

Lecture 08 - B

§8.4 Sparsity and Iterative Methods

T. Takahashi

Department of Mathematics and Statistics
University of New Mexico



We will answer these questions:

- Under what condition does Jacobi converge?
- How are Jacobi and GS coded in practice?
- What are sparse matrices?
- Why is sparsity important for iterative solvers?



Jacobi and GS

Jacobi Iteration

$$x^{(k)} = x^{(k-1)} + D^{-1}r^{(k-1)},$$

Gauss-Seidel Iteration

$$x^{(k)} = x^{(k-1)} + (D + C_L)^{-1}r^{(k-1)}$$

Example

We saw that Jacobi was working for this example:

$$A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad x^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Question

Does Jacobi and GS always work?

Convergence



Direct vs. Iterative

What are we trying to solve?

$$Ax = b$$

- **Direct Solver:** Each step works toward the exact solution x^*
 - Too expensive for large problems

$$\text{Cost} = \mathcal{O}(n^3)$$

- **Iterative Solver:** Each step improves the entire solution vector
 $x^{(k)} \rightarrow x^{(k+1)}$
 - Must worry about **convergence**

$$\lim_{k \rightarrow \infty} e^{(k)} = 0$$

Key Idea

Convergence is always a concern for iterative methods.

Error vs. Residual

Error Vector

The k^{th} **error** vector is

$$e^{(k)} = x^* - x^{(k)}$$

where x^* is the exact solution to $Ax = b$

Residual Vector

The k^{th} **residual** vector is

$$r^{(k)} := b - Ax^{(k)}$$

Their Relation

They are related to each other

$$\begin{aligned} r^{(k)} &= b - Ax^{(k)} \\ &= Ax^* - Ax^{(k)} \\ &= Ae^{(k)} \end{aligned}$$

Error Analysis

Part 1

- Start with the iteration:

$$x^{(k)} = x^{(k-1)} + Q^{-1}r^{(k-1)}.$$

- Subtract from the exact solution:

$$x^* - x^{(k)} = x^* - x^{(k-1)} - Q^{-1}r^{(k-1)}.$$

- Substitute $e^{(k)} = x^* - x^{(k)}$ and $r^{(k-1)} = Ae^{(k-1)}$:

$$e^{(k)} = e^{(k-1)} - Q^{-1}Ae^{(k-1)}.$$



Error Analysis

Part 1

- Start with the iteration:

$$x^{(k)} = x^{(k-1)} + Q^{-1}r^{(k-1)}.$$

- Subtract from the exact solution:

$$x^* - x^{(k)} = x^* - x^{(k-1)} - Q^{-1}r^{(k-1)}.$$

- Substitute $e^{(k)} = x^* - x^{(k)}$ and $r^{(k-1)} = Ae^{(k-1)}$:

$$e^{(k)} = e^{(k-1)} - Q^{-1}Ae^{(k-1)}.$$

How the Error Changes

$$e^{(k)} = Te^{(k-1)}, \quad T := I - Q^{-1}A.$$



Error Analysis

Part 2

- Apply the error relation repeatedly:

$$e^{(k)} = Te^{(k-1)} = T^2e^{(k-2)} = \dots = T^ke^{(0)}.$$

- Using a compatible vector norm and its induced matrix norm:

$$\begin{aligned}\|e^{(k)}\| &= \|T^ke^{(0)}\| \\ &\leq \|T^k\| \|e^{(0)}\| \\ &\leq \|T\|^k \|e^{(0)}\|.\end{aligned}$$

A Sufficient Condition for Convergence

$$\|T\| < 1 \implies \|e^{(k)}\| \longrightarrow 0.$$



What Does the Bound Tell Us?

Error Bound

$$\|e^{(k)}\| \leq \|T\|^k \|e^{(0)}\|.$$

- If $\|T\| < 1$, convergence is guaranteed.
- If $\|T\| \geq 1$, this bound is inconclusive.

Key Idea

As long as Q is chosen in a way that

$$\|I - Q^{-1}A\| < 1$$

stationary solvers will converge.



