Homework 03

NAME: Rosetta Wang

STUDENT ID: 3034171690

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 [1]: import numpy as np
        a = np.array(range(25, 35))
        b = np.linspace(1.0, 6.0, num=10)
        print(a)
        print(b)
        a **= 3
        b **= 3
        print(a)
        print(b)
```

```
[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.        ]
[15625 17576 19683 21952 24389 27000 29791 32768 35937 39304]
[  1.           3.76406036   9.40877915  18.96296296  33.45541838
  53.91495199  81.37037037 116.85048011 161.38408779 216.        ]
```

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 [17]:

```
m = np.array([a, b], np.int32).reshape(4, 5)
print(m)
```

```
[[15625 17576 19683 21952 24389]
 [27000 29791 32768 35937 39304]
 [    1     3     9    18    33]
 [   53    81   116   161   216]]
```

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 [13]:

```
# I assume you mean to store a 2x4 matrix because you can't take the dot pro
m2 = np.column_stack([m[:,2], m[:,3]])
print(m2)
m2 = np.array([m[:,2], m[:,3]])
print(m2)
```

```
[[19683 21952]
 [32768 35937]
 [    9    18]
 [  116   161]]
[[19683 32768     9   116]
 [21952 35937    18   161]]
```

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 [14]:

```
m3 = np.dot(m2, m, out = None)
print(m3)
```

```
[[1192289032 1322149319 1461175850 1609683670 1767987512]
 [1313307551 1456440614 1609683670 1773384518 1947890546]]
```

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

In [16]:

```
m3 = np.around(m3, decimals=3)
print(m3)
```

```
[[1192289032 1322149319 1461175850 1609683670 1767987512]
 [1313307551 1456440614 1609683670 1773384518 1947890546]]
```

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 [18]:

```
print(np.sort(m3))
```

```
[[1192289032 1322149319 1461175850 1609683670 1767987512]
 [1313307551 1456440614 1609683670 1773384518 1947890546]]
```

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 [21]:

```
f = np.cos(np.linspace(0, np.pi, num=50))
print(f)
mask = f>=1/2
print(f[mask])
```

```
[ 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  0.46253829
  0.40478334  0.34536505  0.28452759  0.22252093  0.1595999   0.09602303
  0.03205158 -0.03205158 -0.09602303 -0.1595999  -0.22252093 -0.28452759
 -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 -0.99179001
 -0.99794539 -1.        ]
[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 [29]:

```
# 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  1.6507
5498
  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  2.95546
35
  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 [31]:

```
print(np.mean(x), np.mean(y))
```

```
5.782532541587923 11.012981683344968
```

3b) Find variance of distributions of x and y

In [33]:
```
np.var(x)
```

Out[33]:
```
7.03332752947585
```

In [34]:
```
np.var(y)
```

Out[34]:
```
30.113903575509635
```

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

In [36]:
```
np.cov(x,y)[0][1]
```

Out[36]:
```
14.657743832803439
```

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 [38]:
```
X = np.column_stack([np.ones(len(x)),x])
W = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)
```

In [39]:
```
y_predicted = lambda x: W[0]+W[1]*x
```

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

In [44]:
```
y_error = y_predicted(x) - x
print(y_error)
```

```
[ 0.5144174   0.27512051  0.6933373   0.73289662  0.5775204   0.83754028
  0.98983137  0.99880996  1.10088125  1.20992128  1.5643962   1.44727924
  1.47374909  1.59446454  1.73842587  2.08203311  1.93664503  1.9342736
  2.37318137  2.22638221  2.36825421  2.54257519  2.71373237  2.70676278
  2.81081081  3.06904065  2.96166494  3.09300495  2.87136955  3.34687688
  3.50144492  3.39999276  3.36202194  3.60286053  3.50428164  3.62057272
  3.68101899  3.85158317  4.16977317  4.23488484  4.36336127  4.5978554
  4.71297616  4.73972383  4.47013098  4.90455547  4.89659827  5.08462264
  4.92626266  5.35273607  5.12727262  5.20942846  5.40438857  5.65447973
  5.42468354  5.49094901  6.0388744   5.99233916  6.24846926  6.33146315
  5.99976594  6.43490137  6.53816125  6.41753574  6.85779669  6.74252303
  6.63581927  7.03731915  7.15169508  7.18132783  7.2682202   7.43322632
  7.62538881  7.38379768  7.8263278   7.58132407  8.01952064  7.89547377
  8.0768316   8.16460551  8.32235703  8.24572096  8.4757272   8.29306689
  8.59638029  8.87336784  8.61030294  8.57267417  8.79193935  9.24548463
  9.3050085   9.2280809   9.09797185  9.61827121  9.72147633  9.74108893
  9.79240614  9.64524829  9.80933658 10.10263805]
```

In [ ]:
```
# I assume you mean "put the error"
```

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

```
In [47]: np.sqrt(np.mean(y_error ** 2))

Out[47]: 5.94205703941763

In [ ]: 
```