Assignment – 1

Maor Ashkenazi – 312498017

Pan Eyal - 208722058

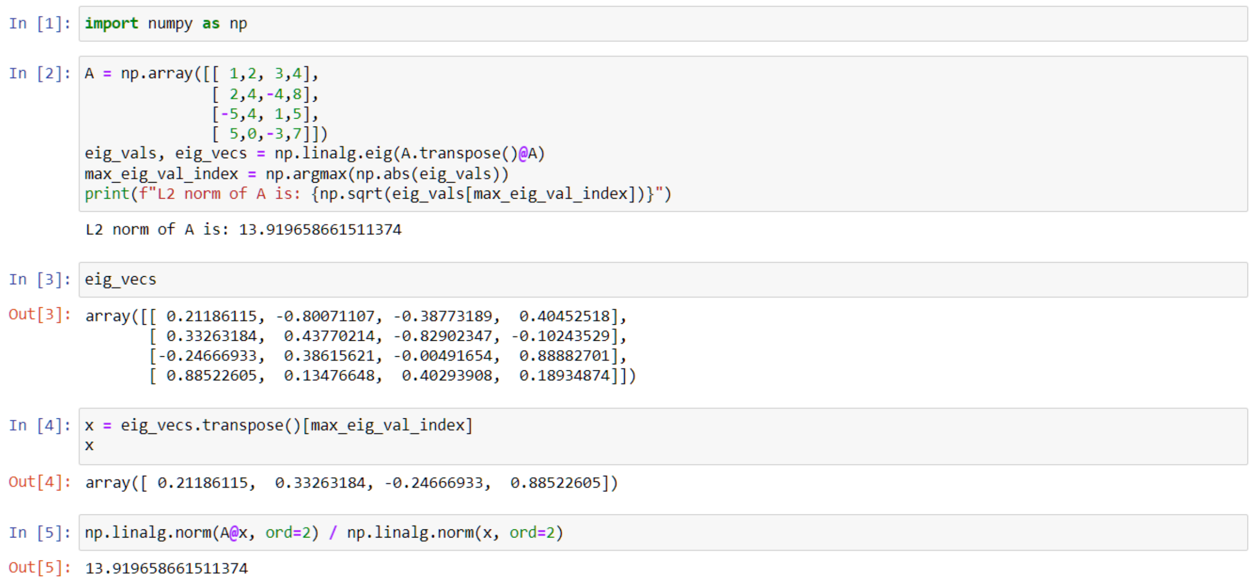
Problem 1:

a) For Matrix A, using l_1 norm, the vector $x_1 = [0,0,0,1]$ will maximize the expression, as it will extract the last column which has the highest sum of absolute values. The intuitive explanation is that we want to focus all the vector "weights" on the specific row/column that results as the highest sum of the absolute values.
Using l_∞ norm, the vector $x = [0.25, 0.25, -0.25, 0.25]$ will maximize the expression, as it will "spread" the weights across the vector indexes, thus will minimize the denominator (which will be 0.25, the l_∞ norm of x). In addition, the matrix-vector product will be of the following form:
$Ax = [0.25(1+2-3+4),\ 0.25(2+4+4+8), 0.25(-5+4-1+5), 0.25(5+0+3-7)] = 0.25[4, 18, 3, 1]$. The second member in the vector, is the sum of absolute values of the second row of A, and the coefficient 0.25 will cancel out when dividing by x's norm.

b) To find the vector x that maximizes the expression, we will look at A^TA's eigenvectors and choose the one with the largest eigenvalue.

$$\max\frac{||Ax||_2}{||x||_2} = \max\frac{\sqrt{<Ax, Ax>}}{||x||_2} = \max\frac{\sqrt{(Ax)^T(Ax)}}{||x||_2} = \max\frac{\sqrt{x^TA^TAx}}{||x||_2} \underbrace{=}_{for\ A^TA\ eigen\ vector\ x} \frac{\sqrt{x^T\lambda x}}{||x||_2} = \frac{\sqrt{\lambda}||x||_2}{||x||_2} = \sqrt{\lambda}$$

If we choose x to be the eigenvector, with the largest eigenvalue, the expression above will be equal to the largest singular value for A, which is in fact the definition of the induced l_2 norm.
Using the following snippet of python code, we will find x: $[0.21186115,\ 0.33263184, -0.24666933,\ 0.88522605]$.



```
In [1]: import numpy as np

In [2]: A = np.array([[ 1,2, 3,4],
                      [ 2,4,-4,8],
                      [-5,4, 1,5],
                      [ 5,0,-3,7]])
        eig_vals, eig_vecs = np.linalg.eig(A.transpose()@A)
        max_eig_val_index = np.argmax(np.abs(eig_vals))
        print(f"L2 norm of A is: {np.sqrt(eig_vals[max_eig_val_index])}")

        L2 norm of A is: 13.919658661511374

In [3]: eig_vecs

Out[3]: array([[ 0.21186115, -0.80071107, -0.38773189,  0.40452518],
               [ 0.33263184,  0.43770214, -0.82902347, -0.10243529],
               [-0.24666933,  0.38615621, -0.00491654,  0.88882701],
               [ 0.88522605,  0.13476648,  0.40293908,  0.18934874]])

In [4]: x = eig_vecs.transpose()[max_eig_val_index]
        x

Out[4]: array([ 0.21186115,  0.33263184, -0.24666933,  0.88522605])

In [5]: np.linalg.norm(A@x, ord=2) / np.linalg.norm(x, ord=2)

Out[5]: 13.919658661511374
```

Problem 2:

a) $\Longrightarrow$ Let there be x s.t. $x \in null(A) \rightarrow Ax = 0$. Therefore, $(A^TA)x = A^T(Ax) = A^T0 = 0$. This means that $x \in null(A^TA)$.
Thus, $null(A) \subseteq null(A^TA)$.

$\Longleftarrow$ Let there be x s.t. $x \in null(A^TA) \rightarrow A^TAx = 0$.

Therefore, $x^TA^TAx = x^T0 = 0 \rightarrow (Ax)^TAx = 0 \rightarrow \langle Ax, Ax\rangle = 0 \rightarrow Ax = 0$. This means that $x \in null(A)$.

Thus, $null(A^TA) \subseteq null(A)$.

By showing both sides, we concur that $null(A) = null(A^TA)$.

b) Using the definition given in the exercise, we conclude the following:
$range((A^TA)^T) = null(A^TA)^\perp = null(A)^\perp = range(A^T)$.
In addition, A^TA is symmetric because:

$$(A^TA)^T = A^T(A^T)^T = A^TA$$

This means that $range((A^TA)^T) = range(A^TA) = range(A^T)$, as requested.

c) By definition $y = A^Tb \in range(A^T)$.
From section 2(b) we conclude that $range(A^T) = range(A^TA)$.
And therefore, $y \in range(A^TA)$.
By definition, there must be some vector x s.t. $A^TAx = y \in range(A^TA)$.

Problem 4:

a) We will solve the Least Square problem as taught in class for A and b, using the Cholesky factorization.
The normal equations for $Ax = b$ are:

$$A^TAx = A^Tb \rightarrow \begin{bmatrix} 7 & 3 & 9 \\ 3 & 10 & 7 \\ 9 & 7 & 15 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 20 \\ 16 \\ 30 \end{bmatrix} \rightarrow \begin{cases} 7x_1 + 3x_2 + 9x_3 = 20 \\ 3x_1 + 10x_2 + 7x_3 = 16 \\ 9x_1 + 7x_2 + 15x_3 = 30 \end{cases}$$

The Cholesky factorization for A^TA is $LL^T \rightarrow L \approx \begin{bmatrix} 2.646 & 0 & 0 \\ 1.134 & 2.952 & 0 \\ 3.402 & 1.065 & 1.515 \end{bmatrix}$.

Solving for x requires two steps:

1. Forward substitution (for lower triangular matrix L) to solve for y ($Ly = A^Tb \rightarrow y = L^Tx$).
2. Backward substitution (for upper triangular matrix L^T) to solve for x.

Using the following code, we solved for $x = \begin{bmatrix} 1.7 \\ 0.6 \\ 0.7 \end{bmatrix}$.

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

```
at_a = a.transpose()@a
at_b = a.transpose()@b
```

```
l = np.linalg.cholesky(at_a)
l
```

```
array([[2.64575131, 0.        , 0.        ],
       [1.13389342, 2.9519969 , 0.        ],
       [3.40168026, 1.06465462, 1.51495279]])
```

```
def bwd_sub(U, b):
    n = np.size(b)
    x = np.zeros(n, dtype=np.float64)
    for i in reversed(range(0, n)):
        sum = 0
        for j in range(i, n):
            sum += U[i, j] * x[j]
        x[i] = (1/U[i, i]) * (b[i] - sum)
    return x

def fwd_sub(L, b):
    n = np.size(b)
    x = np.zeros(n, dtype=np.float64)
    for i in range(0, n):
        sum = 0
        for j in range(0, i):
            sum += L[i, j] * x[j]
        x[i] = (1/L[i, i]) * (b[i] - sum)
    return x
```

```
y = fwd_sub(l, at_b)
x = bwd_sub(l.transpose(), y)
x
```

```
array([1.7, 0.6, 0.7])
```

Checking our solution:

```
x - np.linalg.solve(at_a, at_b)
```

```
array([-1.55431223e-15, -5.55111512e-16,  1.33226763e-15])
```

b) As taught in class, to solve Least Squares problem using the QR factorization we factorize A and do as follows:

$$A = QR \rightarrow A^T A = (QR)^T QR = R^T Q^T QR = R^T IR = R^T R$$

$$A^T Ax = A^T b \rightarrow x = (A^T A)^{-1} A^T b = (R^T R)^{-1} R^T Q^T b = R^{-1} (R^T)^{-1} R^T Q^T b = R^{-1} Q^T b$$

$$\rightarrow Rx = Q^t b$$

And we can solve for this using a backward substitution algorithm for R and $Q^T b$.

Using the following code, we solved for $x = \begin{bmatrix} 1.7 \\ 0.6 \\ 0.7 \end{bmatrix}$

```
q, r = np.linalg.qr(a)
```

```
q
```

```
array([[-0.75592895,  0.04839339,  0.41120147],
       [-0.37796447, -0.82268766, -0.38955929],
       [-0.37796447,  0.53232731, -0.7574764 ],
       [-0.37796447,  0.19357357,  0.32463274]])
```

```
r
```

```
array([[-2.64575131, -1.13389342, -3.40168026],
       [ 0.        ,  2.9519969 ,  1.06465462],
       [ 0.        ,  0.        , -1.51495279]])
```

```
x = bwd_sub(r, q.transpose()@b)
x
```

```
array([1.7, 0.6, 0.7])
```

c) As taught in class, to solve the Least Squares problem using the SVD factorization we do as follows:

$$(*) \ A = U\Sigma V^T \rightarrow \ A^T A = (U\Sigma V^T)^T U\Sigma V^T = V\Sigma^T U^T U\Sigma V^T = V\Sigma^T \Sigma V^T$$

$$(**) \ A^T b = (U\Sigma V^T)^T b = V\Sigma^T U^T b$$

$$A^T Ax = A^T b \rightarrow \ V\Sigma^T \Sigma V^T x = V\Sigma^T U^T b \quad \underbrace{\rightarrow}_{\substack{for\ invertible\ V\Sigma^T \\ (A\ is\ full\ rank)}} \quad \Sigma V^T x = U^T b \quad \underbrace{\rightarrow}_{for\ y=V^T x} \quad \Sigma y = U^T b$$

First, we solve $y = \Sigma^{-1} U^T b$. This is implemented using element-wise division by $diag(\Sigma)$ in the following code:

```
u, sigma, v_t = np.linalg.svd(a, full_matrices=False)
```

```
y = (u.transpose()@b)/sigma
```

After finding y, and because V is orthogonal, we can solve for x:

$$x = Vy => x = \begin{bmatrix} 1.7 \\ 0.6 \\ 0.7 \end{bmatrix}$$

```
x = v_t.transpose()@y
x
```

```
array([1.7, 0.6, 0.7])
```

d) The residual of the Least Squares is: $r = \begin{bmatrix} -0.6 \\ 0.2 \\ 0 \\ 1 \end{bmatrix}$.

We can see from the following code that:

$$A^T r = \vec{0}$$

```
r = a@x - b
r
```

```
array([-6.00000000e-01,  2.00000000e-01, -1.77635684e-15,  1.00000000e+00])
```

```
a.transpose()@r
```

```
array([-9.32587341e-15, -9.76996262e-15, -1.28785871e-14])
```

This is not surprising because we try to solve:

$$A^T A x = A^T b \rightarrow A^T A x - A^T b = 0 \rightarrow A^T(Ax - b) = 0 \rightarrow A^T r = 0$$

e) Using weighted least squares, we try to find:

$$A^T W A x = A^T W b$$

We can solve this using any of the techniques above. We have chosen to use the Cholesky factorization using the same steps listed above, only this time we factorize $A^T W A$ instead of $A^T A$.

This is implemented in the following code:

```
w = np.array([[900, 0, 0, 0],
              [  0, 1, 0, 0],
              [  0, 0, 1, 0],
              [  0, 0, 0, 1]])
```

```
l = np.linalg.cholesky(a.transpose()@w@a)
```

```
y = fwd_sub(l, a.transpose()@w@b)
x = bwd_sub(l.transpose(), y)
x
```

```
array([2.1723696 , 0.69216968, 0.48109701])
```

```
a@x - b
```

```
array([-8.97090862e-04,  2.69127259e-01, -8.92619312e-13,  1.34563629e+00])
```

Using the weight matrix $W = \begin{bmatrix} 900 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, we get the solution $x \approx \begin{bmatrix} 2.172 \\ 0.692 \\ 0.481 \end{bmatrix}$, with the residuals: $r \approx \begin{bmatrix} -9 * 10^{-4} \\ 0.269 \\ 0 \\ 1.346 \end{bmatrix}$.

Problem 5:

a) We implemented the Gram Schmidt and Modified Gram Schmidt in the following way:

```
def gram_schmidt_QR(A):
    m, n = A.shape
    R = np.zeros((n,n))
    Q = np.zeros((m,n))

    R[0,0] = np.linalg.norm(A[:,0], ord=2)
    Q[:,0] = A[:,0]/R[0,0]

    for i in range(1,n):
        Q[:,i] = A[:,i]
        for j in range(0,i):
            R[j,i] = Q[:,j] @ A[:,i]
            Q[:,i] = Q[:,i] - R[j,i] * Q[:,j]
        R[i,i] = np.linalg.norm(Q[:,i], ord=2)
        Q[:,i] = Q[:,i] / R[i,i]

    return Q,R

def gram_schmidt_QR_modified(A):
    m, n = A.shape
    R = np.zeros((n,n))
    Q = np.zeros((m,n))

    R[0,0] = np.linalg.norm(A[:,0], ord=2)
    Q[:,0] = A[:,0]/R[0,0]

    for i in range(1,n):
        Q[:,i] = A[:,i]
        for j in range(0,i):
            R[j,i] = Q[:,j] @ Q[:,i]
            Q[:,i] = Q[:,i] - R[j,i] * Q[:,j]
        R[i,i] = np.linalg.norm(Q[:,i], ord=2)
        Q[:,i] = Q[:,i] / R[i,i]

    return Q,R
```

b) Running the factorization with $\epsilon = 1$ receives the exact same factorization in both implementation:

$$Q = \begin{bmatrix} 0.70710678 & 0.40824829 & 0.28867513 \\ 0.70710678 & -0.40824829 & -0.28867513 \\ 0 & 0.81649658 & -0.28867513 \\ 0 & 0 & 0.8660254 \end{bmatrix} R = \begin{bmatrix} 1.41421356 & 0.70710678 & 0.70710678 \\ 0 & 1.22474487 & 0.40824829 \\ 0 & 0 & 1.15470054 \end{bmatrix}$$

```
eps = 1
A = np.array([[1,   1,   1],
              [eps, 0,   0],
              [0,   eps, 0],
              [0,   0,   eps]])
```

```
q, r = gram_schmidt_QR(A)
print("Original Version:")
print(q)
print(r)
```

```
Original Version:
[[ 0.70710678  0.40824829  0.28867513]
 [ 0.70710678 -0.40824829 -0.28867513]
 [ 0.          0.81649658 -0.28867513]
 [ 0.          0.          0.8660254 ]]
[[1.41421356 0.70710678 0.70710678]
 [0.         1.22474487 0.40824829]
 [0.         0.         1.15470054]]
```

```
q, r = gram_schmidt_QR_modified(A)
print("Modified Version:")
print(q)
print(r)
```

```
Modified Version:
[[ 0.70710678  0.40824829  0.28867513]
 [ 0.70710678 -0.40824829 -0.28867513]
 [ 0.          0.81649658 -0.28867513]
 [ 0.          0.          0.8660254 ]]
[[1.41421356 0.70710678 0.70710678]
 [0.         1.22474487 0.40824829]
 [0.         0.         1.15470054]]
```

Running the factorization with $\epsilon = 1 * 10^{-10}$ receives slightly different results:

$$Q_{original} = \begin{bmatrix} 1 & 0 & 0 \\ 1 * 10^{-10} & -7.07106781 * 10^{-1} & -7.07106781 * 10^{-1} \\ 0 & 7.07106781 * 10^{-1} & 0 \\ 0 & 0 & 7.07106781 * 10^{-1} \end{bmatrix} R_{original} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1.41421356 * 10^{-10} & 0 \\ 0 & 0 & 1.41421356 * 10^{-10} \end{bmatrix}$$

$$Q_{modified} = \begin{bmatrix} 1 & 0 & 0 \\ 1 * 10^{-10} & -7.07106781 * 10^{-10} & -4.08248290 * 10^{-1} \\ 0 & 7.07106781 * 10^{-10} & -4.08248290 * 10^{-1} \\ 0 & 0 & 8.16496581 * 10^{-1} \end{bmatrix} R_{modified} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1.41421356 * 10^{-10} & 7.07106781 * 10^{-11} \\ 0 & 0 & 1.22474487 * 10^{-10} \end{bmatrix}$$

```
eps = 1e-10
A = np.array([[1,   1,   1],
              [eps, 0,   0],
              [0,   eps, 0],
              [0,   0,   eps]])
```

```
q, r = gram_schmidt_QR(A)
print("Original Version:")
print(q)
print(r)
```

```
Original Version:
[[ 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]]
```

```
q, r = gram_schmidt_QR_modified(A)
print("Modified Version:")
print(q)
print(r)
```

```
Modified Version:
[[ 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) Running the factorization with $\epsilon = 1$ receives very similar results for $||Q^TQ - I||_F$ in both implementations:

$$||Q^TQ - I||_F \approx 5 * 10^{-16} \approx 0$$

```
q, r = gram_schmidt_QR(A)
print("Original Version:")
print(f"Norm of (Q^T)Q - I: {np.linalg.norm(q.transpose()@q - np.eye(3), ord='fro')}")
```

```
Original Version:
Norm of (Q^T)Q - I: 4.777941758212735e-16
```

```
q, r = gram_schmidt_QR_modified(A)
print("Modified Version:")
print(f"Norm of (Q^T)Q - I: {np.linalg.norm(q.transpose()@q - np.eye(3), ord='fro')}")
```

```
Modified Version:
Norm of (Q^T)Q - I: 4.987305196443834e-16
```

Running the factorization with $\epsilon = 1 * 10^{-10}$ receives drastically different results between the implementations:

Original: $||Q^TQ - I||_F = 0.7071067811865477 \approx 0.707$

Modified: $||Q^TQ - I||_F \approx 1.15 * 10^{-10} \approx 0$

```
q, r = gram_schmidt_QR(A)
print("Original Version:")
print(f"Norm of (Q^T)Q - I: {np.linalg.norm(q.transpose()@q - np.eye(3), ord='fro')}")
```

```
Original Version:
Norm of (Q^T)Q - I: 0.7071067811865477
```

```
q, r = gram_schmidt_QR_modified(A)
print("Modified Version:")
print(f"Norm of (Q^T)Q - I: {np.linalg.norm(q.transpose()@q - np.eye(3), ord='fro')}")
```

```
Modified Version:
Norm of (Q^T)Q - I: 1.1547005383855975e-10
```

Given the above results, it is obvious that the modified Graham Schmidt results in a better QR factorization. This can be seen by the fact that the original method, using $\epsilon = 1 * 10^{-10}$ results in a Q matrix which is not orthogonal. This can be explained by an accumulation of errors received when multiplying and subtracting in the second loop of the algorithm. The modified algorithm operates on smaller magnitude numbers, thus resulting in a smaller accumulation of errors.

Problem 6:

b) Using the equation in section (a): $V\Sigma^{-1}U^T = \sum_{i=1}^{\min(m,n)} \frac{1}{\sigma_i} v_i u_i^T$, we get:

$$\hat{x} = \left(\sum_{i=1}^{\min(m,n)} \frac{1}{\sigma_i} v_i u_i^T\right) * b \underbrace{\rightarrow}_{\substack{Experssions\ in\ paranthesis\ are\ matrices \\ By\ the\ distributive\ property\ of\ matrices}} \hat{x} = \sum_{i=1}^{\min(m,n)} \frac{1}{\sigma_i} v_i u_i^T b \underbrace{\rightarrow}_{\substack{u_i^T b\ is\ a\ 1\times 1\ matrix \\ Let\ it\ be\ marked\ as\ [k]}} \hat{x} = \sum_{i=1}^{\min(m,n)} \frac{1}{\sigma_i} v_i[k] \rightarrow$$

$$\rightarrow \hat{x} = \sum_{i=1}^{\min(m,n)} \frac{1}{\sigma_i} \begin{bmatrix} v_{i_1} * k \\ \dots \\ v_{i_n} * k \end{bmatrix} \rightarrow \hat{x} = \sum_{i=1}^{\min(m,n)} \frac{1}{\sigma_i} k v_i \rightarrow \hat{x} = \sum_{i=1}^{\min(m,n)} \frac{1}{\sigma_i} (u_i^T b) v_i$$

c) We start from the previous expression and replace b with $\sum_{i=1}^{m} \alpha_i u_i$

$$\hat{x} = \sum_{i=1}^{\min(m,n)} \frac{1}{\sigma_i} (u_i^T b) v_i \rightarrow \hat{x} = \sum_{i=1}^{\min(m,n)} \frac{1}{\sigma_i} \left(u_i^T \left(\sum_{j=1}^{m} \alpha_j u_j \right) \right) v_i \rightarrow$$

$$\underbrace{\rightarrow}_{\substack{distributive\ property\ of\ matrices \\ move\ u_i^T inside\ the\ sum}} \hat{x} = \sum_{i=1}^{\min(m,n)} \frac{1}{\sigma_i} \left(\left(\sum_{j=1}^{m} \alpha_j u_i^T u_j \right) \right) v_i \underbrace{\rightarrow}_{\substack{U\ is\ an\ orthogonal\ matrix: \\ for\ i\neq j \rightarrow u_i^T u_j = 0 \\ for\ i=j \rightarrow u_i^T u_j = 1}} \hat{x} = \sum_{i=1}^{\min(m,n)} \frac{1}{\sigma_i} \alpha_i v_i$$

d) To solve the regularized LS problem $argmin\left\{||Ax-b||_2^2 + \lambda||x||_2^2\right\}$, we will derive the expression (marked as f) and solve $\nabla_x f = 0$. Using basic derivative rules, the derivative of a sum is the sum of derivates, so we will break the expression into two derivatives:

$$\nabla_x\left(||Ax-b||_2^2\right) \underbrace{=}_{As\ developed\ in\ class} 2A^TAx - 2A^Tb$$

$$\nabla_x\left(\lambda||x||_2^2\right) \underbrace{=}_{As\ seen\ in\ NLA\ section\ 4.1.3\ (point\ 2)} 2\lambda x$$

$$2A^TAx - 2A^Tb + 2\lambda x = 0 \rightarrow 2A^TAx + 2\lambda x = 2A^Tb \rightarrow A^TAx + \lambda x = A^Tb \rightarrow (A^TA + \lambda I)x = A^Tb$$

Now we show that $(A^TA + \lambda I)$ is Positive Definite. Let there be y s.t $y \neq 0$:

$$y^T(A^TA + \lambda I)y = y^T(A^TAy + \lambda y) = \underbrace{y^TA^TAy}_{\geq 0} + y^T\lambda y = \underbrace{y^TA^TAy}_{\geq 0} + \underbrace{\lambda}_{\substack{>0 \\ by \\ definition}} * \underbrace{y^Ty}_{\substack{>0 \\ for \\ y\neq 0}} > 0$$

e) We need to show that the regularized Least Squares problem is given by:

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

From section (d) we get that:

$$(A^TA + \lambda I)\hat{x} = A^Tb$$

And since $A = U\Sigma V^T$,

$$((U\Sigma V^T)^T U\Sigma V^T + \lambda I)\hat{x} = (U\Sigma V^T)^T b$$
$$(V\Sigma^T U^T U\Sigma V^T + \lambda I)\hat{x} = V\Sigma^T U^T b$$

Since U is orthogonal, $U^TU = I$

$$(V\Sigma^T\Sigma V^T + \lambda I)\hat{x} = V\Sigma^T U^T b$$
$$(V\Sigma^2 V^T + \lambda I)\hat{x} = V\Sigma^T U^T b$$

Where Σ^2 is a diagonal matrix with $diag(\Sigma^2) = [\sigma_1^2, \dots, \sigma_{\min(m,n)}^2]$

Now we multiply by V^T from left:

$$(V^TV\Sigma^2V^T + V^T\lambda I)\hat{x} = V^TV\Sigma^TU^Tb$$

Since V is orthogonal, $V^TV = I$

$$(\Sigma^2V^T + \lambda V^T)\hat{x} = \Sigma^TU^Tb$$
$$(\Sigma^2 + \lambda)V^T\hat{x} = \Sigma^TU^Tb$$

Therefore, after multiplying V from the left again:

$$V^T\hat{x} = (\Sigma^2 + \lambda)^{-1}\Sigma^T U^T b$$
$$\hat{x} = V(\Sigma^2 + \lambda)^{-1}\Sigma^T U^T b$$

And since $\Sigma = \Sigma^T$:

$$\hat{x} = V(\Sigma^2 + \lambda I)^{-1}\Sigma U^T b$$

We notice that $(\Sigma^2 + \lambda I)^{-1}\Sigma$ is a diagonal matrix:

$$\begin{bmatrix} \frac{\sigma_1}{\sigma_1^2 + \lambda} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{\sigma_{\min(n,m)}}{\sigma_{\min(n.m)}^2 + \lambda} \end{bmatrix} \coloneqq \underbrace{\Sigma' = diag(\sigma_1', .., \sigma_{\min(n,m)}')}_{New\ notation}$$

We can use the results from sections (a)+(b) by using the diagonal matrix $((\Sigma')^{-1})^{-1}$) to conclude that:

$$\hat{x} = \sum_{i=1}^{\min(m,n)} \frac{1}{\frac{1}{\sigma_i{}'}}(u_i^T b)v_i$$

And following from section (c):

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

As requested.

f) In the tutorial we were presented with image deblurring example in the following format:

$$A - \ blurring\ operator$$

$$b - blurred\ image, corrupted\ by\ random\ noise$$

$$x - deblurred\ image\ (target)$$

When solving a straightforward least squares problem as follows: $argmin\left\{\left||Ax - b|\right|_2^2\right\}$, the resulting image was uninterpretable.

This issue can be explained by the fact that A is close to being singular, thus some of its eigenvalues are extremely small, and when computing it's inverse and multiplying by the noise term the noise increased in magnitude.

To fix this issue easily, we added a regularization term that enlarged A's eigenvalues such that A's inverse won't result in huge eigenvalues: $argmin\left\{\left||Ax - b|\right|_2^2 + \left||x|\right|_2^2\right\}$.

In the case presented here, we assume that the noise in the blurred image is represented in the singular value decomposition by a singular vector u_i and a corresponding (close to zero) singular value σ_i.

As we can see, trying to solve the regular least squares problem using SVD results in a term that divides by σ_i, which becomes very unstable when σ_i is close to zero:

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

When using the regularized version, with an SVD decomposition, this term is numerically stable because of the addition of λ to the denominator:

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

This means that even if one of the singular values is close to zero, the resulting coefficient will be stable.

Problem 7:

a) For the problem $\begin{bmatrix} u \\ v \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & x_0 \\ 0 & f_y & z_0 \\ 0 & 0 & 1 \end{bmatrix}}_{K} * \begin{bmatrix} x \\ y \\ z \end{bmatrix}$, two unique correspondences would be required to find a solution for K.

From the matrix equality we get the following equation system:

$$\begin{cases} u = xf_x + zx_0 \\ v = yf_y + zy_0 \end{cases}$$

We have four variables. To solve them we will need four equations in the equation system.

Each correspondence provides two equations. Given two unique correspondences will be enough to find all four variables of K.

b) We will find a good solution for K by formalizing our problem as follows:

$$\underbrace{\begin{bmatrix} x_1 & z_1 & 0 & 0 \\ \dots & \dots & \dots & \dots \\ x_n & z_n & 0 & 0 \\ 0 & 0 & y_1 & z_1 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & y_n & z_n \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} f_x \\ x_0 \\ f_y \\ y_0 \end{bmatrix}}_{\varphi} = \underbrace{\begin{bmatrix} u_1 \\ \dots \\ u_n \\ v_1 \\ \dots \\ v_n \end{bmatrix}}_{b}$$

Now, our least squares problem is:

$$argmin_\varphi \left\{ \left|\left|A\varphi - b\right|\right|_2^2 \right\}$$

We can solve this using the normal equations:

$$A^T A\varphi = A^T b$$

$$A^T A = \begin{bmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i z_i & 0 & 0 \\ \sum_{i=1}^{n} x_i z_i & \sum_{i=1}^{n} z_i^2 & 0 & 0 \\ 0 & 0 & \sum_{i=1}^{n} y_i^2 & \sum_{i=1}^{n} y_i z_i \\ 0 & 0 & \sum_{i=1}^{n} y_i z_i & \sum_{i=1}^{n} z_i^2 \end{bmatrix}, \quad A^T b = \begin{bmatrix} \sum_{i=1}^{n} x_i u_i \\ \sum_{i=1}^{n} z_i u_i \\ \sum_{i=1}^{n} y_i v_i \\ \sum_{i=1}^{n} z_i v_i \end{bmatrix}$$

The normal equations can be solved by any of the methods we used in problem (4).