ("cube of a: ", cubeA)

cubeB = np.power(b,3)
print ("cube of b: ", cubeB)

#ii) add both cubed arrays (element wise)

sumCube = cubeA + cubeB
print ("element added cubed arrays: ", sumCube)


#iii) sum the elements of even indicies in the new added array

evenSumCube = sumCube[::2]
print ("sum of even elements in cube array: ", (evenSumCube))

#vi) take the square root of array (iii)

sqrtSumCube = np.power(sumCube, .5)
print ("square root of the added cube array : ", sqrtSumCube)
```

```
array a:  [25 26 27 28 29 30 31 32 33 34]
array b:  [1.         1.55555556 2.11111111 2.66666667 3.22222222 3.777
77778
 4.33333333 4.88888889 5.44444444 6.        ]
cube of a:  [15625 17576 19683 21952 24389 27000 29791 32768 35937 3930
4]
cube of b:  [  1.           3.76406036   9.40877915  18.96296296  33.45
541838
  53.91495199  81.37037037 116.85048011 161.38408779 216.        ]
element added cubed arrays:  [15626.         17579.76406036 19692.40877
915 21970.96296296
 24422.45541838 27053.91495199 29872.37037037 32884.85048011
 36098.38408779 39520.        ]
sum of even elements in cube array:  [15626.         19692.40877915 244
22.45541838 29872.37037037
 36098.38408779]
square root of the added cube array :  [125.00399994 132.58870261 140.3
2964327 148.22605359 156.27685503
 164.48074341 172.83625306 181.34180566 189.99574755 198.79637824]
```

1b) Append b to a, reshape the appended array so that it is a 4x5, 2d array and store the results in a variable called m. Print m.

In [27]:
```
temp = np.concatenate([a, b], axis = 0) #append b to a

print (temp)

m = temp.reshape(4,5) #reshape the concatenation to a 4x5 matrix

print (m)
```

```
[25.          26.          27.          28.          29.          30.
 31.          32.          33.          34.           1.           1.5555555
6
  2.11111111  2.66666667  3.22222222  3.77777778  4.33333333  4.8888888
9
  5.44444444  6.        ]
[[25.          26.          27.          28.          29.        ]
 [30.          31.          32.          33.          34.        ]
 [ 1.          1.55555556  2.11111111  2.66666667  3.22222222]
 [ 3.77777778  4.33333333  4.88888889  5.44444444  6.        ]]
```

1c) Extract the third and the fourth column of the m matrix. Store the resulting 4x2 matrix in a new variable called m2. Print m2.

In [29]:
```
m2 = m[:,3:] # all rows, all columns 3 and after
print (m2)
```

```
[[28.         29.        ]
 [33.         34.        ]
 [ 2.66666667  3.22222222]
 [ 5.44444444  6.        ]]
```

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 [34]:
```
#m3 = m2.dot(m) #4x2 dot 4x5
#this doesn't work...I think that it will only work for square matricies


#transpose m2
m3 = np.matmul((m2.T),m)


print (m3)
```

```
[[1713.2345679  1778.74074074 1844.24691358 1909.75308642 1975.2592592
6]
 [1770.88888889 1839.01234568 1907.13580247 1975.25925926 2043.3827160
5]]
```

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

In [35]:
```
m3 = np.around(m3, decimals=3)
print (m3)
```

```
[[1713.235 1778.741 1844.247 1909.753 1975.259]
 [1770.889 1839.012 1907.136 1975.259 2043.383]]
```

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

In [52]:
```
m3 = m3.reshape(1,10)


m3 = np.sort(m3)

m3 = m3.reshape(2,5)


print (m3)
```

```
[[1713.235 1770.889 1778.741 1839.012 1844.247]
 [1907.136 1909.753 1975.259 1975.259 2043.383]]
```

NumPy and Masks

2a) create an array called 'f' where the values are cosine(x) for x from 0 to pi with 50 equally spaced 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 [42]:
```
f = np.linspace(0,np.pi,50)

f = np.cos(f)

print (f)

mask = (f>=1/2)

print (mask)

fWithMask = f[mask]

print (fWithMask)
```

```
[ 1.          0.99794539  0.99179001  0.98155916  0.96729486  0.9490557
5
  0.92691676  0.90096887  0.8713187   0.8380881   0.80141362  0.7614459
6
  0.71834935  0.67230089  0.6234898   0.57211666  0.51839257  0.4625382
9
  0.40478334  0.34536505  0.28452759  0.22252093  0.1595999   0.0960230
3
  0.03205158 -0.03205158 -0.09602303 -0.1595999  -0.22252093 -0.2845275
9
 -0.34536505 -0.40478334 -0.46253829 -0.51839257 -0.57211666 -0.6234898
 -0.67230089 -0.71834935 -0.76144596 -0.80141362 -0.8380881  -0.8713187
 -0.90096887 -0.92691676 -0.94905575 -0.96729486 -0.98155916 -0.9917900
1
 -0.99794539 -1.        ]
[ True  True  True  True  True  True  True  True  True  True  True  Tru
e
  True  True  True  True  True False False False False False False Fals
e
 False False False False False False False False False False False Fals
e
 False False False False False False False False False False False Fals
e
 False False]
[1.         0.99794539 0.99179001 0.98155916 0.96729486 0.94905575
 0.92691676 0.90096887 0.8713187  0.8380881  0.80141362 0.76144596
 0.71834935 0.67230089 0.6234898  0.57211666 0.51839257]
```

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 [44]:
```
# 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  1.65
075498
  1.79399331  1.80243817  1.89844195  2.00100023  2.3344038   2.2242487
2
  2.24914511  2.36268477  2.49808849  2.8212704   2.68452475  2.6822942
7
  3.09511169  2.95703884  3.09047742  3.2544361   3.41541904  3.4088637
5
  3.50672677  3.74960644  3.64861355  3.7721462   3.56368566  4.0109270
1
  4.15630694  4.06088549  4.02517179  4.25169402  4.15897504  4.2683533
3
  4.32520644  4.48563164  4.78490721  4.84614839  4.96698768  5.1875425
9
  5.29582013  5.32097781  5.0674106   5.47601124  5.46852704  5.6453745
2
  5.49642807  5.89755027  5.68548923  5.76276141  5.94613234  6.1813571
3
  5.96522091  6.0275473   6.54290191  6.4991329   6.74003765  6.8180980
7
  6.50611821  6.91538752  7.01250925  6.89905417  7.31314433  7.2047229
7
  7.1043621   7.48199528  7.58957227  7.61744354  7.6991707   7.8543682
2
  8.03510784  7.80787781  8.22410224  7.99366248  8.40581097  8.2891379
2
  8.45971515  8.54227144  8.6906456   8.61856507  8.83489887  8.6630965
8
  8.94837987  9.20890222  8.9614749   8.92608294  9.13231416  9.5588989
6
  9.61488451  9.54252979  9.42015491  9.90952569 10.00659591 10.0250426
5
 10.07330937  9.93489915 10.0892334  10.36509991]
y=  [ 1.6635012   2.0214592   2.10816052  2.26016496  1.96287558  2.955
4635
  3.02881887  3.33565296  2.75465779  3.4250107   3.39670148  3.3937776
7
  3.78503343  4.38293049  4.32963586  4.03925039  4.73691868  4.3009839
9
  4.8416329   4.78175957  4.99765787  5.31746817  5.76844671  5.9372374
9
  5.72811642  6.70973615  6.68143367  6.57482731  7.17737603  7.5486325
2
  7.30221419  7.3202573   7.78023884  7.91133365  8.2765417   8.6920328
1
  8.78219865  8.45897546  8.89094715  8.81719921  8.87106971  9.6619256
2
  9.4020625   9.85990783  9.60359778 10.07386266 10.6957995  10.6672191
6
 11.18256285 10.57431836 11.46744716 10.94398916 11.26445259 12.0975482
8
 12.11988037 12.121557   12.17613693 12.43750193 13.00912372 12.8640719
4
 13.24640866 12.76120085 13.11723062 14.07841099 14.19821707 14.2728900
1
 14.30624942 14.63060835 14.2770918  15.0744923  14.45261619 15.1189731
3
```

```
 15.2378667  15.27203124 15.32491892 16.01095271 15.71250558 16.2948850
6
 16.70618934 16.56555394 16.42379457 17.18144744 17.13813976 17.6961362
5
 17.37763019 17.90942839 17.90343733 18.01951169 18.35727914 18.1684126
9
 18.61813748 18.66062754 18.81217983 19.44995194 19.7213867  19.7196672
6
 19.78961904 19.64385088 20.69719809 20.07974319]
```

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

In [45]:

```
avX = np.average(x)
avY = np.average(y)

print ("E[x] = ", avX)
print ("E[y] = ", avY)
```

```
E[x] =  5.782532541587923
E[y] =  11.012981683344968
```

3b) Find variance of distributions of x and y

In [46]:

```
varX = np.var(x)

print ("var(x) = ", varX)
```

```
var(x) =  7.03332752947585
```

In [47]:

```
varY = np.var(y)

print ("var(y) = ", varY)
```

```
var(y) =  30.113903575509635
```

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

In [50]:

