**Name and section:** ______________________________

**ID number:** ______________________________

**E-mail:** ______________________________

1. Implement the Gaussian elimination algorithm in MATLAB (or in any other programming language). Subsequently, create a function (or subroutine) stored as `my_linsolver.m`, which utilizes Gaussian elimination and backward substitution to solve a system consisting of $n$ linear equations written in the form of

$$A\mathbf{x} = \mathbf{b},$$

with $A$ being an $n \times n$ non-singular matrix, $\mathbf{b}$ a given $n$-dimensional column vector, and $\mathbf{x}$ the ($n$-dimensional) solution vector. Also, the first line of your function (in MATLAB) should read: `function x = my_linsolver(A,b)` and the inputs must be a square matrix $A$ and rhs (right-hand-side) vector $\mathbf{b}$. Note that the output of your function is just the solution vector $\mathbf{x}$. Test your code with the $3 \times 3$ system:

$$\begin{aligned} 3x_1 + x_2 + 4x_3 &= 6, \\ x_2 - 2x_3 &= -3, \\ x_1 + 2x_2 - x_3 &= -2. \end{aligned}$$

The exact solution is $\mathbf{x} = [1, -1, 1]^T$. Then, use your function in order to solve the $4 \times 4$ system:

$$\begin{aligned} x_1 + x_2 + x_4 &= 2, \\ 2x_1 + x_2 - x_3 + x_4 &= 1, \\ 4x_1 - x_2 - 2x_3 + 2x_4 &= 0, \\ 3x_1 - x_2 - x_3 + x_4 &= -3. \end{aligned}$$

Compute the $l_2$-norm of the residual $||\mathbf{b} - A\hat{\mathbf{x}}||_2$ where $\hat{\mathbf{x}}$ is the solution computed for the $4 \times 4$ system. Attach **all** your codes and provide MATLAB output.

2. Write a MATLAB function (or subroutine) stored as `my_linsolver_lu.m` which utilizes the LU decomposition. The first line of your function (in MATLAB) should read: `function x = my_linsolver_lu(A,b)` and internally must employ Gaussian elimination (in order to convert $A$ into $U$ and $L$) together with forward and backward substitutions (recall the algorithm for solving a linear system by LU decomposition). Then, use your function to solve the $4 \times 4$ linear system of Question 1 **only**. Attach your codes and provide MATLAB output.

3. Write a MATLAB function called `trisolve.m` in order to solve the linear system $A\mathbf{x} = \mathbf{f}$ where $A$ is a tridiagonal $n \times n$ matrix of the form of

$$A = \begin{bmatrix} a_1 & c_1 & & & \\ b_2 & a_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ & & & b_n & a_n \end{bmatrix}.$$

Its first line should read: `function x = trisolve(a,b,c,f)`. The inputs here are the $n$-dimensional vectors: $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{f}$, and the output is the solution vector $\mathbf{x}$. Test your code with the $5 \times 5$ system with $a_i = 2$, $b_i = -1$, $c_i = -1$, and rhs vector $\mathbf{f} = [1, 0, 0, 0, 1]^T$. The exact solution is $x = [1, 1, 1, 1, 1]^T$. Attach your code and provide MATLAB output.

4. Consider the second-order, non-homogeneous ordinary differential equation

$$u'' - u = x, \tag{1}$$

where $u = u(x)$ satisfies the boundary conditions: $u(0) = u(1) = 0$. Problems of this sort [cf. (1)] together with boundary conditions on the unknown function $u(x)$ are called **boundary value problems** (BVPs) while the ODE given is called the *Helmholtz equation*.

(a) Use Taylor series to show that the second derivative of a function $u(x) \in C^4$ at a point $x_0$ can be approximated by:

$$u''(x_0) \approx \frac{u(x_0 + h) - 2u(x_0) + u(x_0 - h)}{h^2} + O(h^2).$$

This is called a **centered**, **finite difference approximation of the second derivative**.

(b) Use the approximation from part (a) and your `trisolve.m` function from question 2 in order to solve the BVP of Eq. (1) with $n = 24$ (see *Hints*, for details). If the exact solution to Eq. (1) is given by

$$u_{\text{exact}}(x) = \frac{e}{e^2 - 1}\left(e^x - e^{-x}\right) - x,$$

plot the numerical and exact solutions in the **same** figure using a different marker (say, open circles and solid line, respectively) and include a legend. Finally, calculate the $l_2$-norm of the absolute error: $||u_{\text{exact}} - u_{\text{numerical}}||_2$.

Include your code, any figure and MATLAB output.

*Hints:*

*(a) Divide the interval* $[0, 1]$ *into* $n + 1$ *equal subintervals and set* $x_i = ih$, $i = 0, 1, \ldots, n + 1$ *such that* $(n + 1)h = 1$ *holds. This way, you create an one-dimensional computational grid (or mesh).*

*(b) Then, we are interested in looking for the approximate solution* $u(x_i) \equiv u_i$ *with* $i = 1, \ldots, n$ *using the boundary conditions* $u_0 = u_{n+1} = 0$.

(c) *To do so, the BVP at the discrete level is written as a difference equation:*

$$\begin{aligned} \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} - u_i &= x_i, \Rightarrow \\ u_{i+1} - \left(2 + h^2\right) u_i + u_{i-1} &= h^2 x_i, \quad i = 1, \ldots, n. \end{aligned}$$

*Note that the latter equation is just a linear system of the form of* $A\boldsymbol{x} = \boldsymbol{f}$ *with* $A$ *being a tridiagonal matrix.*