ML Assignment 2

By Harshitha MU

Roll no.: J076

In [1]:

```
import numpy as np
import time
import random
```

Properties of Matrix Multiplication

In [2]:

```
A=np.array([[np.random.randint(0,100) for j in range(10)] for i in range(10)])
B=np.array([[np.random.randint(0,100) for j in range(10)] for i in range(10)])
C=np.array([[np.random.randint(0,100) for j in range(10)] for i in range(10)])
```

In [3]:

```
def proof(A,B):
    if A.shape!=B.shape:
        return False
    else:
        for i in range(A.shape[0]):
            for j in range(A.shape[1]):
                if A[i][j]!=B[i][j]:
                    return False
        return True
```

Associative Property

A(BC) = (AB)C

In [4]:

```
a=np.matmul(A,np.matmul(B,C))
b=np.matmul(np.matmul(A,B),C)
if proof(a,b):
    print("Matrices are Associative!")
else:
    print("Matrices are not Associative!")
```

```
Matrices are Associative!
```

Distributive Property

A(B + C) == (AB) + (AC)

In [5]:

```
abc=np.matmul(A,np.add(B,C))
abac=np.add(np.matmul(A,B),np.matmul(A,C))
if proof(abc,abac):
    print("Matrices are Distributive!")
else:
    print("Matrices are not Distributive!")
```

```
Matrices are Distributive!
```

Non-commutative property

AB != BA

In [6]:

```
ab=np.matmul(A,B)
ba=np.matmul(B,A)
if proof(ab,ba):
    print("Matrices are Commutative!")
else:
    print("Matrices are Non-Commutative!")
```

```
Matrices are Non-Commutative!
```

Inverse of matric using numpy:

In [7]:

```
a_inv=np.linalg.inv(A)
print(a_inv)
```

```
[[-5.13352330e-03  3.87356311e-04 -9.53691044e-03  6.09746269e-03
   1.30413324e-02  3.57543243e-03 -5.18533449e-03 -3.42282343e-03
   4.10707661e-03 -4.34448202e-03]
 [-1.23311614e-02  1.63077641e-05  1.83806117e-02 -1.35927767e-02
  -2.65199406e-02  7.88960805e-03  9.63436698e-03 -4.85689867e-03
  -1.69111057e-02  3.31634715e-02]
 [-3.80340198e-03  4.85466659e-03 -8.93138632e-03  5.43535571e-03
  -8.02470912e-06 -4.48258320e-03  3.22295240e-04  7.17410523e-03
  -1.25562221e-03  1.89037170e-03]
 [ 3.51328689e-03 -2.25134957e-03  1.14290539e-02 -7.63366738e-03
  -8.36715663e-03 -1.33551846e-02  8.99071841e-03 -4.66080139e-04
  -6.97237910e-03  1.84890187e-02]
 [ 9.61174563e-03 -6.30954568e-03 -8.14272994e-03  9.04854744e-03
   9.29921972e-03  2.33496711e-03 -8.58362475e-03 -3.94644500e-03
   1.40574177e-02 -1.58383146e-02]
 [ 1.49546449e-02  1.86618337e-03 -1.03531960e-02  6.87503026e-03
   1.08172055e-02 -1.55715324e-03  4.23897036e-04  1.33754290e-03
   1.45456779e-03 -1.81747170e-02]
 [-9.10236727e-03 -8.96235679e-03  7.18448201e-03 -1.52149315e-02
  -1.18549036e-02  9.31113437e-03  4.37400386e-03  5.90407226e-03
   7.15107961e-04  1.31389426e-02]
 [-1.37136359e-02 -3.54587626e-03  1.60096212e-02  1.79823770e-03
  -5.51095177e-03 -5.25609506e-03  4.40116834e-03  2.00183753e-03
  -9.71796599e-03  1.47301673e-02]
 [ 1.31788567e-03  4.97247471e-03  6.28818957e-03  4.48867610e-03
   3.21348105e-03 -2.59167774e-03  4.96717982e-03 -9.02668203e-03
   5.31995492e-03 -1.09557966e-02]
 [ 1.00059907e-02  6.15566765e-03 -1.42586346e-02  3.16673211e-03
   1.40714995e-02  5.07780693e-03 -1.32717404e-02  4.56486912e-03
   1.09342764e-02 -2.18108501e-02]]
```

