
# 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 [13]:
```
import numpy as np

a = np.arange(10,20)
print("i:")
print(a)
print()
b = np.linspace(1,7,10)
print(b)

print("\nii:")

a = np.square(a)
print(a)
print()
b = np.square(b)
print(b)
print()
c = (a+b)
print(c)

print("\niii:")

print(c[::2])
sum_c = c[::2]
sum_c = sum_c.sum()
print(sum_c)

print("\niv:")
sqrt_c = np.sqrt(c)
print(sqrt_c)
```

```
i:
[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.        ]

ii:
[100 121 144 169 196 225 256 289 324 361]

[  1.           2.77777778   5.44444444   9.          13.44444444
  18.77777778  25.          32.11111111  40.11111111  49.        ]

[ 101.          123.77777778  149.44444444  178.          209.44444444
  243.77777778  281.          321.11111111  364.11111111  410.        ]

iii:
[ 101.          149.44444444  209.44444444  281.          364.11111111]
1105.0

iv:
[ 10.04987562  11.12554618  12.22474721  13.34166406  14.47219556
  15.61338457  16.76305461  17.91957341  19.08169571  20.24845673]
```

**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 [14]:
```
m = np.append(a,b)
print(m)
m = np.reshape(np.ravel(m), (5,4))
print(m)
```

```
[ 100.          121.          144.          169.          196.          225.
  256.          289.          324.          361.            1.
    2.77777778    5.44444444    9.           13.44444444   18.77777778
   25.           32.11111111   40.11111111   49.        ]
[[ 100.          121.          144.          169.        ]
 [ 196.          225.          256.          289.        ]
 [ 324.          361.            1.            2.77777778]
 [   5.44444444    9.           13.44444444   18.77777778]
 [  25.           32.11111111   40.11111111   49.        ]]
```

**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 [15]:
```
m2 = m[:, [1, 2]]
print(m2)
```

```
[[ 121.          144.        ]
 [ 225.          256.        ]
 [ 361.            1.        ]
 [   9.           13.44444444]
 [  32.11111111   40.11111111]]
```

**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 [26]:
```
m3 = np.dot(m.T,m2)
m3
```

Out[26]:
```
array([[ 174015.77777778,   65975.97530864],
       [ 196699.12345679,   76794.01234568],
       [  76794.01234568,   88062.65432099],
       [  88219.22222222,  100540.67901235]])
```

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

In [27]:
```
m3 = np.around(m3,2)
print(m3)
```

```
[[ 174015.78   65975.98]
 [ 196699.12   76794.01]
 [  76794.01   88062.65]
 [  88219.22  100540.68]]
```

**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 [28]:
```
m4 = np.sort(m3, axis=None)
m4 = np.reshape(np.ravel(m4[::-1]),(4,2))
m4
```

Out[28]:
```
array([[ 196699.12,  174015.78],
       [ 100540.68,   88219.22],
       [  88062.65,   76794.01],
       [  76794.01,   65975.98]])
```

