Numerical Computing

Lecture 7: Iterative Methods for Linear Systems

Francisco Richter Mendoza

Università della Svizzera Italiana
Faculty of Informatics

1 Introduction

For large sparse linear systems $Ax = b$, direct methods like Gaussian elimination become computationally prohibitive due to fill-in and $O(n^3)$ complexity. Iterative methods provide an alternative approach that can achieve $O(n^2)$ or even $O(n)$ complexity per iteration while maintaining sparsity.

This lecture covers the mathematical foundations of iterative methods, from classical stationary methods to modern Krylov subspace techniques, with emphasis on convergence theory and practical implementation considerations.

2 Matrix Splitting Framework

Definition 1 (Matrix Splitting)**.** *For a nonsingular matrix $A \in \mathbb{R}^{n\times n}$, a **matrix splitting** is a decomposition $A = M - N$ where M is nonsingular.*

The corresponding iterative method is:

$$Mx^{(k+1)} = Nx^{(k)} + b \tag{1}$$

$$x^{(k+1)} = \underbrace{M^{-1}N}_{G}x^{(k)} + M^{-1}b \tag{2}$$

where $G = M^{-1}N$ is the **iteration matrix**.

Theorem 1 (Convergence of Matrix Splitting Methods)**.** *The iterative method $x^{(k+1)} = Gx^{(k)} + c$ converges to the unique solution $x^* = (I - G)^{-1}c$ for any initial guess $x^{(0)}$ if and only if $\rho(G) < 1$, where $\rho(G) = \max_i |\lambda_i(G)|$ is the spectral radius.*

Proof. The error $e^{(k)} = x^{(k)} - x^*$ satisfies $e^{(k+1)} = Ge^{(k)}$, so $e^{(k)} = G^k e^{(0)}$. The method converges if and only if $\lim_{k\to\infty} G^k = 0$, which occurs if and only if $\rho(G) < 1$. □

3 Classical Iterative Methods

Consider the standard decomposition $A = D - L - U$ where:

- D = diagonal part of A
- $-L$ = strictly lower triangular part of A
- $-U$ = strictly upper triangular part of A

3.1 Jacobi Method

Definition 2 (Jacobi Method)**.** *The Jacobi method uses the splitting $M = D$, $N = L + U$:*

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n$$

The iteration matrix is $G_J = D^{-1}(L + U) = I - D^{-1}A$.

3.2 Gauss-Seidel Method

Definition 3 (Gauss-Seidel Method)**.** *The Gauss-Seidel method uses the splitting $M = D - L$, $N = U$:*

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

The iteration matrix is $G_{GS} = (D - L)^{-1}U$.

3.3 Successive Over-Relaxation (SOR)

Definition 4 (SOR Method)**.** *The SOR method introduces a relaxation parameter $\omega > 0$:*

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

The iteration matrix is $G_{SOR} = (D - \omega L)^{-1}((1 - \omega)D + \omega U)$.

Theorem 2 (Optimal SOR Parameter)**.** *For symmetric positive definite matrices with consistent ordering, the optimal relaxation parameter is:*

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho(G_J)^2}}$$

where G_J is the Jacobi iteration matrix.

4 Krylov Subspace Methods

Definition 5 (Krylov Subspace)**.** *For a matrix $A \in \mathbb{R}^{n \times n}$ and vector $v \in \mathbb{R}^n$, the m-th Krylov subspace is:*

$$\mathcal{K}_m(A, v) = \mathit{span}\{v, Av, A^2 v, \ldots, A^{m-1} v\}$$

Krylov methods seek approximate solutions in affine Krylov subspaces $x_0 + \mathcal{K}_m(A, r_0)$ where $r_0 = b - Ax_0$ is the initial residual.

