Problem 1. For an $n \times n$ matrix A, where

$$A_{1,1} = 2, \quad A_{1,2} = -1, \quad A_{n,n} = 2, \quad A_{n-1,n} = -1$$

and

$$A_{i,i} = 2, \quad A_{i,i+1} = -1, \quad A_{i,i-1} = -1, \quad \forall i \neq 1, n$$

and $A_{i,j} = 0$ otherwise. Compute its LU factorization with MATLAB/Python. Can the LU factorization be obtained faster than $O(n^3)$ complexity? If so, what would the algorithm be?

Solution: The matrix A is the following form:

$$A = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 & 0 & 0 \\ -1 & 2 & -1 & \ddots & \vdots & \vdots & \vdots \\ 0 & -1 & 2 & \ddots & -1 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & 2 & -1 & 0 \\ 0 & \cdots & -1 & 2 & -1 & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 2 & -1 & \\ 0 & \cdots & 0 & 0 & -1 & 2 & \end{bmatrix}$$

The code is as follows:

```
A = generate_tridiagonal_matrix(5);
[L_A, U_A] = LU_decomp(A);


function A = generate_tridiagonal_matrix(n)
    % Generates an n x n symmetric tridiagonal matrix with:
    %  2 on the diagonal and -1 on the sub- and super-diagonals

    e = ones(n,1);
    A = 2*diag(e) - diag(e(1:end-1),1) - diag(e(1:end-1),-1);
end

function[L, U] = LU_decomp(A)
    n_tuple = size(A);
    assert(n_tuple(1) == n_tuple(2));
    n = n_tuple(1);
    U = A;
    L = eye(n_tuple(1), n_tuple(2));
    for j = 1:n
        for i = j+1:n
            if U(j,j) == 0
                error("entry in U is 0");
            end
            L(i,j) = U(i,j) / U(j,j);
            U(i, j:n) = U(i,j:n) - L(i,j) * U(j,j:n);
```

```
            end
        end
    end
```

Note that given the matrix is a tridiagonal matrix, we know that we can exploit the structure of the matrix to get a faster run time. Let m denote the main diagonal, □

Problem 2. Implement a gradient descent method (with fixed step size) to solve

$$\min_{x \in \mathbb{R}^n} \frac{1}{2}(x - x_*)^T A (x - x_*)$$

where $x_* = [1, 2, \ldots, n]^T$. What is the complexity per step of gradient descent? (Show the complexity as n grows using a plot). Furthermore, by repeating the experiment for different n, extract the rate of convergence and show its dependency on n using big-O notation.

Problem 3. Repeat Problem 2 by implementing a conjugate gradient method.

Problem 4. Suppose in a gradient descent scheme, an error ϵ_k occurs:

$$x^{(k+1)} = x^{(k)} - s_k \nabla f(x^{(k)}) + \epsilon_k$$

where $\|\epsilon_k\| \leq \epsilon$, and $f = \frac{1}{2}(x - x_*)^T A (x - x_*)$ with A positive definite. Show the scheme can converge with a suitable choice of s_k.