```
EofXY = np.average(np.multiply(x, y))

#print ("E[XY] = ", EofXY)

covXY = EofXY - (avX*avY)

print (covXY)
```

```
14.511166394475424
```

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 [57]:
```
def linear_predictor_for_3d(input_array):
    slope = covXY/varX
    yInt = avY - (slope*avX)

    y_predicted = (slope*(input_array))+yInt

    return y_predicted
```

In [58]:
```
linear_predictor_for_3d(x)
```

Out[58]:
```
array([ 1.86125717,  1.39688809,  2.20846128,  2.28522836,  1.98371207,
        2.48829527,  2.78382468,  2.80124813,  2.9993232 ,  3.21092152,
        3.8988    ,  3.67152796,  3.7228942 ,  3.9571493 ,  4.23651436,
        4.9033035 ,  4.62116978,  4.61656787,  5.46829307,  5.18342105,
        5.45873164,  5.79701128,  6.12915141,  6.11562653,  6.31753758,
        6.81864709,  6.61027849,  6.86515115,  6.43505522,  7.35780389,
        7.65775187,  7.46087825,  7.38719373,  7.85455455,  7.66325667,
        7.88892606,  8.00622544,  8.33721481,  8.95468038,  9.08103323,
        9.33034895,  9.78539799, 10.00879629, 10.06070164,  9.53754157,
       10.38056671, 10.36512531, 10.72999716, 10.42269073, 11.25028634,
       10.81276185, 10.97218988, 11.35052091, 11.83583685, 11.38990445,
       11.51849632, 12.58177632, 12.49147206, 12.98850691, 13.14956122,
       12.50588416, 13.35028889, 13.5506705 , 13.31658991, 14.17094102,
       13.947246  , 13.74018137, 14.51931443, 14.74126735, 14.79877137,
       14.96739089, 15.28759454, 15.66049665, 15.1916755 , 16.05043004,
       15.57498655, 16.42533161, 16.18461169, 16.53654675, 16.70687695,
       17.01300263, 16.86428603, 17.31062607, 16.95616347, 17.54476017,
       18.08227006, 17.57177784, 17.49875711, 17.92425351, 18.80438359,
       18.91989301, 18.77061069, 18.51812677, 19.5277969 , 19.72807224,
       19.76613158, 19.8657155 , 19.58014745, 19.89856998, 20.4677379
7])
```

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

In [59]:
```
predicted_y = linear_predictor_for_3d(x)

y_error = predicted_y - y

print (y_error)
```

```
[ 0.19775597 -0.62457111  0.10030076  0.02506341  0.02083649 -0.4671682
3
 -0.24499418 -0.53440482  0.24466541 -0.21408918  0.50209852  0.2777502
9
 -0.06213923 -0.42578118 -0.0931215   0.86405311 -0.1157489   0.3155838
8
  0.62666017  0.40166149  0.46107377  0.47954311  0.3607047   0.1783890
4
  0.58942116  0.10891094 -0.07115518  0.29032384 -0.74232081 -0.1908286
3
  0.35553767  0.14062095 -0.39304511 -0.0567791  -0.61328502 -0.8031067
6
 -0.77597321 -0.12176065  0.06373323  0.26383402  0.45927925  0.1234723
8
  0.60673379  0.20079382 -0.0660562   0.30670405 -0.33067419  0.062778
 -0.75987212  0.67596798 -0.65468531  0.02820071  0.08606832 -0.2617114
3
 -0.72997592 -0.60306068  0.40563939  0.05397013 -0.02061681  0.2854892
8
 -0.7405245   0.58908804  0.43343988 -0.76182107 -0.02727604 -0.3256440
1
 -0.56606805 -0.11129392  0.46417555 -0.27572093  0.5147747   0.1686214
2
  0.42262995 -0.08035574  0.72551112 -0.43596616  0.71282602 -0.1102733
7
 -0.16964259  0.14132301  0.58920807 -0.31716141  0.17248631 -0.7399727
8
  0.16712997  0.17284167 -0.33165948 -0.52075457 -0.43302563  0.6359709
  0.30175553  0.10998314 -0.29405306  0.07784496  0.00668554  0.0464643
1
  0.07609646 -0.06370343 -0.79862812  0.38799477]
```

In [61]:
```
print (np.average(y_error))
```

```
1.27675647831893e-15
```

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

In [62]:
```
def RMSE_HW3 (actual_y, calculated_y):
    return np.sqrt(np.mean(np.square(calculated_y - actual_y)))
```

In [63]:
```
RMSE_HW3(y, predicted_y)
```

Out[63]:
```
0.41767772366856115
```