Question 1

In [5]:

```
# Some pointers
# Numpy matrices are strictly 2-dimensional, while numpy arrays (ndarrays) are N-dimension
al.
# Matrix objects are a subclass of ndarray, so they inherit all the attributes and methods
of ndarrays.
```

In [14]:

```
# 1.Creating matrix M and its transpose
import numpy as np

m = np.array([[1,1],[2,4],[3,9],[4,16]])

t = m.T

print(f"M:\n{m}\n")
print(f"transpose(M):\n{t}")
```

```
M:
[[ 1  1]
 [ 2  4]
 [ 3  9]
 [ 4 16]]

transpose(M):
[[ 1  2  3  4]
 [ 1  4  9 16]]
```

In [15]:

```
# 2. Finding MMt and MtM

import numpy as np
m = np.array([[1,1],[2,4],[3,9],[4,16]])
t = m.T

MMt = m.dot(t)
MtM = t.dot(m)

print(f"MMt:\n{MMt}\n")
print(f"MtM:\n{MtM}")
```

```
MMt:
[[  2   6  12  20]
 [  6  20  42  72]
 [ 12  42  90 156]
 [ 20  72 156 272]]

MtM:
[[ 30 100]
 [100 354]]
```

In [ ]:

```
# 3. Eigen pairs of MtM
```

In [1]:

```
# 4. Eigen values and vectors of MMt

# The function la.eig returns a tuple (eigvals,eigvecs)
# where eigvals is a 1D NumPy array of complex numbers giving the eigenvalues of , and
# eigvecs is a 2D NumPy array with the corresponding eigenvectors in the columns

import scipy.linalg as la
import numpy as np
m = np.array([[1,1],[2,4],[3,9],[4,16]])
t = m.T

MMt = m.dot(t)

eigen = la.eig(MMt)
print(f"Eigen Values:\n{eigen[0].real}\n")


eigenvecs = eigen[1]
print(f"Eigen Vectors:\n")
for i in range(len(eigenvecs)):
    print(eigenvecs[:,i])
```

```
Eigen Values:
[ 3.82378570e+02  1.62142978e+00 -7.45088994e-15  1.29844423e-16]

Eigen Vectors:

[-0.06315773 -0.22470839 -0.484652   -0.84298854]
[ 0.54109638  0.65339536  0.33689693 -0.40839891]
[-0.45663653  0.72081202 -0.50444616  0.13208775]
[ 0.42599675  0.27225124 -0.7889984   0.34912399]
```

Question 2

In [20]:

```
# 1. Create Matrix M and find its transpose

import numpy as np

m = np.array([[1,2,3],[3,4,5],[5,4,3],[0,2,4], [1,3,5]])

t = m.T

print(f"M:\n{m}\n")
print(f"transpose(M):\n{t}")
```

```
M:
[[1 2 3]
 [3 4 5]
 [5 4 3]
 [0 2 4]
 [1 3 5]]

transpose(M):
[[1 3 5 0 1]
 [2 4 4 2 3]
 [3 5 3 4 5]]
```

In [21]:

```
# 2. Compute matrices MtM and MMt

import numpy as np

m = np.array([[1,2,3],[3,4,5],[5,4,3],[0,2,4], [1,3,5]])
t = m.T

MMt = m.dot(t)
MtM = t.dot(m)

print(f"MMt:\n{MMt}\n")
print(f"MtM:\n{MtM}")
```

```
MMt:
[[14 26 22 16 22]
 [26 50 46 28 40]
 [22 46 50 20 32]
 [16 28 20 20 26]
 [22 40 32 26 35]]

MtM:
[[36 37 38]
 [37 49 61]
 [38 61 84]]
```