In [8]:

```
b_inv=np.linalg.inv(B)
print(b_inv)
```

```
[[-7.01763892e-03  8.47264346e-03  5.17212375e-04 -9.79268162e-03
  -3.76386264e-03 -3.01315165e-03  6.28173181e-03  1.01359522e-02
  -2.46158958e-03  3.48962819e-03]
 [ 1.90123585e-02 -3.75770340e-02  1.76826158e-02  4.91998687e-03
   1.03891301e-03 -8.97603321e-04 -1.93498090e-02  1.01614061e-02
   7.61231877e-03 -3.27982112e-03]
 [ 4.02966316e-02 -6.66176229e-02  2.95939153e-02  1.92926147e-02
   2.65071922e-02 -2.21104152e-02 -3.81097839e-02 -2.13828742e-02
   8.06022952e-03  2.40397258e-02]
 [-2.32413694e-02  3.81120764e-02 -1.33145710e-02  6.05635663e-04
```

```
 -1.66830331e-02  1.10896422e-02  2.07228622e-02 -2.61949269e-06
 -3.66137465e-03 -1.40663818e-02]
 [-1.69248332e-02  2.49140354e-02 -7.89385989e-03 -1.17242695e-02
  2.04647848e-03  8.49191500e-03  1.85026097e-02 -1.43208484e-02
 -2.34305443e-03  1.71297105e-03]
 [ 1.58212656e-02 -1.52321836e-02  4.06817193e-03  3.42238972e-03
 -4.94004266e-04 -1.10149908e-02 -7.42374862e-04  3.03835305e-03
  2.96761769e-03 -1.14352776e-04]
 [-2.46778350e-03  4.16417912e-03 -9.09456120e-03  6.87319284e-03
  3.71005657e-03  4.72361549e-04  5.97584749e-03 -2.87229812e-03
 -5.71085530e-03  4.20020000e-03]
 [ 1.91896541e-02 -3.21745944e-02  9.98511291e-03  8.76149084e-03
  1.63077484e-02 -2.03532659e-02 -2.07685613e-02 -6.71965071e-04
  9.01573108e-03  1.31195746e-02]
 [-1.51338636e-02  3.20530755e-02 -8.12102619e-03 -8.84310100e-03
 -1.00469193e-02  1.31474884e-02  1.30745632e-02  1.16184154e-02
 -7.22863349e-03 -1.72639770e-02]
 [-2.17245051e-02  3.75546055e-02 -1.80736204e-02 -1.09768273e-02
 -1.16555118e-02  1.75505894e-02  1.58175612e-02  4.02874077e-03
 -1.89004002e-03 -6.01822882e-03]]
```

In [9]:

```python
c_inv=np.linalg.inv(C)
print(c_inv)
```

```
[[ 0.01701773 -0.00916301 -0.00965099 -0.02084687  0.01278552  0.01721986
  -0.01291315  0.00534465 -0.00732402  0.02173062]
 [ 0.00633813  0.0064285  -0.00690707 -0.018583    0.02282933  0.0074117
  -0.00105287 -0.0083262  -0.01739292  0.0237761 ]
 [ 0.00333575  0.00444986  0.00383473 -0.00934565  0.00653342 -0.00308415
  -0.00470422  0.00160406 -0.00200978  0.00754495]
 [-0.00812212 -0.00471042  0.00806076  0.00486763 -0.00266057 -0.00686953
   0.00826728 -0.00768912  0.01130147 -0.00974008]
 [-0.0106018  -0.0143301   0.00194375  0.03228707 -0.03212923 -0.01752799
   0.0152011   0.00738566  0.02722735 -0.02678255]
 [-0.00126874 -0.01422495 -0.00262482 -0.00066265 -0.01313272  0.00363825
   0.00148975  0.01403641  0.00805623  0.00804478]
 [-0.01110337  0.00564731  0.00422035  0.01977089 -0.02064502 -0.00590279
   0.00566673 -0.0035415   0.01085229 -0.01422449]
 [-0.00213909  0.00628339 -0.00631197  0.00105671  0.00656335 -0.00074434
  -0.0015447   0.00589427  0.00056777 -0.00849244]
 [-0.00204126  0.00276393  0.00886072  0.01447631  0.00126324  0.00263172
  -0.00832675 -0.00662879 -0.00540462 -0.01191712]
 [ 0.0066508   0.02076034  0.0033177  -0.01374157  0.0194213   0.00396351
   0.00258667 -0.01366678 -0.02342594  0.00557933]]
```

Is Numpy faster than traditional looping?

In [11]:

```python
X=np.array([[np.random.randint(0,100) for j in range(1000)] for i in range(1000)])
Y=np.array([[np.random.randint(0,100) for j in range(1000)] for i in range(1000)])
```

Traditional Looping:

In [13]:

```python
start=time.time()
res=np.array([[0 for i in range(1000)] for j in range(1000)])
for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        res[i][j]=X[i][j]+Y[i][j]

loop_time=time.time()-start
print(loop_time)
```

```
2.742208480834961
```

Numpy Array:

In [14]:

```
start_time=time.time()
res_np=np.add(X,Y)

np_time=time.time()-start_time
print(np_time)
```

```
0.004006624221801758
```

In [ ]: