Assignment 1

Question 1

a.

Given the defenition of $\left||A|\right|_1 = \max_j\left(\sum_{i=1}^{m}\left|a_{i,j}\right|\right)$
To maximaize $\left||A|\right|_1$ *we will look for the column which holds the maximum values when summed as absolute values.*
Therefore, It is easy to see that the forth column maximaizes the term.

$$x = C_4 = \begin{bmatrix} 4 \\ 8 \\ 5 \\ -7 \end{bmatrix} \Rightarrow \left||A|\right|_1 = 24$$

So we can pick $x = (0,0,0,-1)$
Because:

$$\frac{\left||Ax|\right|_1}{\left||x|\right|_1} = \frac{24}{1} = 24$$

Given the defenition of $\left||A|\right|_\infty = \max_i\left(\sum_{j=1}^{m}\left|a_{i,j}\right|\right)$
To maximaize $\left||A|\right|_\infty$ *we will look for the row which holds the maximum values when summed as absolute values.*
Therefore, It is easy to see that the second row maximaizes the term.

$$x = R_2 = [2 \quad 4 \quad -4 \quad 8] \Rightarrow \left||A|\right|_\infty = 18$$

So we can pick x=(1,1,-1,1)
Because:

$$\frac{\left||Ax|\right|_\infty}{\left||x|\right|_\infty} = \frac{18}{1} = 18$$

b.

Using SVD decomposition from linalg libaray, we get D matrix – diagonal matrix with sorted singular values. Taking the first, hence the largest – we end up with the value "13.8581".

```
A = np.array([[1,2,3,4],[2,4,-4,8],[-5,4,1,5],[5,0,-3,-7]])
U,D,V_T = ln.svd(A)
print(D[0])

print(ln.norm(A@V_T[0])-D[0])
print(V_T[0])
```

```
13.858100376465329
-1.7763568394002505e-15
[-0.29618621  0.35616716  0.06730298  0.88367923]
```

Using the given formula,

$$\|A\|_2 = \max_{\mathbf{x}} \frac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \sigma_{max}.$$

We conclude that: $\left|\left|A\right|\right|_2 = \sigma_{\max} = 13.8581$

We will take x to be the corresponding vector to the largest singular value, $V^T[0]$

$$x = [-0.296, 0.356, 0.067, 0.88]$$

$$\frac{\left|\left|Ax\right|\right|_2}{\left|\left|x\right|\right|_2} = \frac{\sigma_{max}}{1}$$

Question 2

a.

$$\subseteq$$

$$null(A^TA) = \{x: A^TAx = 0\} \subseteq \{x: x^TA^TAx = 0\} = \{x: (Ax)^TAx = 0\} = \{x: < Ax, Ax >= 0\}$$

From inner product definition:

$$< Ax, Ax \geq 0 \Leftrightarrow Ax = 0$$

And therefore:

$$\{x: < Ax, Ax \geq 0\} = \{x: Ax = 0\} = null(A)$$

We can conclude,

$$null(A^TA) \subseteq null(A)$$

$$\supseteq$$

$$null(A) = \{x: Ax = 0\} \subseteq \{x: A^T(Ax) = 0\} = null(A^TA)$$

Q.E.D

b.

We'll mark: $A^TA = B = B^T$

$$range(A^TA) = range(B) = range(B^T) = null(B)^\perp = null(A^TA)^\perp = null(A)^\perp = range(A^T)$$

Q.E.D

c.

We need to prove that for given b, there is a guaranteed x which satisfies:

$$A^TAx = A^Tb$$

In other words, that $A^Tb \in range(A^TA)$

From defenition $A^Tb \in range(A^T)$,

Using section b,($range(\ A^TA) = range(A^T)$ *)*

Therefore $A^Tb \in range(A^TA)$

Question 4

a.
Using the Cholesky decomposition:

$$(A^T A)x = A^T b$$
$$(L^T L)x = A^T b$$
$$Lx = y$$
$$L^T y = A^T b$$

$$Solve\ L^T y = A^T b \rightarrow Solve\ Lx = y$$

We can program as follows:

```
[17] import numpy as np
     import numpy.linalg as ln
     import math

[18] A = np.array([[2,1,2],[1,-2,1],[1,2,3],[1,1,1]])
     b=  np.array([6,1,5,2])

[19] A_TA = A.T@A

[21] L = np.linalg.cholesky(A_TA)

[24] def backwards_substitution(A,b):
       x = np.zeros(len(b))

       for i in range(len(b)-1, -1, -1):
           tmp = b[i]
           for j in range(len(b)-1, i, -1):
               tmp -= x[j]*A[i,j]
           x[i] = tmp/A[i,i]

       return x

     def forwards_substitution(A,b):
       x = []

       for i in range(len(b)):
           x.append(b[i])
           for j in range(i):
               x[i] -= A[i,j] * x[j]
           x[i] /= A[i,i]

       return x

     y = A.T @ b
     forward = forwards_substitution(L,y)
     sol = backwards_substitution(L.T,forward)
     print(sol)

     [1.7 0.6 0.7]
```

And finally get-

$$x = [1.7,0.6,0.7]$$

b.

<u>QR</u>

Using what we've learned in class

$$\hat{\mathbf{x}} = (A^\top A)^{-1} A^\top \mathbf{b} = (R^\top R)^{-1} R^\top Q^\top \mathbf{b} = R^{-1}(R^\top)^{-1} R^\top Q^\top \mathbf{b} = R^{-1} Q^\top \mathbf{b},$$

Where R is triangular,

```
Q,R = ln.qr(A)

x = (backwards_substitution(R,Q.T@b))
print(x)

[1.7 0.6 0.7]
```

Getting the same $x = [1.7,0.6,0.7]$

<u>SVD</u>

Using what we've learned in class

$$\Sigma V^\top \hat{\mathbf{x}} = U^\top \mathbf{b}$$

And the fact $V^T = V^{-1}$

So we get that

$$\hat{x} = (V^T)^{-1}\Sigma^{-1}U^T b = V\Sigma^{-1}U^T b$$

```
[12] U,sigma,V_T = ln.svd(A,full_matrices=False)

     x = V_T.T @ ln.inv(np.diag(sigma)) @U.T @ b
     print(x)

     [1.7 0.6 0.7]
```

Getting the same $x = [1.7,0.6,0.7]$

c.

```
residual = A@sol - b
A.T @ residual

array([2.66453526e-15, 3.55271368e-15, 6.21724894e-15])
```

d.

Using what we've proved in class:

$$\hat{\mathbf{x}} = (A^\top W A)^{-1} A^\top W \mathbf{b}.$$

```
[77] w = [1000,1,1,1]
     W = np.diag(w)
     A_TWA = A.T@W@A
     A_TWb = A.T@W@b

     x_hat = ln.inv(A_TWA) @ A_TWb

     print(x_hat)

     [2.17244031 0.69218347 0.48106425]
```

Getting a weighted x result:

$$x = [2.17, 0.69, 0.48]$$

Question 5

```python
epsilon_1 = 1
epsilon_2 = 0.0000000001
epsilon = epsilon_1
QR_me_1 = np.array([[1,1,1],[epsilon,0,0],[0,epsilon,0],[0,0,epsilon]])
epsilon = epsilon_2
QR_me_2 = np.array([[1,1,1],[epsilon,0,0],[0,epsilon,0],[0,0,epsilon]])

def QR_gram_schmidt(A, modified = False):
  R = np.zeros((len(A[0]),len(A[0])))
  Q = np.zeros(A.shape)

  R[0,0] = ln.norm(A[:,0]) # first column

  if R[0,0] == 0:
    raise 'No gram schmidt basis could found due to 0 vector'

  Q[:,0] = np.divide(A[:,0] , R[0,0] )
  start_i = 1

  for i in range(start_i,A.shape[1]):
    Q[:,i] = A[:,i]
    for j in range(0,i):
      if modified:
        R[j,i] = Q[:,j].transpose() @ Q[:,i]
      else:
        R[j,i] = Q[:,j].transpose() @ A[:,i]

      Q[:,i]-= R[j,i]*Q[:,j]
    R[i,i] = ln.norm(Q[:,i])

    if R[i,i] == 0:
      raise 'No gram schmidt basis could found due to 0 vector'
    Q[:,i] /= R[i,i]
  return Q,R

Q_1,R_1 = QR_gram_schmidt(QR_me_1)
Q_2,R_2 = QR_gram_schmidt(QR_me_2)

Q_1_modified,R_1_modified = QR_gram_schmidt(QR_me_1,True)
Q_2_modified,R_2_modified = QR_gram_schmidt(QR_me_2,True)
```

```python
print("regular gram schmidt:\n")
print(Q_2)
print(R_2)
print("MODIFIED gram schmidt:\n")
print(Q_2_modified )
print(R_2_modified)
```

```
regular gram schmidt:

[[ 1.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 1.00000000e-10 -7.07106781e-01 -7.07106781e-01]
 [ 0.00000000e+00  7.07106781e-01  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  7.07106781e-01]]
[[1.00000000e+00 1.00000000e+00 1.00000000e+00]
 [0.00000000e+00 1.41421356e-10 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 1.41421356e-10]]
MODIFIED gram schmidt:

[[ 1.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 1.00000000e-10 -7.07106781e-01 -4.08248290e-01]
 [ 0.00000000e+00  7.07106781e-01 -4.08248290e-01]
 [ 0.00000000e+00  0.00000000e+00  8.16496581e-01]]
[[1.00000000e+00 1.00000000e+00 1.00000000e+00]
 [0.00000000e+00 1.41421356e-10 7.07106781e-11]
 [0.00000000e+00 0.00000000e+00 1.22474487e-10]]
```

c.

```python
[82] QT_Q1_regular   =  Q_1.transpose()@Q_1
     QT_Q1_modified   =  Q_1_modified.transpose()@Q_1_modified

     accuracy_regular1 = ln.norm((QT_Q1_regular - np.eye(QT_Q1_regular.shape[0])))
     accuracy_modified1 = ln.norm((QT_Q1_modified - np.eye(QT_Q1_modified.shape[0])))

     print(accuracy_regular1)
     print(accuracy_modified1)
```

```
5.319287782567757e-16
4.987305196443834e-16
```

```python
QT_Q2_regular   =  Q_2.transpose()@Q_2
QT_Q2_modified   =  Q_2_modified.transpose()@Q_2_modified

accuracy_regular2 = ln.norm((QT_Q2_regular - np.eye(QT_Q2_regular.shape[0])))
accuracy_modified2 = ln.norm((QT_Q2_modified - np.eye(QT_Q2_modified.shape[0])))

print(accuracy_regular2)
print(accuracy_modified2)
```

```
0.7071067811865477
1.1547005383855976e-10
```

The modified algorithm produces a smaller error norm, means the matrix $Q^T Q$ is more "close" to the identity, and more "close" to being orthogonal, therefore it is better than the non-modified algorithm.

Question 6

b.

Using the equation from section a:

$$V\Sigma^{-1}U^T = \sum_{i=1}^{\min(m,n)} \sigma_i^{-1} v_i u_i^T =$$

$$\hat{x} = V\Sigma^{-1}U^T b = \sum_{i=1}^{\min(m,n)} \sigma_i^{-1} v_i u_i^T * b = \sum_{i=1}^{\min(m,n)} \sigma_i^{-1} \begin{pmatrix} v_{1i} \\ .. \\ v_{ni} \end{pmatrix} \begin{pmatrix} u_{1i} & \dots & u_{ni} \end{pmatrix} * \begin{pmatrix} b_1 \\ .. \\ b_n \end{pmatrix}$$

$$= \sum_{i=1}^{\min(m,n)} \sigma_i^{-1} \begin{pmatrix} v_{1i}u_{1i} \dots v_{1i}u_{ni} \\ .. \\ v_{ni}u_{1i} \dots v_{ni}u_{ni} \end{pmatrix} \begin{pmatrix} b_1 \\ .. \\ b_n \end{pmatrix} = \sum_{i=1}^{\min(m,n)} \sigma_i^{-1} \begin{pmatrix} \sum_{k=1}^{n} v_{1i}u_{ki}b_k \\ .. \\ \sum_{k=1}^{n} v_{ni}u_{ki}b_k \end{pmatrix}$$

$$= \sum_{i=1}^{\min(m,n)} \sigma_i^{-1} \sum_{k=1}^{n} u_{ki}b_k \begin{pmatrix} v_{1i} \\ .. \\ v_{ni} \end{pmatrix} = \sum_{i=1}^{\min(m,n)} \sigma_i^{-1} u_i^T\, b v_i$$

c.

using section b,

$$\hat{x} = \sum_{i=1}^{\min(m,n)} \sigma_i^{-1} u_i^T\, b v_i$$

$$u_i^T b = \alpha_i \Rightarrow \sum_{i=1}^{\min(m,n)} \sigma_i^{-1} \alpha_i\, v_i = \hat{x}$$

Therefore we'll prove that $u_i^T b = \alpha_i$

$$b = \sum_{i=1,\dots,m} \alpha_i u_i$$

From U being orthogonal, we'll get that:

$$u_i^T b = u_i^T * \textstyle\sum_{j=1,\dots,m} \alpha_j u_j = \begin{cases} 0, \ i \neq j \\ u_j^T \alpha_j u_j = \alpha_j \left\|u_j\right\| = \alpha_j, \ i = j \end{cases}$$

We can sum up,

$$u_i^T b = \alpha_i$$

d.
As we proved in class, $A^T A \; is \; positive \; semi \; definite \; matrix \rightarrow x^T(A^T A)x \geq 0$
in addition, $\lambda I \; is \; diagonal \; matrix \; with \; \lambda > 0, defined \; as \; positive \; definite \; matrix \rightarrow x^T(\lambda I)x > 0$
So, $x^T(A^T A + \lambda I)x = x^T(A^T A)x + x^T(\lambda I)x > 0$
$\rightarrow \; (A^T A + \lambda I) \; is \; positive \; definite \; matrix$

e) From previous sections:
$(A^T A + \lambda I)x = \; A^T b, \quad A = U\Sigma V^T$
U,V are orthogonal, and Σ *diagonal so,*

$$U^T U = I \, , V^T V = I, \Sigma^T = \Sigma$$

$\rightarrow ((U\Sigma V^T)^T U\Sigma V^T + \lambda I)x = (U\Sigma V^T)^T b \;\; \rightarrow (V\Sigma^T U^T U\Sigma V^T + \lambda I)x = V\Sigma^T U^T b$
$\rightarrow (V\Sigma^2 V^T + \lambda I)x = V\Sigma^T U^T b$
We multiply from left with $\mathrm{V^T}$:
$\rightarrow (V^T V\Sigma^2 V^T + V^T \lambda I)x = V^T V\Sigma^T U^T b \;\rightarrow\; (\Sigma^2 V^T + V^T \lambda I)x = \Sigma^T U^T b$
$\rightarrow (\Sigma^2 V^T + \lambda I V^T)x = \Sigma^T U^T b \rightarrow (\Sigma^2 + \lambda I)V^T x = \Sigma^T U^T b$
We notice that $(\Sigma^2 + \lambda I) \; is \; positive \; definite \; matrix \; so \; it's \; vertible \; matrix,$
$\rightarrow V^T x = (\Sigma^2 + \lambda I)^{-1}\Sigma^T U^T b$
We multiply from left with V:
$\rightarrow VV^T x = V(\Sigma^2 + \lambda I)^{-1}\Sigma^T U^T b \rightarrow x = V((\Sigma^2 + \lambda I)^{-1}\Sigma^T)U^T b$

$((\Sigma^2 + \lambda I)^{-1}\Sigma^T) \; is \; diagonal \; matrix \; with \; the \; values : \; \dfrac{\sigma_i}{\sigma_i^2 + \lambda},$

$$so, ((\Sigma^2 + \lambda I)^{-1}\Sigma^T = \begin{pmatrix} \dfrac{\sigma_1}{{\sigma_1}^2 + \lambda} & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & \dfrac{\sigma_n}{{\sigma_n}^2 + \lambda} \end{pmatrix}$$

From section b, $x = \sum_{i=1}^{\min(n,m)} \frac{\sigma_i}{{\sigma_i}^2+\lambda} u_i^T b \; v_i$
And from section c, $x = \sum_{i=1}^{\min(n,m)} \frac{\sigma_i}{{\sigma_i}^2+\lambda} \alpha_i \; v_i$

f.
from previous sections, we can assume that by using SVD decomposition:

$$\hat{x} = \sum_{i=1}^{\min(n,m)} \frac{1}{\sigma_i} \alpha_i v_i$$

And by using regularized LS problem:

$$\hat{x} = \sum_{i=1}^{\min(n,m)} \frac{\sigma_i}{\sigma_i^2 + \lambda} \alpha_i v_i$$

Assuming that the noise in the image corresponds to a singular vector u_i with a **small** corresponding singular value σ_i, we notice that $\frac{1}{\sigma_i}$ might be very big an create unwanted results, in contrast to $\frac{\sigma_i}{\sigma_i^2+\lambda}$ that by using the right λ we can get more balance result, and it not effect that much on the deblurring process.

Question 7

a. There are 4 unknown variables in K, therefore we need 4 equations.
Every correspondence contains 2 equations,

$$f_x x_i + x_0 z_i = u_i$$
$$f_y y_i + y_0 z_i = v_i$$

So, given 2 linear independence correspondences we can find K, which is the minimal number of correspondences required.

b. Given n>2 correspondences we'll note the samples as follows-

$$< \begin{bmatrix} u_1 \\ v_1 \end{bmatrix}, \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} > \cdots < \begin{bmatrix} u_n \\ v_n \end{bmatrix}, \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} >,$$

for each tuple of samples we will create:

$$A_i = \begin{pmatrix} x_i & z_i & 0 & 0 \\ 0 & 0 & y_i & z_i \end{pmatrix}$$

$$b_i = \begin{bmatrix} u_i \\ v_i \end{bmatrix}$$

We aim to find x:

$$x = \begin{bmatrix} f_x \\ x_0 \\ f_y \\ y_0 \end{bmatrix}$$

It holds that,

$$A_i \cdot x = \begin{bmatrix} x_i f_x + x_0 z_i \\ y_i f_y + y_0 z_i \end{bmatrix} = \begin{bmatrix} u_i \\ v_i \end{bmatrix} = b_i$$

Then we stack the matrices as rows of matrix "A", and the vectors b_i as rows of b,
And finally solve with LS the equation Ax=b :

$$A = \begin{pmatrix} A_1 \\ \dots \\ A_n \end{pmatrix} = \begin{pmatrix} x_1 & z_1 & 0 & 0 \\ 0 & 0 & y_1 & z_1 \\ & & \dots & \\ x_n & z_n & 0 & 0 \\ 0 & 0 & y_n & z_n \end{pmatrix} \in M^{2n\ x\ 4}$$

$$b = \begin{bmatrix} b_1 \\ \dots \\ b_n \end{bmatrix} = \begin{bmatrix} u_1 \\ v_2 \\ \dots \\ u_n \\ v_n \end{bmatrix} \in M^{2n\ x\ 1}$$

$$Ax = b \leftrightarrow \begin{pmatrix} x_1 & z_1 & 0 & 0 \\ 0 & 0 & y_1 & z_1 \\ & & \dots & \\ x_n & z_n & 0 & 0 \\ 0 & 0 & y_n & z_n \end{pmatrix} * \begin{bmatrix} f_x \\ x_0 \\ f_y \\ y_0 \end{bmatrix} = \begin{bmatrix} u_1 \\ v_2 \\ \dots \\ u_n \\ v_n \end{bmatrix}$$