MTH 261 — Computational Linear Algebra (S26) **Coding Homework 1**

Due: By **11:59pm** on **Monday, February 9, 2026**

Name: ______________________________

Goal. The goal of this assignment is to get comfortable writing clear MATLAB code that matches the mathematics we are doing in class. In particular, you will implement two basic building blocks for solving linear systems:

- **back-substitution** for an upper triangular system $Ux = b$,
- the **first pivot step** of Gaussian elimination on an augmented matrix.

What to turn in. Upload **one** MATLAB file (a single `.m` file) that contains the two functions

`backsub(U,b)` and `FirstPivot(A)`.

You may include a short testing section at the bottom of the same file (recommended), or test in a separate script on your computer, but only upload the one file containing the functions. A template function is provided on moodle.

Assumptions (for this assignment).

- For `backsub(U,b)`: U is an $n \times n$ *upper triangular* matrix and $u_{ii} \neq 0$ for all i.
- For `FirstPivot(A)`: A is an $n \times (n+1)$ augmented matrix and $A(1,1) \neq 0$. We will discuss pivoting later; for now, do **not** swap rows.

Restrictions. Inside your functions, do **not** use MATLAB solvers or shortcuts such as

$\backslash$ `inv` `rref` `linsolve`

or any built-in function that directly solves the system / performs elimination for you. You *may* use \ or `rref` *outside* your functions to check your work.

Part 1 — Back-substitution: `backsub(U,b)`

Consider the system $Ux = b$, where U is upper triangular:

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}, \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Because U is upper triangular, you can solve for x_n first and work upward. Find the formula by hand first, i.e. for $x_n, x_{n-1}, \ldots$

Task. Write a MATLAB function `backsub(U,b)` that takes U (an $n \times n$ upper triangular matrix) and b (an $n \times 1$ vector) and returns the solution vector x.

Function requirements.

- Use a loop (or loops) that follows the mathematical formula above.
- Do not use \, `inv`, etc. inside the function.
- You may assume $u_{ii} \neq 0$ so division by u_{ii} is valid.

Check (include in a comment or a short test block). Verify your function on:

$$U = \begin{bmatrix} 1 & 2 \\ 0 & 2 \end{bmatrix}, \qquad b = \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

You should run `x = backsub(U,b)` and confirm that $\|Ux - b\|$ is close to 0.

Part 2 — First pivot step: `FirstPivot(A)`

Warm-up (recommended; not turned in)

To prepare, do one Gaussian elimination example by hand (no pivoting). For the augmented matrix

$$A = \begin{bmatrix} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{bmatrix},$$

work through elimination to reach reduced row echelon form (RREF). You do **not** need to submit this work, but it will help you predict what your code should do.

Task

Write a MATLAB function `FirstPivot(A)` that takes an $n \times (n+1)$ augmented matrix and performs *only the first pivot column step* of Gaussian elimination (no pivoting):

1. Scale the first row so that $A(1,1) = 1$.

2. Use row operations to make $A(2,1) = A(3,1) = \cdots = A(n,1) = 0$.

Your output should have the form

$$A_{\text{out}} = \begin{bmatrix} 1 & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & * & * & \cdots & * \end{bmatrix}.$$

(You do not need to reduce any other columns.)

Function requirements.

- Do not swap rows (no pivoting). You may assume $A(1,1) \neq 0$.
- Do not use `rref` inside the function.
- Use clear row operations. A typical pattern is:

$$R_1 \leftarrow \frac{1}{A(1,1)} R_1, \qquad R_i \leftarrow R_i - A(i,1) R_1 \text{ for } i \geq 2.$$

Check. Use the matrix above and run `Aout = FirstPivot(A)`. Confirm that the first column of `Aout` is $[1,0,0]^T$ (up to rounding). You may compare with `rref(A)` *as a check only.*

Style / clarity expectations

Write code that a classmate could read:

- Use meaningful variable names (e.g. `n`, `x`, `s` for a running sum).
- Include a short comment header for each function describing inputs/outputs.
- Indent loops cleanly.
- Avoid quick coding tricks that are hard to interpret or unexplained constants.