Assignment 2: Numpy Basics

Name: Falguni Gaikwad

Rollno: J077

Properties of Matrix Multiplication

In [3]:

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

In [3]:

```
F=np.array([[24,50,31,72],[25,32,42,57],[3,8,3,21],[4,12,36,10]])
F
```

Out[3]:

```
array([[24, 50, 31, 72],
       [25, 32, 42, 57],
       [ 3,  8,  3, 21],
       [ 4, 12, 36, 10]])
```

In [4]:

```
H=np.array([[1,2,4,7],[5,5,28,11],[4,60,21,21],[32,1,7,28]])
H
```

Out[4]:

```
array([[ 1,  2,  4,  7],
       [ 5,  5, 28, 11],
       [ 4, 60, 21, 21],
       [32,  1,  7, 28]])
```

In [5]:

```
V=np.array([[5,8,2,10],[29,5,28,13],[5,6,52,1],[12,57,2,78]])
V
```

Out[5]:

```
array([[ 5,  8,  2, 10],
       [29,  5, 28, 13],
       [ 5,  6, 52,  1],
       [12, 57,  2, 78]])
```

Property 1: AB!=BA

In [16]:

```
np.dot(F,H), np.dot(H,F)
```

Out[16]:

```
(array([[2702, 2230, 2651, 3385],
        [2177, 2787, 2277, 3005],
        [ 727,  247,  446,  760],
        [ 528, 2238, 1178, 1196]]),
 array([[ 114,  230,  379,  340],
        [ 373,  766,  845, 1343],
```

```
        [1743, 2540, 3463, 4359],
        [ 926, 2024, 2063, 2788]]))
```

Property 2: A(BC)=(AB)C

In [15]:

```
F.dot(H.dot(V)), (F.dot(H)).dot(V)
```

Out[15]:

```
(array([[132055, 241617, 212466, 322691],
        [139153, 216298, 206804, 294668],
        [ 22148,  53047,  33082,  70207],
        [ 87784,  90654, 127368, 128840]]),
 array([[132055, 241617, 212466, 322691],
        [139153, 216298, 206804, 294668],
        [ 22148,  53047,  33082,  70207],
        [ 87784,  90654, 127368, 128840]]))
```

Property 3: A(B+C)= AB+AC

In [11]:

```
a1=np.dot(F,(H+V))
a1
```

Out[11]:

```
array([[5291, 6962, 5855, 9922],
       [4124, 6648, 5521, 8159],
       [1241, 1526,  874, 2535],
       [1196, 3116, 3414, 2208]])
```

In [12]:

```
a2=np.dot(F,H)+np.dot(F,V)
a2
```

Out[12]:

```
array([[5291, 6962, 5855, 9922],
       [4124, 6648, 5521, 8159],
       [1241, 1526,  874, 2535],
       [1196, 3116, 3414, 2208]])
```

Property 4: AI=IA

In [13]:

```
I=np.identity(4)
I
```

Out[13]:

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

In [14]:

```
np.dot(F,I), np.dot(I,F)
```

Out[14]:

```
(array([[24., 50., 31., 72.],
        [25., 32., 42., 57.],
```

```
        [ 3.,  8.,  3., 21.],
        [ 4., 12., 36., 10.]]),
 array([[24., 50., 31., 72.],
        [25., 32., 42., 57.],
        [ 3.,  8.,  3., 21.],
        [ 4., 12., 36., 10.]]))
```

Calculate inverse of a matrix using numpy (inbuilt api and/or manual coding)

In [17]:

```
np.linalg.inv(F)
```

Out[17]:

```
array([[-1.00324652e-02,  7.82581102e-02, -1.44384505e-01,
        -7.06300193e-02],
       [ 6.35267410e-02, -5.39336539e-02, -7.84411844e-02,
         1.47557790e-02],
       [-1.43041961e-02,  6.95681890e-03,  1.55491005e-02,
         3.06832328e-02],
       [-2.07239974e-02,  8.37259257e-03,  9.59064613e-02,
         8.54346181e-05]])
```

Show how numpy is faster than traditional looping

In [4]:

```
mat1=np.random.randint(5,size=(10000,10000))
mat2=np.random.randint(5,size=(10000,10000))
```

In [5]:

```
# Numpy

start=time.time()
ans=np.add(mat1,mat1)
print("Time taken using numpy:")
print(time.time()-start)
```

```
Time taken using numpy:
0.11232900619506836
```

In [6]:

```
# Traditional Looping

start=time.time()
for i in range(len(mat2)):
    for j in range(len(mat2)):
        mat1[i][j]=mat1[i][j]+1
print(time.time()-start)
```

```
78.7838454246521
```

In [ ]:

```

```