More Results

Divergence

A stationary iterative solver $x^{(k)} = x^{(k-1)} + Q^{-1}r^{(k-1)}$ will diverge if

$$\|I - Q^{-1}A\| > 1$$

Convergence Rate

Since error decreases like

$$\|e^{(k+1)}\| = \|I - Q^{-1}A\| \|e^{(k)}\|$$

the quantity $\|I - Q^{-1}A\|$ acts as the convergence rate.

The smaller this number is $\|I - Q^{-1}A\|$, the faster the iteration converges.



Convergence Rate

- The quantity

$$r(Q) = \|I - Q^{-1}A\|$$

depending on Q controls how quickly the iterative method converges

- Choices for Q
 - Jacobi $Q = D$
 - GS $Q = D + C_L$
 - There are several more but we focus only on these two

Question

- How many FLOPs do each method cost per iteration?
- Which method has a lower value of $r(Q)$?
- Which method has a lower runtime?
- Does the size of the matrix matter for convergence rate?



Convergence Example

Example

Will Jacobi iteration for the matrix

$$A = \begin{bmatrix} 3 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

converge or diverge?

- We know that for Jacobi iteration

$$Q = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 2 \end{bmatrix} \quad Q^{-1} = \begin{bmatrix} 1/3 & 0 & 0 \\ 0 & 1/3 & 0 \\ 0 & 0 & 1/2 \end{bmatrix}$$

- We have

$$I - Q^{-1}A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & -1/3 & 0 \\ -1/3 & 1 & -1/3 \\ 0 & -1/2 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1/3 & 0 \\ 1/3 & 0 & 1/3 \\ 0 & 1/2 & 0 \end{bmatrix}$$

- $\|I - Q^{-1}A\|_{\infty} = 2/3 < 1 \implies$ **convergent**

Diagonally Dominant

Definition: Definition: Diagonal Dominance

An $n \times n$ matrix A is **strictly diagonally dominant** if

$$|a_{ii}| > \sum_{j=1, j \neq i}^n |a_{ij}|$$

for all i .

Example:

$$A = \begin{bmatrix} 3 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

- Row 1: $|3| > |-1| + |0| = 1$
- Row 2: $|3| > |-1| + |-1| = 2$
- Row 3: $|2| > |0| + |-1| = 1$
- Therefore, A is diagonally dominant



Sufficient Convergence Criterion

Theorem

If A is strictly diagonally dominant, then the Jacobi and Gauss-Seidel methods converge for any initial guess $x^{(0)}$.

Key Idea

Checking the convergence criterion

$$\|I - Q^{-1}A\| < 1$$

can be a hassle. Checking for diagonal dominance is much faster and will guarantee convergence.



Why Diagonal Dominance Works

Jacobi

- For Jacobi, since $T = I - D^{-1}A$, the diagonal elements are always zero

$$t_{ii} = 0,$$

- The infinity norm is therefore the largest sum of the remaining values

$$t_{ij} = -\frac{a_{ij}}{a_{ii}} \quad (i \neq j), \quad \|T\|_{\infty} = \max_i \frac{\sum_{j \neq i} |a_{ij}|}{|a_{ii}|}.$$

Strict Diagonal Dominance

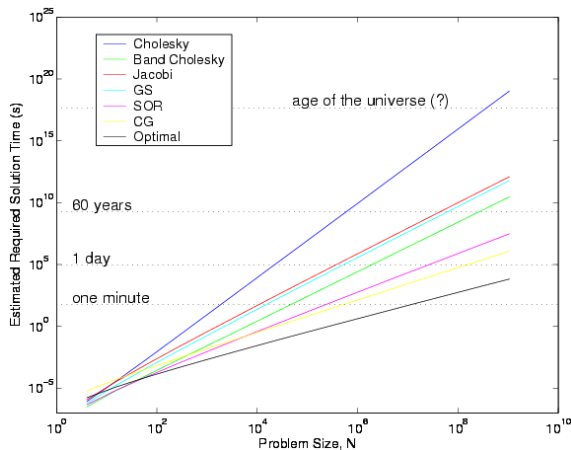
$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \implies \frac{\sum_{j \neq i} |a_{ij}|}{|a_{ii}|} < 1 \implies \|T\|_{\infty} < 1.$$



Component-Wise Implementation



Recall: Why do we need iterative solvers?



Key Idea

For problems of size $N = 10^6$ and greater, direct solvers are not feasible.

What Must We Compute?

- Every stationary iteration needs a correction:

Solve and Update

$$Qc^{(k)} = r^{(k)}, \quad x^{(k+1)} = x^{(k)} + c^{(k)}.$$

- Jacobi: $Q = D$.
- Gauss-Seidel: $Q = D + C_L$.

Why it matters:

All vectors and matrices created in Matlab requires RAM.



Explicitly Constructing Q

```
1 % A is loaded
2 Q = diag(diag(A));    % Jacobi: build diagonal matrix
3 x = x0;
4 for k = 1:kmax
5     r = b - A*x;
6     if norm(r) <= tol
7         break
8     end
9     c = Q\r;
10    x = x + c;
11 end
```

Memory Requirement

- Matrix A
- Matrix Q
- Vectors x , r , and c

Checking Memory

- Use the command `whos` to see how much memory a variable costs

```
1 >> A = zeros(10000,10000);  
2  
3 >> whos A  
4 Name              Size              Bytes    Class  
   Attributes  
5 A                10000x10000         8000000000 double
```

- A $10,000 \times 10,000$ matrix takes up 800 Megabytes of RAM
- Even the zeros take up memory



Checking Memory

- Use the command `whos` to see how much memory a variable costs

```
1 >> A = zeros(10000,10000);  
2  
3 >> whos A  
4      Name              Size              Bytes    Class  
      Attributes  
5      A                10000x10000          8000000000    double
```

- A $10,000 \times 10,000$ matrix takes up 800 Megabytes of RAM
- Even the zeros take up memory

Question

What if A is 1,000,000 by 1,000,000?



The Storage Problem

- Suppose A is $10^6 \times 10^6$.
- Each number in A is stored in FP64 $\implies$ each number is 8 bytes
- Any variable created in Matlab takes up Dynamic Random Access Memory (DRAM)
- Storing another dense double-precision matrix Q requires:

Additional Memory

$$8(10^6)^2 \text{ bytes} = 8 \times 10^{12} \text{ bytes} = \boxed{8 \text{ TB of RAM}}.$$

Key Idea

We need a way to an alternative way to program iterative solvers due to memory limitations.

Coding the Matrix as a Loop

Issue

Don't have enough memory to construct Q .

Solution

Code the action of Q^{-1} rather than the actual matrix.

- Instead of $D^{-1}b$ like this:

```
1 D\b
```

- We can code the action of $D^{-1}b$ in a loop

```
1 for i = 1:n  
2     b(i)/A(i,i)  
3 end
```



Use the Entries of A Directly

General Update

$$Qc^{(k)} = r^{(k)}, \quad x^{(k+1)} = x^{(k)} + c^{(k)}.$$

Jacobi: Independent Divisions

$$c_i = \frac{r_i}{a_{ii}}.$$

Gauss-Seidel: Forward Substitution

$$c_i = \frac{r_i - \sum_{j=1}^{i-1} a_{ij}c_j}{a_{ii}}, \quad i = 1, \dots, n.$$

Key Idea

We can solve $Qc = r$ without storing Q separately.

Jacobi: Vector vs. Component-Wise Form

Vector-update Form

$$x^{(k+1)} = -D^{-1}((C_L + C_U)x^{(k)} + b)$$

Component-Wise Form

For $i = 1, 2, \dots, n$

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

- Both are equivalent forms of Jacobi iteration
- Component-wise form does not require D , C_U , C_L
- The component-wise form is meant to be written a loop



Component-Wise Gauss-Seidel

Vector-update Form

$$x^{(k)} = -(D + C_L)^{-1}C_Ux^{(k-1)} + (D + C_L)^{-1}b$$

Component-wise Form

For $i = 1, 2, \dots, n$

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

- GS also has a component-wise form



Sparsity



Definition: Sparse Matrix

- A matrix is **sparse** if most of its entries are zero.
- Let A be $n \times n$. We assume each row has $m \ll n$ nonzeros.
- Many useful applications
 - Differential equations.
 - Circuits and networks.
 - Structural engineering.
- Example: Tridiagonal Toeplitz Matrix

$$A = \begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & -1 & 4 & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 4 \end{bmatrix}$$



Example: A Tridiagonal Matrix

$$A = \begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & -1 & 4 & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 4 \end{bmatrix}$$

- Blank entries are zero.
- At most $m = 3$ nonzeros per row.
- Only $3n - 2$ nonzeros out of n^2 entries.

Question

Why store all those zeros?



Sparse Storage

Dense Storage

Store all n^2 entries, including zeros.

$$\text{Memory} = \mathcal{O}(n^2).$$

Sparse Storage

Store nonzero values and indexing information.

$$\text{Memory} = \mathcal{O}(mn),$$

where m is the maximum number of nonzeros per row.

Key Idea

If m stays fixed as n grows, storage grows linearly.



Matlab Commands

- `gallery()` - construct various types of matrices
 - `doc gallery` for details
- `numel(A)` - number of elements in A
- `nnz(A)` - number of non-zero elements in A
- `sparse(A)` - create a sparse version of A
- `full(A)` - create a dense version of A
- `whos A` - view memory size of A

```
1 >> A = gallery('clement',5) % dense 5x5 Clement matrix
2 A =
3      0      1      0      0      0
4      4      0      2      0      0
5      0      3      0      3      0
6      0      0      2      0      4
7      0      0      0      1      0
8
9 >> fprintf('Sparsity ratio: %.2f\n',nnz(A)/numel(A))
10 Sparsity ratio: 0.32
```

Sparse vs. Full

```
1 >> A = gallery('clement',5);
2 >> whos A
3      Name      Size      Bytes  Class  Attributes
4      A         5x5        200    double
5
6 >> A = sparse(A)
7 A =
8      (2,1)      4
9      (1,2)      1
10     (3,2)      3
11     (2,3)      2
12     (4,3)      2
13     (3,4)      3
14     (5,4)      1
15     (4,5)      4
16
17 >> whos A
18      Name      Size      Bytes  Class  Attributes
19      A         5x5        176    double  sparse
```

Storage Format

```
1 >> A = sparse(A)
2 A =
3     (2,1)         4
4     (1,2)         1
5     (3,2)         3
6     (2,3)         2
7     (4,3)         2
8     (3,4)         3
9     (5,4)         1
10    (4,5)         4
```

- A sparse matrix in Matlab is stored as a pair between index and value
- For example, the first line (2, 1) 4 means $A_{21} = 4$

Key Idea

Zeros do not take up memory!

- Elimination-based solvers can create new nonzeros.

A is sparse $\nRightarrow L, U$ are sparse

Definition: Fill-in

If a direct solver creates non-zero elements in the factorization where there were originally zero elements, it is called **fill-in**.

- Fill-in increases memory cost.
- Reordering may reduce fill-in.

Key Idea

Direct solvers are not the most suited for handling sparse A matrices.



Fill-in Example

- Consider what happens if you attempt LU factorization on the matrix

$$A = \begin{bmatrix} 4 & -1 & \cdots & -1 \\ 4 & 4 & & \\ \vdots & & \ddots & \\ 4 & & & 4 \end{bmatrix}$$

- This matrix has exactly $3n - 2$ non-zero elements out of n^2
- But after one step of GE

$$\begin{bmatrix} 4 & -1 & \cdots & -1 \\ 4 & 4 & & \\ \vdots & & \ddots & \\ 4 & & & 4 \end{bmatrix} \longrightarrow \begin{bmatrix} 4 & -1 & -1 & \cdots & -1 \\ 0 & 5 & 1 & \cdots & 1 \\ 0 & 1 & 5 & \cdots & 1 \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 1 & 1 & \cdots & 5 \end{bmatrix}$$

- This is **fill-in**



Sparsity for Iterative Solvers



Make A and Q Sparse

```
1 % A is loaded as a sparse matrix
2 n = size(A,1);
3 Q = spdiags(diag(A), 0, n, n); % for Jacobi
4 % Q = tril(A);                % for GS
5 x = x0;
6 for k = 1:kmax
7     r = b - A*x;
8     if norm(r) <= tol
9         break
10    end
11    c = Q\r;
12    x = x + c;
13 end
```

- For iterative solvers, let Q be sparse
- You can operate with Q as if it's a regular matrix

Question

What is the bottleneck here?

Sparse Matrix-Vector Multiplication

- Each row computes a dot product:

$$(Ax)_i = \sum_{j=1}^n a_{ij}x_j.$$

- Sparse operations skip the zero entries:

$$(Ax)_i = \sum_{j: a_{ij} \neq 0} a_{ij}x_j.$$

At Most m Nonzeros per Row

$$\underbrace{n}_{\text{rows}} \times \underbrace{\mathcal{O}(m)}_{\text{work per row}} = \mathcal{O}(mn).$$

Key Idea

Calculating $r = b - Ax$ normally costs $\mathcal{O}(n^2)$ but only $\mathcal{O}(nm)$ if it's sparse.

Cost per Iteration

- Assume at most m nonzeros per row.

Both Methods: Compute the Residual

$$r = b - Ax \quad \text{cost: } \mathcal{O}(mn).$$

Compute the Correction

$$\text{Jacobi:} \quad Dc = r \quad \mathcal{O}(n)$$

$$\text{Gauss-Seidel:} \quad (D + C_L)c = r \quad \mathcal{O}(mn)$$

- Updating $x \leftarrow x + c$ costs $\mathcal{O}(n)$.

Key Idea

Both methods cost $\mathcal{O}(mn)$ per iteration when implemented with sparse operations.

Cost when A is Sparse

One step of Jacobi

$$x^{(k+1)} = \underbrace{D^{-1}}_{\mathcal{O}(n)} \underbrace{(- (C_L + C_U)x^{(k)} + b)}_{\mathcal{O}(mn)}$$

- Inversion: $\mathcal{O}(n)$
- MatVec Multiplication: $\mathcal{O}(mn)$

One step of GS

$$x^{(k+1)} = \underbrace{(D + C_L)^{-1}}_{\mathcal{O}(n^2)} \underbrace{(-C_U x^{(k)} + b)}_{\mathcal{O}(mn)}$$

- Inversion: $\mathcal{O}(n^2)$
- MatVec Multiplication: $\mathcal{O}(mn)$

Cheap Iterations Are Only Half the Story

- Suppose we need K iterations to reach our tolerance.

Total Work

$$\underbrace{\mathcal{O}(mn)}_{\text{cost per iteration}} \times \underbrace{K}_{\text{iterations}} = \mathcal{O}(mnK).$$

- Sparsity makes each iteration inexpensive.
- Slow convergence can still make the solve expensive.
- K may increase as the problem grows.



Direct vs. Iterative Methods

Cost for Direct Solvers

- Cost is $\mathcal{O}(n^3)$
- May cause Fill-in

Cost for Iterative Solvers

- Each iteration is $\mathcal{O}(mn)$
- Cost is $\mathcal{O}(mnK)$ in total

Key Idea

Iterative solvers excel in particular over direct methods if A is sparse.



Key Takeaways on Sparsity

- **Sparsity is everywhere:** Many scientific and engineering applications naturally lead to sparse matrices.
- **Direct solvers don't utilize sparsity:** LU factorization causes **fill-in**, increasing storage and computational cost.
- **Iterative solvers preserve sparsity:** They operate only on nonzero entries, leading to lower FLOPs and memory use.
- **For large problems, iterative solvers scale better:** They maintain efficiency even as n grows.



Questions

- Suppose it takes K iterations of Jacobi to get an acceptable answer. Does K depend on the size of the matrix?
- Which method is faster, Jacobi or GS?
- Which is faster? The vector-update form or the component-wise form?
- Are there other iterative solvers besides stationary iterative solvers?

