Numerical Linear Algebra II

Mohamed Amine Hamadi

Plan

Introduction

The Big Picture

The Problem: We want to solve a system of linear equations:

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

- **Direct Methods (e.g., Gaussian Elimination, LU factorization):**
  - Find the *exact* solution (ignoring rounding errors).
  - Great for small matrices.
  - Expensive for huge matrices (e.g., $100,000 \times 100,000$).
- **Iterative Methods :**
  - Start with a *guess* $\mathbf{x}^{(0)}$.
  - Improve the guess step-by-step: $\mathbf{x}^{(0)} \to \mathbf{x}^{(1)} \to \mathbf{x}^{(2)} \dots$
  - Stop when the answer is "good enough".

The Weather Forecast

Imagine you are predicting tomorrow's temperature (x).

The Iterative Logic

1. You start with a rough guess: "Maybe 20°C."
2. You use a formula to refine it based on today's humidity, pressure, etc.
3. The calculation gives you a *better* guess: "21°C."
4. You use the *better* guess to calculate again.
5. Repeat until the value stops changing.

Key Concept: We don't calculate the answer directly; we *converge* towards it.

THE GENERAL IDEA

Rearranging the Equations

Consider a 3×3 system $A\mathbf{x} = \mathbf{b}$:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases}$$

Step 1: Solve for the diagonal variable.

$$\begin{cases} x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3) \\ x_2 = \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3) \\ x_3 = \frac{1}{a_{33}}(b_3 - a_{31}x_1 - a_{32}x_2) \end{cases}$$

This looks like a job for iteration!

The Jacobi Method

Jacobi Method: The "Batch" Update

Algorithm:

1. Start with an initial guess: $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots)$.
2. Plug the **old** values into the formulas to calculate **all** new values.
3. Replace the old batch with the new batch simultaneously.

The Formula (for row i):

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right)$$

Important Note

In the Jacobi method, to calculate $x_2^{(k+1)}$, you still use the **old** value of $x_1^{(k)}$, even if you just calculated a shiny new $x_1^{(k+1)}$.

Jacobi Example

System:

$$10x_1 + 2x_2 = 6$$

$$1x_1 + 10x_2 = 15$$

Guess: $x_1^{(0)} = 0, x_2^{(0)} = 0$.

Iteration 1:

- Calculate new x_1:

$$x_1^{(1)} = \frac{1}{10}(6 - 2(0)) = \mathbf{0.6}$$

- Calculate new x_2: *(Still using old $x_1 = 0$!)*

$$x_2^{(1)} = \frac{1}{10}(15 - 1(0)) = \mathbf{1.5}$$

Iteration 2:

- $x_1^{(2)} = \frac{1}{10}(6 - 2(1.5)) = \mathbf{0.3}$
- $x_2^{(2)} = \frac{1}{10}(15 - 1(0.6)) = \mathbf{1.44}$

Exact Solution: $x_1 = 0.25, x_2 = 1.475$. We are getting closer!

The Gauss-Seidel Method

Gauss-Seidel Method: The "Immediate" Update

The Question: In Jacobi, we calculate $x_1^{(k+1)}$ but don't use it until the next round. Isn't that wasteful?

The Improvement: Use the **newest available information** immediately.

The Formula:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right)$$

- If you have already updated x_1, use the **new** x_1 to update x_2.
- If you haven't updated x_3 yet, use the **old** x_3.

Analogy

Jacobi is like updating all apps on your phone after downloading them all. Gauss-Seidel is like installing and using each app as soon as it downloads.

Gauss-Seidel Example (Same System)

System: $10x_1 + 2x_2 = 6$, $1x_1 + 10x_2 = 15$. Guess: $x_1^{(0)} = 0, x_2^{(0)} = 0$.

Iteration 1:

- Calculate x_1:

$$x_1^{(1)} = \frac{1}{10}(6 - 2(0)) = \mathbf{0.6}$$

- Calculate x_2: *(Use the NEW* $x_1 = 0.6$ *just calculated!)*

$$x_2^{(1)} = \frac{1}{10}(15 - 1(\mathbf{0.6})) = \mathbf{1.44}$$

Result:

- Jacobi Iteration 1 Result: $(0.6, 1.5)$
- Gauss-Seidel Iteration 1 Result: $(0.6, 1.44)$

Notice how Gauss-Seidel is already closer to the true answer for x_2 (1.475) *than Jacobi was!*

Convergence Behavior

When does it work? (Convergence)

The methods don't always work. They only converge to the solution if the matrix A has specific properties.

Sufficient Condition: Diagonal Dominance

The method is guaranteed to converge if the matrix is **Strictly Diagonally Dominant**.

What does that mean? For every row, the magnitude of the diagonal element must be larger than the sum of magnitudes of all other elements in that row.

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \quad \text{for all } i$$

Example:

$$A = \begin{pmatrix} \mathbf{10} & 2 \\ 1 & \mathbf{10} \end{pmatrix} \quad \text{(Good! } 10 > 2 \text{ and } 10 > 1\text{)}$$

$$B = \begin{pmatrix} \mathbf{2} & 10 \\ 10 & \mathbf{1} \end{pmatrix} \quad \text{(Bad! } 2 \not> 10\text{)}$$

Complexity Comparison: Theory

Aspect	**Gaussian Elimination**	**Gauss-Seidel (Iterative)**
Time Complexity	$O(n^3)$	$O(k \cdot n)$ (or $O(n^2)$ sparse)
Memory	$O(n^2)$ (dense storage)	$O(n)$ (sparse storage)
For $n = 10^6$:	10^{18} operations	$\approx 10^{10}$ operations
Sparsity	Destroys sparsity (fill-in)	Preserves sparsity

Key Insight

For sparse matrices (e.g., from PDEs), Gaussian elimination creates **fill-in** (turning zeros into non-zeros), wasting memory and CPU time. Gauss-Seidel keeps the matrix sparse and processes only the relevant data.

Real-World Case Study: Bridge Stability Analysis

The Problem: Simulating stress on a bridge structure.

System Size:

- Matrix A size: $n = 20,000$ (unknowns).
- Unknown vector $\mathbf{x}$: Displacements.
- RHS $\mathbf{b}$: External forces.

Direct Method (Gaussian Elim.)

Storage:

$$20,000^2 \times 8 \text{ bytes} \approx \mathbf{3.2} \text{ GB}$$

(Note: A becomes dense during elimination!)

CPU Time:

- **2 Hours 15 Minutes**.

Iterative Method (Gauss-Seidel)

Storage:

$$200,000 \text{ non-zeros} \times 8 \text{ bytes} \approx \mathbf{1.6} \text{ MB}$$

(Stays sparse!)

CPU Time:

- **48 Seconds**.

Conclusion

- **Why Iterative?** Essential for large systems where direct methods are too slow.
- **Jacobi:** Simple, easy to parallelize, but slower.
- **Gauss-Seidel:** Faster convergence, but harder to compute in parallel.
- **Condition:** Both work best (and reliably) when the matrix is **Diagonally Dominant**.

Next Steps in Numerical Linear Algebra

If these are too slow, mathematicians use **Krylov Subspace Methods** (like Conjugate Gradient) or **Preconditioners** to speed things up even more!