## 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 [8]:
```
f = np.array(np.sin(np.linspace(0,np.pi,100)))
print(f)

print()

mask = f>=(1/2)
print(mask)
```

```
print()

true_array = f[mask]
print(true_array)
```

```
[  0.00000000e+00   3.17279335e-02   6.34239197e-02   9.50560433e-02
   1.26592454e-01   1.58001396e-01   1.89251244e-01   2.20310533e-01
   2.51147987e-01   2.81732557e-01   3.12033446e-01   3.42020143e-01
   3.71662456e-01   4.00930535e-01   4.29794912e-01   4.58226522e-01
   4.86196736e-01   5.13677392e-01   5.40640817e-01   5.67059864e-01
   5.92907929e-01   6.18158986e-01   6.42787610e-01   6.66769001e-01
   6.90079011e-01   7.12694171e-01   7.34591709e-01   7.55749574e-01
   7.76146464e-01   7.95761841e-01   8.14575952e-01   8.32569855e-01
   8.49725430e-01   8.66025404e-01   8.81453363e-01   8.95993774e-01
   9.09631995e-01   9.22354294e-01   9.34147860e-01   9.45000819e-01
   9.54902241e-01   9.63842159e-01   9.71811568e-01   9.78802446e-01
   9.84807753e-01   9.89821442e-01   9.93838464e-01   9.96854776e-01
   9.98867339e-01   9.99874128e-01   9.99874128e-01   9.98867339e-01
   9.96854776e-01   9.93838464e-01   9.89821442e-01   9.84807753e-01
   9.78802446e-01   9.71811568e-01   9.63842159e-01   9.54902241e-01
   9.45000819e-01   9.34147860e-01   9.22354294e-01   9.09631995e-01
   8.95993774e-01   8.81453363e-01   8.66025404e-01   8.49725430e-01
   8.32569855e-01   8.14575952e-01   7.95761841e-01   7.76146464e-01
   7.55749574e-01   7.34591709e-01   7.12694171e-01   6.90079011e-01
   6.66769001e-01   6.42787610e-01   6.18158986e-01   5.92907929e-01
   5.67059864e-01   5.40640817e-01   5.13677392e-01   4.86196736e-01
   4.58226522e-01   4.29794912e-01   4.00930535e-01   3.71662456e-01
   3.42020143e-01   3.12033446e-01   2.81732557e-01   2.51147987e-01
   2.20310533e-01   1.89251244e-01   1.58001396e-01   1.26592454e-01
   9.50560433e-02   6.34239197e-02   3.17279335e-02   1.22464680e-16]

[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  True  True  True  True False
 False False False False False False False False False False False False
 False False False False]

[ 0.51367739  0.54064082  0.56705986  0.59290793  0.61815899  0.64278761
  0.666769    0.69007901  0.71269417  0.73459171  0.75574957  0.77614646
  0.79576184  0.81457595  0.83256985  0.84972543  0.8660254   0.88145336
  0.89599377  0.909632    0.92235429  0.93414786  0.94500082  0.95490224
  0.96384216  0.97181157  0.97880245  0.98480775  0.98982144  0.99383846
  0.99685478  0.99886734  0.99987413  0.99987413  0.99886734  0.99685478
  0.99383846  0.98982144  0.98480775  0.97880245  0.97181157  0.96384216
  0.95490224  0.94500082  0.93414786  0.92235429  0.909632    0.89599377
  0.88145336  0.8660254   0.84972543  0.83256985  0.81457595  0.79576184
  0.77614646  0.75574957  0.73459171  0.71269417  0.69007901  0.666769
  0.64278761  0.61815899  0.59290793  0.56705986  0.54064082  0.51367739]
```

## 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 [32]:
```
# 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 [33]:
```
E_x = np.mean(x)
print(E_x)

print()

E_y = np.mean(y)
print(E_y)
```

```
5.78253254159

11.0129816833
```

**3b) Find variance of distributions of x and y**

In [34]:
```
E_x = np.var(x)
print(E_x)
```

```
7.03332752948
```

In [35]:
```
E_y = np.var(y)
print(E_y)
```

```
30.1139035755
```

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

In [36]:
```
Cov_xy = np.cov(x,y)
print(Cov_xy)
```

```
[[  7.10437124  14.65774383]
 [ 14.65774383  30.41808442]]
```

**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 [37]:
```
x_trp = np.vstack([x, np.ones(len(x))]).T
print(x_trp)
m, c = np.linalg.lstsq(x_trp, y)[0]
print()
print(m,c)
```

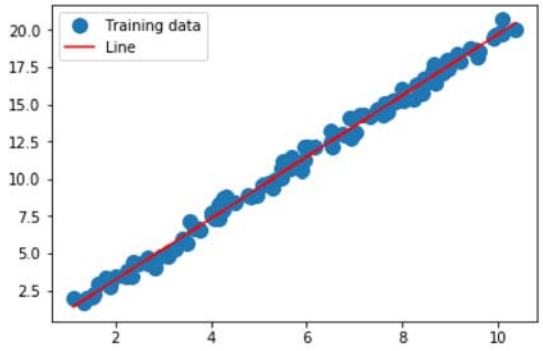
```
[[  1.34683976   1.        ]
 [  1.12176759   1.        ]
 [  1.51512398   1.        ]
 [  1.55233174   1.        ]
 [  1.40619168   1.        ]
 [  1.65075498   1.        ]
 [  1.79399331   1.        ]
 [  1.80243817   1.        ]
 [  1.89844195   1.        ]
 [  2.00100023   1.        ]
 [  2.3344038    1.        ]
 [  2.22424872   1.        ]
 [  2.24914511   1.        ]
 [  2.36268477   1.        ]
 [  2.49808849   1.        ]
 [  2.8212704    1.        ]
 [  2.68452475   1.        ]
 [  2.68229427   1.        ]
 [  3.09511169   1.        ]
 [  2.95703884   1.        ]
 [  3.09047742   1.        ]
 [  3.2544361    1.        ]
 [  3.41541904   1.        ]
 [  3.40886375   1.        ]
 [  3.50672677   1.        ]
 [  3.74960644   1.        ]
 [  3.64861355   1.        ]
 [  3.7721462    1.        ]
 [  3.56368566   1.        ]
 [  4.01092701   1.        ]
 [  4.15630694   1.        ]
 [  4.06088549   1.        ]
 [  4.02517179   1.        ]
 [  4.25169402   1.        ]
 [  4.15897504   1.        ]
 [  4.26835333   1.        ]
 [  4.32520644   1.        ]
 [  4.48563164   1.        ]
 [  4.78490721   1.        ]
 [  4.84614839   1.        ]
 [  4.96698768   1.        ]
 [  5.18754259   1.        ]
 [  5.29582013   1.        ]
 [  5.32097781   1.        ]
 [  5.0674106    1.        ]
 [  5.47601124   1.        ]
 [  5.46852704   1.        ]
 [  5.64537452   1.        ]
 [  5.49642807   1.        ]
 [  5.89755027   1.        ]
 [  5.68548923   1.        ]
 [  5.76276141   1.        ]
 [  5.94613234   1.        ]
 [  6.18135713   1.        ]
 [  5.96522091   1.        ]
 [  6.0275473    1.        ]
 [  6.54290191   1.        ]
 [  6.4991329    1.        ]
 [  6.74003765   1.        ]
 [  6.81809807   1.        ]
 [  6.50611821   1.        ]
 [  6.91538752   1.        ]
 [  7.01250925   1.        ]
 [  6.89905417   1.        ]
 [  7.31314433   1.        ]
 [  7.20472297   1.        ]
 [  7.1043621    1.        ]
 [  7.48199528   1.        ]
 [  7.58957227   1.        ]
 [  7.61744354   1.        ]
 [  7.6991707    1.        ]
 [  7.85436822   1.        ]
 [  8.03510784   1.        ]
 [  7.80787781   1.        ]
 [  8.22410224   1.        ]
 [  7.99366248   1.        ]
 [  8.40581097   1.        ]
 [  8.28913792   1.        ]
 [  8.45971515   1.        ]
```

```
[  8.54227144   1.        ]
[  8.6906456    1.        ]
[  8.61856507   1.        ]
[  8.83489887   1.        ]
[  8.66309658   1.        ]
[  8.94837987   1.        ]
[  9.20890222   1.        ]
[  8.9614749    1.        ]
[  8.92608294   1.        ]
[  9.13231416   1.        ]
[  9.55889896   1.        ]
[  9.61488451   1.        ]
[  9.54252979   1.        ]
[  9.42015491   1.        ]
[  9.90952569   1.        ]
[ 10.00659591   1.        ]
[ 10.02504265   1.        ]
[ 10.07330937   1.        ]
[  9.93489915   1.        ]
[ 10.0892334    1.        ]
[ 10.36509991   1.        ]]

2.06320071597 -0.917543596587
```

In [38]:
```
import matplotlib.pyplot as pplt

pplt.plot(x, y, 'o', label='Training data', markersize=10)
pplt.plot(x, m*x + c, 'r', label='Line')
pplt.legend()
pplt.show()
```




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

In [39]:
```
y_predict = c + m*x
print(y_predict)
```

```
[  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  20.46773797]
```

In [40]:
```
y_error = y-y_predict
print(y_error)
```

```
[-0.19775597  0.62457111 -0.10030076 -0.02506341 -0.02083649  0.46716823
  0.24499418  0.53440482 -0.24466541  0.21408918 -0.50209852 -0.27775029
  0.06213923  0.42578118  0.0931215  -0.86405311  0.1157489  -0.31558388
 -0.62666017 -0.40166149 -0.46107377 -0.47954311 -0.3607047  -0.17838904
```

```
-0.58942116 -0.10891094  0.07115518 -0.29032384  0.74232081  0.19082863
-0.35553767 -0.14062095  0.39304511  0.0567791   0.61328502  0.80310676
 0.77597321  0.12176065 -0.06373323 -0.26383402 -0.45927925 -0.12347238
-0.60673379 -0.20079382  0.0660562  -0.30670405  0.33067419 -0.062778
 0.75987212 -0.67596798  0.65468531 -0.02820071 -0.08606832  0.26171143
 0.72997592  0.60306068 -0.40563939 -0.05397013  0.02061681 -0.28548928
 0.7405245  -0.58908804 -0.43343988  0.76182107  0.02727604  0.32564401
 0.56606805  0.11129392 -0.46417555  0.27572093 -0.5147747  -0.16862142
-0.42262995  0.08035574 -0.72551112  0.43596616 -0.71282602  0.11027337
 0.16964259 -0.14132301 -0.58920807  0.31716141 -0.17248631  0.73997278
-0.16712997 -0.17284167  0.33165948  0.52075457  0.43302563 -0.6359709
-0.30175553 -0.10998314  0.29405306 -0.07784496 -0.00668554 -0.04646431
-0.07609646  0.06370343  0.79862812 -0.38799477]
```

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

In [43]:
```
RMSE=np.sqrt(((y_error**2).sum())/np.size(y_error))
print(RMSE)
```

```
0.417677723669
```

In [ ]:

In [ ]: