MMAE 350 Homework 2: Direct Solvers, LU, and Performance

Matrix Algebra and Computational Cost

Due 1/27/2026

Learning objectives. By completing this assignment, you will:

- Implement Gaussian elimination and LU factorization for dense linear systems.
- Compare accuracy and runtime against optimized library solvers.
- Use `Numba` to accelerate loop-based numerical code.
- Interpret algorithmic scaling and performance limitations.

Problem statement. You will solve linear systems

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

of increasing size using multiple approaches and measure their computational cost. All computations are performed in Python.

To avoid numerical issues related to pivoting, you will generate *symmetric positive definite* matrices of the form

$$\mathbf{A} = \mathbf{B}^T\mathbf{B} + \alpha\mathbf{I},$$

with $\alpha > 0$.

Part A: Test system generation. For each system size

$$n \in \{200,\ 400,\ 600,\ 800\},$$

generate a random matrix $\mathbf{B}$, construct $\mathbf{A} = \mathbf{B}^T\mathbf{B} + \alpha\mathbf{I}$ (use $\alpha = 1$), and generate a random right-hand side $\mathbf{b}$. Use a fixed random seed for reproducibility.

Part B: Solver implementation and accuracy. Implement the following methods:

1. Gaussian elimination with back substitution.
2. LU factorization followed by forward and back substitution.

For each case, compute a reference solution using

$$\mathbf{x}_{\text{ref}} = \texttt{numpy.linalg.solve}(A, \mathbf{b}),$$

and report the relative error

$$\frac{\|\mathbf{x} - \mathbf{x}_{\text{ref}}\|}{\|\mathbf{x}_{\text{ref}}\|}.$$

Part C: Timing (pure Python). Measure the CPU time required to solve the system using:

- your Gaussian elimination implementation,
- your LU factorization and solve,
- `numpy.linalg.solve`.

Repeat each timing three times and report the minimum time.

Part D: Numba acceleration. Apply `@numba.njit` to *your own* Gaussian elimination, LU factorization, and triangular solve routines.

- Do *not* apply Numba to `numpy.linalg.solve`.
- Perform one warm-up call before timing to exclude JIT compilation cost.

Compute and report the speedup

$$\text{speedup} = \frac{t_{\text{Python}}}{t_{\text{Numba}}}.$$

Deliverables. Submit a PDF or notebook containing:

- A table of runtimes and relative errors for all methods.
- A plot of CPU time versus system size n.
- Short written responses (3–6 sentences each):
  1. Why does Numba not speed up `numpy.linalg.solve`?
  2. Why do loop-based solvers benefit from Numba?
  3. Which method scales best in your results, and why?
  4. What practical limitations appear as n increases?

Important note. This assignment is designed to highlight that *algorithm choice matters more than hardware.* Optimized library solvers rely on compiled linear algebra routines, while custom numerical methods require careful performance considerations to scale effectively.