In [1]:
```
import numpy as np
from scipy.io import loadmat
```

Question 2

In [2]:
```
Xtrue = loadmat("incomplete.mat")["Xtrue"]
Y1 = loadmat("incomplete.mat")["Y1"]
Y2 = loadmat("incomplete.mat")["Y2"]
Y3 = loadmat("incomplete.mat")["Y3"]
```

In [3]:
```
def ItSingValThresh(Y, r):
    """
    Iterative Singular Value Thresholding function for Matrix Completion
    """
    tol = 10**(-3)  # difference between iterates at termination
    max_its = 100;
    n,p = Y.shape
    X = np.array(Y) # make a copy so operations do not mutate the original
    X[np.isnan(X)] = 0 # Fill in missing entries with zeros

    err = 10**6
    itt = 0

    while err > tol and itt < max_its:
        U,s,VT = np.linalg.svd(X, full_matrices=False)
        V, S = VT.T, np.diag(s)
        Xnew = U[:,:r]@S[:r,:r]@VT[:r,:] ### Complete this line
        for i in range(n):
            for j in range(p):
                if ~np.isnan(Y[i,j]):  #replace Xnew with known entries
                    Xnew[i,j] = Y[i,j]
        err = np.linalg.norm(X-Xnew,'fro')
        X = Xnew
        itt += 1
    return X
```

a) Apply the iterative singular value thresholding function (provided in the script) to the three incomplete matrices assuming the rank is 2. You will first need to complete the line of code in the function. Compare your recovered completed matrices to Xtrue (Note: compare the output by subtracting the completed matrix from the original matrix, and then displaying them). Does the number of missing entries affect the accuracy of the completed matrix?

Yes, the number of missing entries affect the accuracy of the completed matrix, because when the data has more missing entries, the accuracy decreases.

In [16]:
```
Er = Xtrue - ItSingValThresh(Y1, 2)
er_norm = np.linalg.norm(Er,ord='fro')
print('# of Y1 missing values:', np.count_nonzero(np.isnan(Y1)), 'and the frobenius norm is', er_norm)
```

```
# of Y1 missing values: 136 and the frobenius norm is 87.24667705099748
```

In [18]:
```
Er = Xtrue - ItSingValThresh(Y2, 2)
er_norm = np.linalg.norm(Er,ord='fro')
print('# of Y2 missing values:', np.count_nonzero(np.isnan(Y2)), 'and the frobenius norm is', er_norm)
```

```
# of Y2 missing values: 76 and the frobenius norm is 0.004735599527383998
```

In [19]:
```
Er = Xtrue - ItSingValThresh(Y3, 2)
er_norm = np.linalg.norm(Er,ord='fro')
print('# of Y3 missing values:', np.count_nonzero(np.isnan(Y3)), 'and the frobenius norm is', er_norm)
```

```
# of Y3 missing values: 16 and the frobenius norm is 0.0007153218655157115
```

b) Now apply your routine to the three incomplete matrices assuming the rank is 3. Compare your recovered completed matrices to Xtrue. Comment on the impact of using the incorrect rank in the completion process.

When we use the incorrect rank, the error increases..

In [21]:
```
Er = Xtrue - ItSingValThresh(Y1, 3)
er_norm = np.linalg.norm(Er,ord='fro')
print('# of Y1 missing values:', np.count_nonzero(np.isnan(Y1)), 'and the frobenius norm is', er_norm)
Er = Xtrue - ItSingValThresh(Y2, 3)
er_norm = np.linalg.norm(Er,ord='fro')
print('# of Y2 missing values:', np.count_nonzero(np.isnan(Y2)), 'and the frobenius norm is', er_norm)
Er = Xtrue - ItSingValThresh(Y3, 3)
er_norm = np.linalg.norm(Er,ord='fro')
print('# of Y3 missing values:', np.count_nonzero(np.isnan(Y3)), 'and the frobenius norm is', er_norm)
```

```
# of Y1 missing values: 136 and the frobenius norm is 128.77804846772077
# of Y2 missing values: 76 and the frobenius norm is 48.97940976510761
# of Y3 missing values: 16 and the frobenius norm is 20.78506989160173
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

△ Activity 15

1. $V = \begin{bmatrix} 1 & x & x \\ x & 2 & 4 \\ -1 & 2 & x \\ x & -2 & x \end{bmatrix}$

rank 1 matrix $V = \sigma_1 u_1 v_1^T$

a) col(V) is linear dep.

$$V = \begin{bmatrix} 1 & -2 & -4 \\ -1 & 2 & 4 \\ -1 & 2 & 4 \\ 1 & -2 & -4 \end{bmatrix}$$

b) each row should have at lease 1 value between columns, we need 2 values in the same row to compare (build relation)

so the minimun number is 6.