1) Find the LU, LDU, and LDL^{T} factorization of the matrix

$$A = \begin{pmatrix} 1 & -1 & 2 & 0 \\ -1 & 2 & -3 & 1 \\ 2 & -3 & 1 & 3 \\ 0 & 1 & 3 & -4 \end{pmatrix}. \tag{1}$$

We can find the LU factorization by performing Gaussian elimination, recording the coefficients to find the L matrix, and the remaining upper triangular matrix is U:

$$A = \begin{pmatrix} 1 & -1 & 2 & 0 \\ -1 & 2 & -3 & 1 \\ 2 & -3 & 1 & 3 \\ 0 & 1 & 3 & -4 \end{pmatrix} \overset{1}{\to} \begin{pmatrix} 1 & -1 & 2 & 0 \\ 0 & 1 & -1 & 1 \\ 0 & -1 & -3 & 3 \\ 0 & 1 & 3 & -4 \end{pmatrix} \overset{2}{\to} \begin{pmatrix} 1 & -1 & 2 & 0 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & -4 & 4 \\ 0 & 0 & 4 & -5 \end{pmatrix} \tag{2}$$

$$\overset{3}{\to} \boxed{\begin{pmatrix} 1 & -1 & 2 & 0 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & -4 & 4 \\ 0 & 0 & 0 & -1 \end{pmatrix} = U}, \tag{3}$$

where

$$\tilde{L}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ -2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \tilde{L}_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix} \quad \tilde{L}_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}, \tag{4}$$

meaning

$$L = (\tilde{L}_3\tilde{L}_2\tilde{L}_1)^{-1} = \tilde{L}_1^{-1}\tilde{L}_2^{-1}\tilde{L}_3^{-1} \tag{5}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \tag{6}$$

$$= \boxed{\begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 2 & -1 & 1 & 0 \\ 0 & 1 & -1 & 1 \end{pmatrix}}. \tag{7}$$

We can write the LDU factorization by letting

$$\boxed{D = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -4 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}}, \tag{8}$$

which makes

$$\boxed{U = \begin{pmatrix} 1 & -1 & 2 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} = L^{\mathrm{T}}}, \tag{9}$$

which also conveniently gives us our LDL^{T} factorization of A.

2) Let A be an upper triangular $n \times n$ matrix and b be an n-vector.

a) Please write out the algorithm to solve $Ax = b$ by backward substitution, and calculate flops.

The algorithm is as follows:

```
x(n) = b(n)/A(n,n)
For i = n-1:1:-1
    x(i) = (y(i) - sum([x(k)*A(i,k) for k=i+1:n]))/A(i,i)
end
```

The flops can be calculated as follows (ignoring assignment operations). At the i^{th} step (for $i = 1, 2, \ldots, n$) we have 1 multiplication, 1 subtraction, $n - i$ multiplications, and $n - i - 1$ additions. Thus,

$$\boxed{\begin{aligned} \text{flops} &= \sum_{i=1}^{n} [1 + 1 + (n - i) + (n - i - 1)] = \sum_{i=1}^{n} [2n + 1 - 2i] \\ &= n(2n + 1) - n(n + 1) = n^2 = \mathcal{O}(n^2) \end{aligned}}. \tag{10}$$

b) Perform round-off error analysis, i.e. the substitution algorithm is backward stable in the sense that $(A + \delta A)\hat{x} = b$ with

$$\frac{|\delta a_{ij}|}{|a_{ij}|} \leq n\epsilon + \mathcal{O}(\epsilon^2), \tag{11}$$

where ϵ is the machine precision, δa_{ij} is the (i, j)-entry in δA, and a_{ij} is the (i, j)-entry in A.

We would like to show the equivalent result

$$|\delta A| \leq n\epsilon|A| + \mathcal{O}(\epsilon^2). \tag{12}$$

Note that we can write $Ax = b$ in expanded form as

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n & = b_1 \\ \qquad a_{22}x_2 + \ldots + a_{2n}x_n & = b_2 \\ & \vdots \\ \qquad\qquad a_{nn}x_n & = b_n \end{cases}. \tag{13}$$

This is solved using back-substitution:

$$x_n = b_n/a_{nn} \tag{14}$$
$$x_j = \frac{b_i - \sum_{k=j-1}^{n} a_{jk}x_k}{a_{ii}}. \tag{15}$$

To analyze the error we can use floating point arithmetic, making note that this is a recursive process (i.e. each successive calculation depends on the floating point errors accumulated in the previous caluculations). Consider the following algorithm:

```
w(1) = b(i)
For j = n,n-1,...,i-1
    w(j+1) = w(j) - a(i,j)x(j)
end
x(i) = w(i)/a(i,i)
```

Then,

$$\mathrm{fl}(w^{(j+1)}) = (\mathrm{fl}(w^{(j)}) - a_{ij}x_j(1+\delta_j))(1+\delta_j') \tag{16}$$
$$x_i a_{ii} = (1+\delta_i)\mathrm{fl}(w^i) \tag{17}$$

for $j = n, n-1, \ldots, i-1$. Expanding Eq. (16) explicitly, then

$$\frac{a_{ii}x_i}{1+\delta_i} = b_i(1+\delta_n')(1+\delta_{n-1}')\ldots(1+\delta_{i-1}') - \sum_{j=i-1}^{n} a_{ij}x_j(1+\delta_j)(1+\delta_n')\ldots(1+\delta_{i-1}') \tag{18}$$
$$\frac{a_{ii}x_i}{(1+\delta_i)(1+\delta_n')\ldots(1+\delta_{i-1}')} = b_i - \sum_{j=i-1}^{n} a_{ij}x_j\frac{(1+\delta_j}{(1+\delta_n')\ldots(1+\delta_{j-1}')}. \tag{19}$$

Thus,

$$\tilde{a}_{ii}x_i = b_i - \sum_{j=i-1}^{n} \tilde{a}_{ij}x_j, \tag{20}$$

where

$$\tilde{a}_{ii} = \frac{a_{ii}}{(1+\delta_i)(1+\delta_n')\ldots(1+\delta_{i-1}')} = (1+\epsilon_{ii})a_{ii}. \tag{21}$$

Furthermore,

$$\tilde{a}_{ij} = \frac{(1+\delta_j)a_{ij}}{(1+\delta'_n)\dots(1+\delta'_{j-1})} = a_{ij}(1+\epsilon_{ij}). \tag{22}$$

Observe that

$$\epsilon_{jj} \leq j\epsilon + \mathcal{O}(\epsilon^2) \leq n\epsilon + \mathcal{O}(\epsilon^2), \tag{23}$$

for some $\epsilon \in \mathbb{R}$. That is,

$$(A+\delta A)\hat{x} = b, \tag{24}$$

where

$$|\delta A| \leq n\epsilon|\delta A| + \mathcal{O}(\epsilon^2). \tag{25}$$

3) Let $A, \delta A \in \mathbb{R}^{n\times n}$ be full rank and $b, x, \delta x \in \mathbb{R}^n$. Prove that if $Ax = b$ and $(A+\delta A)(x+\delta x) = b$, then

$$\frac{||\delta x||}{||x+\delta x||} \leq \kappa(A)\frac{||\delta A||}{||A||}, \tag{26}$$

where $\kappa(A)$ is the condition number of A and $||\cdot||$ is any norm.

The second equality gives us

$$(A+\delta A)(x+\delta x) = Ax + A\delta x + \delta A(x+\delta x) = b \tag{27}$$

$$A\delta x + \delta A(x+\delta x) = 0. \tag{28}$$

Rearranging and multiplying both sides by A^{-1}, which exists since $Ax = b$ has unique solution for nontrivial b, we find

$$\delta x = -A^{-1}\delta A(x+\delta x) \Rightarrow ||\delta x|| = ||A^{-1}\delta A(x+\delta x)|| \leq ||A^{-1}||\,||\delta A||\,||x+\delta x|| \tag{29}$$

$$\boxed{\frac{||\delta x||}{||x+\delta x||} \leq ||A^{-1}||\,||\delta A|| = \kappa(A)\frac{||\delta A||}{||A||}}. \tag{30}$$

4) Solve the following linear system by direct method via hand calculation and computer programming:

$$\begin{cases} 4x_1 + x_2 - x_3 + x_4 = -2 \\ x_1 + 4x_2 - x_3 - x_4 = -1 \\ -x_1 - x_2 + 5x_3 + x_4 = 0 \\ x_1 - x_2 + x_3 + 3x_4 = 1 \end{cases}. \tag{31}$$

The system above is equivalent to the matrix equation $Ax = b$, which can be solved using Gaussian elimination via the following "augmented" matrix:

$$\begin{pmatrix} 4 & 1 & -1 & 1 & -2 \\ 1 & 4 & -1 & -1 & -1 \\ -1 & -1 & 5 & 1 & 0 \\ 1 & -1 & 1 & 3 & 1 \end{pmatrix} \to \begin{pmatrix} 1 & 1/4 & -1/4 & 1/4 & -1/2 \\ 1 & 4 & -1 & -1 & -1 \\ -1 & -1 & 5 & 1 & 0 \\ 1 & -1 & 1 & 3 & 1 \end{pmatrix} \tag{32}$$

$$\to \begin{pmatrix} 1 & 1/4 & -1/4 & 1/4 & -1/2 \\ 0 & 15/4 & -3/4 & -5/4 & -1/2 \\ 0 & -3/4 & 19/4 & 5/4 & -1/2 \\ 0 & -5/4 & 5/4 & 11/4 & 3/2 \end{pmatrix} \tag{33}$$

$$\to \begin{pmatrix} 1 & 1/4 & -1/4 & 1/4 & -1/2 \\ 0 & 1 & -1/5 & -1/3 & -2/15 \\ 0 & -3/4 & 19/4 & 5/4 & -1/2 \\ 0 & -5/4 & 5/4 & 11/4 & 3/2 \end{pmatrix} \tag{34}$$

$$\to \begin{pmatrix} 1 & 0 & -1/5 & 1/3 & -7/15 \\ 0 & 1 & -1/5 & -1/3 & -2/15 \\ 0 & 0 & 23/5 & 1 & -3/5 \\ 0 & 0 & 1 & 7/3 & 4/3 \end{pmatrix} \tag{35}$$

$$\to \begin{pmatrix} 1 & 0 & -1/5 & 1/3 & -7/15 \\ 0 & 1 & -1/5 & -1/3 & -2/15 \\ 0 & 0 & 1 & 5/23 & -3/23 \\ 0 & 0 & 1 & 7/3 & 4/3 \end{pmatrix} \tag{36}$$

$$\to \begin{pmatrix} 1 & 0 & -1/5 & 1/3 & -34/69 \\ 0 & 1 & -1/5 & -1/3 & 11/69 \\ 0 & 0 & 1 & 5/23 & -3/23 \\ 0 & 0 & 0 & 146/69 & 101/69 \end{pmatrix} \tag{37}$$

$$\to \begin{pmatrix} 1 & 0 & 0 & 26/69 & -34/69 \\ 0 & 1 & 0 & -20/69 & 11/69 \\ 0 & 0 & 1 & 5/23 & -3/23 \\ 0 & 0 & 0 & 146/69 & 101/69 \end{pmatrix} \tag{38}$$

$$\to \begin{pmatrix} 1 & 0 & 0 & 0 & -55/73 \\ 0 & 1 & 0 & 0 & 3/73 \\ 0 & 0 & 1 & 0 & -41/146 \\ 0 & 0 & 0 & 1 & 101/146 \end{pmatrix}. \tag{39}$$

Thus,

$$x = \begin{pmatrix} -55/73 & 3/73 & -41/146 & 101/146 \end{pmatrix}^{\mathrm{T}}. \tag{40}$$

This problem was solved numerically using the program below.

```python
#!/usr/bin/env python3

import numpy as np

def LU_factorize(A):
    n = np.shape(A)[0]

    L = np.zeros(np.shape(A))
    U = np.copy(A)

    for j in range(n):
        L[j,j] = 1
        for i in range(j+1,n):
            L[i,j] = U[i,j]/U[j,j]
        for l in range(j+1,n):
            for m in range(j,n):
                U[l,m] = U[l,m] - U[j,m]*L[l,j]
    return L,U

def solve_x(A,b):
    n = np.shape(A)[0]

    L,U = LU_factorize(A)

    y = np.zeros(n)
    y[0] = b[0]/L[0,0]
    for i in range(1,n):
        temp = np.array([y[k]*L[i,k] for k in range(i)])
        y[i] = (b[i] - np.sum(temp))/L[i,i]

    x = np.zeros(n)
    x[-1] = y[-1]/U[-1,-1]
    for i in range(n-2,-1,-1):
        temp = np.array([x[k]*U[i,k] for k in range(i+1,n)])
        x[i] = (y[i] - np.sum(temp))/U[i,i]


    return x

if __name__ == '__main__':

    A = np.array([
        [ 4.0, 1.0,-1.0,1.0],
        [ 1.0, 4.0,-1.0,-1.0],
        [-1.0,-1.0, 5.0,1.0],
        [ 1.0,-1.0, 1.0,3.0]
        ],)
    b = np.array([-2,-1,0,1])
    L,U = LU_factorize(A)
    x = solve_x(A,b)

    np.set_printoptions(precision=3)
    print('\nL=\n{}\n'.format(L))
```

```
    print('U=\n{}\n'.format(U))
    print('x={}\n'.format(x))
```

The solution is shown in the image below:

```
L=
[[ 1.     0.     0.     0.   ]
 [ 0.25   1.     0.     0.   ]
 [-0.25  -0.2    1.     0.   ]
 [ 0.25  -0.333  0.217  1.   ]]

U=
[[ 4.     1.    -1.     1.   ]
 [ 0.     3.75  -0.75  -1.25 ]
 [ 0.     0.     4.6    1.   ]
 [ 0.     0.     0.     2.116]]

x=[-0.753  0.041 -0.281  0.692]
```

5) Show how LU factorization with partial pivoting works for the matrix via hand calculation and computer programming:

$$A = \begin{pmatrix} 4 & 7 & 3 \\ 1 & 3 & 2 \\ 2 & -4 & -1 \end{pmatrix}. \tag{41}$$

Give the P, L, and U matrices.

The LU factorization with partial pivoting for this matrix occurs as follows:

$$\begin{pmatrix} 4 & 7 & 3 \\ 1 & 3 & 2 \\ 2 & -4 & -1 \end{pmatrix} \xrightarrow{1} \begin{pmatrix} 4 & 7 & 3 \\ 0 & 5/4 & 5/4 \\ 0 & -15/2 & -5/2 \end{pmatrix} \xrightarrow{2} \begin{pmatrix} 4 & 7 & 3 \\ 0 & -15/2 & -5/2 \\ 0 & 5/4 & 5/4 \end{pmatrix} \tag{42}$$

$$\xrightarrow{3} \begin{pmatrix} 4 & 7 & 3 \\ 0 & -15/2 & -5/2 \\ 0 & 0 & 5/6 \end{pmatrix} = U. \tag{43}$$

Thus,

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \tag{44}$$

and

$$L_1 = \begin{pmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix} \;\; L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1/6 & 0 \end{pmatrix} \Rightarrow L = PL_1PL_2 = \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ -1/4 & -1/6 & 1 \end{pmatrix}. \quad (45)$$

The program below shows the algorithm to factor a matrix as $A = PLU$:

```python
#!/usr/bin/env python3

import numpy as np


def LU_factorize_pivot(A):
    n = np.shape(A)[0]

    P = np.identity(n)
    L = np.zeros(np.shape(A))
    U = np.copy(A)

    for j in range(n):
        L[j,j] = 1
        switch_idx = j+np.argmax(np.abs(U[j:,j]))
        if switch_idx != j:
            temp = np.copy(U[switch_idx,:])
            U[switch_idx,:] = U[j,:]
            U[j,:] = temp

            temp = np.copy(P[switch_idx,:])
            P[switch_idx,:] = P[j,:]
            P[j,:] = temp

            temp = np.copy(L[j,:j])
            L[j,:j] = L[switch_idx,:j]
            L[switch_idx,:j] = temp
        for i in range(j+1,n):
            L[i,j] = U[i,j]/U[j,j]
        for l in range(j+1,n):
            for m in range(j,n):
                U[l,m] = U[l,m] - U[j,m]*L[l,j]

    return P,L,U


if __name__ == '__main__':

    A = np.array([
        [4.0, 7.0,3.0],
        [1.0,3.0,2.0],
        [2.0,-4.0,-1.0],
        ])

    P,L,U = LU_factorize_pivot(A)
```

```
np.set_printoptions(precision=3)
print('\nP=\n{}\n'.format(P))
print('\nL=\n{}\n'.format(L))
print('U=\n{}\n'.format(U))
```

The output of the code above is displayed in the image below:

```
P=
[[1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]]


L=
[[ 1.     0.     0.   ]
 [ 0.5    1.     0.   ]
 [ 0.25  -0.167  1.   ]]

U=
[[ 4.     7.     3.   ]
 [ 0.    -7.5   -2.5  ]
 [ 0.     0.     0.833]]
```