Matrix Algebra for Engineering Systems

Solving Linear Systems

MMAE 350

Tuesday, January 20

Where We Are in the Course

In the previous lecture, we:

- Introduced matrices and vectors
- Defined the identity matrix and matrix inverse
- Wrote linear systems in the form $A\mathbf{x} = \mathbf{b}$
- Explored numerical issues using the Hilbert matrix

Today we focus on how engineers *actually solve* linear systems.

Why Linear Systems Matter in Engineering

Linear systems appear everywhere in engineering:

- Structural analysis (forces and displacements)
- Heat conduction (nodal temperatures)
- Electrical circuits (currents and voltages)
- Fluid flow and transport problems

In most applications:

$$A\mathbf{x} = \mathbf{b} \quad \text{with} \quad A \in \mathbb{R}^{n \times n},\ n \gg 1.$$

The Linear System $A\mathbf{x} = \mathbf{b}$

We seek a vector $\mathbf{x}$ such that:

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

- A: system matrix (geometry, material properties, connectivity)
- $\mathbf{x}$: unknowns (displacements, temperatures, potentials)
- $\mathbf{b}$: known loads, sources, or boundary data

This is the core mathematical structure of many engineering models.

Why Not Use the Matrix Inverse?

From algebra, we know:

$$\mathbf{x} = A^{-1}\mathbf{b}.$$

However, in practice:

- Computing A^{-1} is expensive
- It amplifies numerical errors
- It is unnecessary for solving systems

Engineers almost never compute matrix inverses explicitly.

What We Learned from the Hilbert Matrix

The Hilbert matrix example showed that:

- Some matrices are *ill-conditioned*
- Small changes in $\mathbf{b}$ can cause large changes in $\mathbf{x}$
- Numerical accuracy depends on the matrix, not just the algorithm

This motivates solving linear systems carefully.

Solving Linear Systems: The Right Way

Instead of computing A^{-1}, we:

$$\boxed{\text{Solve } A\mathbf{x} = \mathbf{b} \text{ directly}}$$

- Gaussian elimination
- LU factorization
- Iterative methods (later in the course)

These approaches are:

- More efficient
- More stable
- Scalable to large systems

Gaussian Elimination (Conceptual View)

Gaussian elimination:

- Transforms the system into an equivalent upper-triangular system
- Uses row operations
- Preserves the solution

After elimination:

$$U\mathbf{x} = \mathbf{c},$$

which can be solved by back-substitution.

LU Factorization

Many solvers factor the matrix:

$$\boxed{A = LU}$$

- L: lower triangular
- U: upper triangular

Then solve:

$$L\mathbf{y} = \mathbf{b},$$
$$U\mathbf{x} = \mathbf{y}.$$

This is the foundation of most numerical linear algebra software.

What Software Actually Does

When you write in Python:

```
x = np.linalg.solve(A, b)
```

Behind the scenes:

- No inverse is computed
- The system is factorized
- Forward and backward substitution are used

This is exactly what professional engineering codes do.

Conditioning and Accuracy

The accuracy of the solution depends on:

- The conditioning of A
- The numerical precision
- The solver algorithm

Even a *perfect* algorithm cannot fix a badly conditioned problem.

Key Takeaways

- Linear systems are central to engineering analysis
- The form $A\mathbf{x} = \mathbf{b}$ appears everywhere
- Matrix inverses are almost never used in practice
- Conditioning matters as much as algorithms
- Modern solvers rely on matrix factorization