In [32]:

```
# 3. Find eigen values and vectors

import numpy as np
import scipy.linalg as la

m = np.array([[1,2,3],[3,4,5],[5,4,3],[0,2,4], [1,3,5]])
t = m.T

MMt = m.dot(t)
MtM = t.dot(m)

eigen1 = la.eig(MMt)
print("For MMt:\n")
print(f"Eigen Values:\n{eigen1[0].real}\n")  #.real is to get rid of the complex nature
eigenvecs = eigen1[1]
print(f"Eigen Vectors:\n")
for i in range(len(eigenvecs)):
    print(f">>  {eigenvecs[:,i]}")

eigen2 = la.eig(MtM)
print("\n\nFor MtM:\n")
print(f"Eigen Values:\n{eigen2[0].real}\n")
eigenvecs = eigen2[1]
print(f"Eigen Vectors:\n")
for i in range(len(eigenvecs)):
    print(f">>  {eigenvecs[:,i]}")
```

```
For MMt:

Eigen Values:
[ 1.53566996e+02 -1.03322028e-14  1.54330035e+01  2.54653026e-15
 -3.61063094e-15]

Eigen Vectors:

>>  [0.29769568 0.57050856 0.52074297 0.32257847 0.45898491]
>>  [ 0.94131607 -0.17481584 -0.04034212 -0.18826321 -0.21515796]
>>  [-0.15906393  0.0332003   0.73585663 -0.5103921  -0.41425998]
>>  [ 0.12508859 -0.45318832  0.32553276  0.72000366 -0.39318742]
>>  [ 0.07520849 -0.07287035 -0.10566284 -0.72571726  0.67171677]


For MtM:

Eigen Values:
[1.53566996e+02 1.54330035e+01 4.80589926e-15]

Eigen Vectors:

>>  [-0.40928285 -0.56345932 -0.7176358 ]
>>  [-0.81597848 -0.12588456  0.56420935]
>>  [ 0.40824829 -0.81649658  0.40824829]
```

In [ ]:

```
# COMPUTATION OF SVD : STEPS
# Anxp= Unxn Snxp VTpxp
```

In [8]:

```
import numpy as np
import scipy.linalg as la

m = np.array([[1,2,3],[3,4,5],[5,4,3],[0,2,4], [1,3,5]])
#m = np.array([[2,4],[1,3],[0,0],[0,0]])
#m = np.array([[1,0,1,0], [0,1,0,1]])
t = m.T

MMt = m.dot(t)

MtM = t.dot(m)

# Step 1
e1 = la.eig(MMt)
e2 = la.eig(MtM)
vals1 = []
vals2 = []
ind1 =[]
ind2 =[]
for i in range(len(e1[0].real)):
    a = format(e1[0].real[i], 'f')
    if '0.0' not in a:
        vals1.append(e1[0].real[i])
        ind1.append(i)
for i in range(len(e2[0].real)):
    a = format(e2[0].real[i], 'f')
    if '0.0' not in a:
        vals2.append(e2[0].real[i])
        ind2.append(i)

U = e1[1][:,ind1]
V = e2[1].T[:,ind2]
singulars = [i**0.5 for i in vals1]
'''
print(vals1,"\n\n")
print(ind1)
print(ind2)
print(U)
print(V)
print(singulars)
'''
S = np.diagflat(singulars)

print("U: \n", U)
print("\nS: \n",S)
print("\nV: \n",V)
print("\ntranspose(V):\n", V.T)
```

```
U:
 [[ 0.29769568 -0.15906393]
 [ 0.57050856  0.0332003 ]
 [ 0.52074297  0.73585663]
 [ 0.32257847 -0.5103921 ]
 [ 0.45898491 -0.41425998]]

S:
 [[12.39221516  0.        ]
 [ 0.          3.92848616]]

V:
 [[-0.40928285 -0.56345932]
 [-0.81597848 -0.12588456]
 [ 0.40824829 -0.81649658]]

transpose(V):
 [[-0.40928285 -0.81597848  0.40824829]
 [-0.56345932 -0.12588456 -0.81649658]]
```

In [ ]:

```

```

In [ ]:

```

```