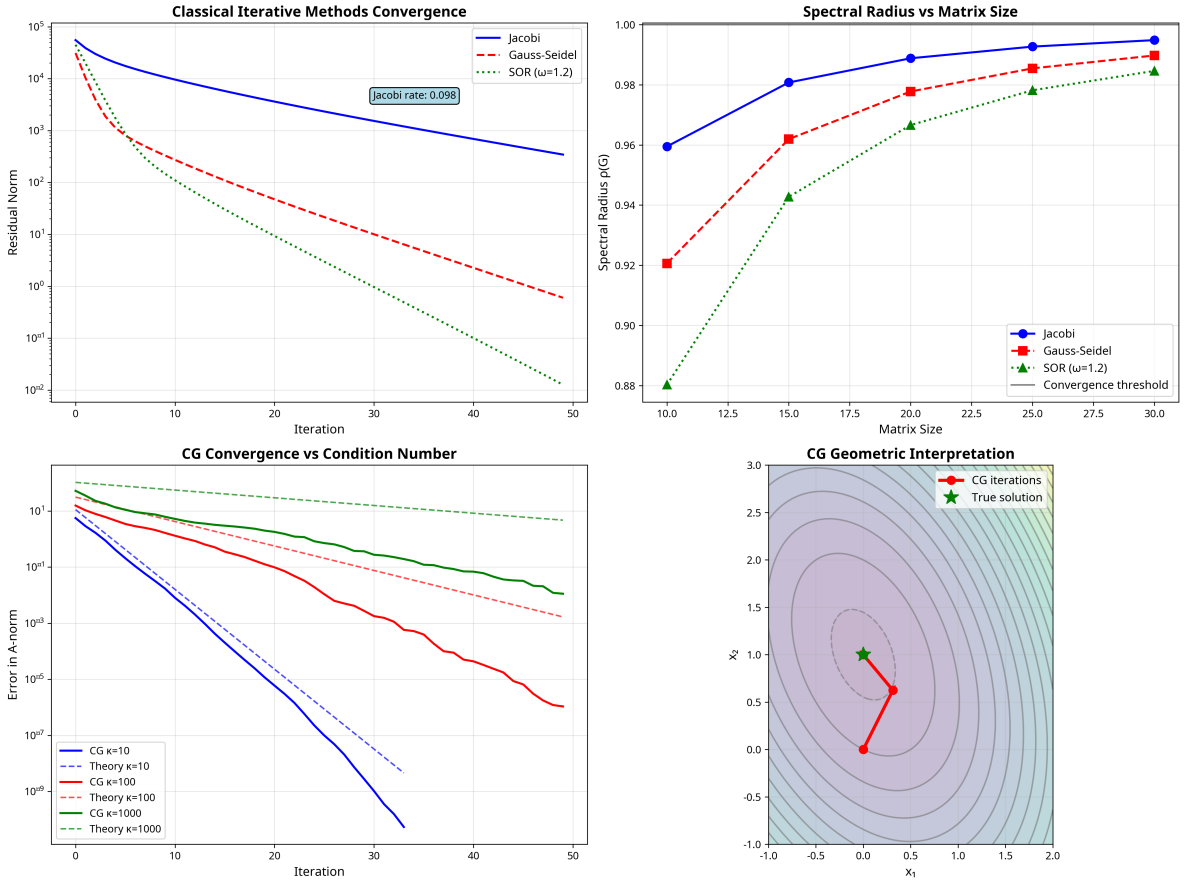

Figure 1: Comprehensive analysis of classical iterative methods showing convergence rates, spectral analysis, condition number effects, and geometric interpretation. The figure demonstrates how spectral radius determines convergence behavior and illustrates the geometric intuition behind the conjugate gradient method.

4.1 Conjugate Gradient Method

For symmetric positive definite (SPD) systems, the conjugate gradient (CG) method is optimal.

Theorem 3 (CG Optimality)**.** *For SPD matrix A, the CG iterate x_k minimizes the A-norm of the error:*

$$x_k = \arg \min_{x \in x_0 + \mathcal{K}_k(A, r_0)} \|x - x^*\|_A$$

where $\|x\|_A = \sqrt{x^T A x}$ is the energy norm.

Algorithm 1 Conjugate Gradient Method

Input: SPD matrix A, vector b, initial guess x_0, tolerance ϵ
$r_0 = b - Ax_0$, $p_0 = r_0$
for $k = 0, 1, 2, \ldots$ until $\|r_k\| < \epsilon$ **do**
  $\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$
  $x_{k+1} = x_k + \alpha_k p_k$
  $r_{k+1} = r_k - \alpha_k A p_k$
  $\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
  $p_{k+1} = r_{k+1} + \beta_k p_k$
end for
Output: Approximate solution x_{k+1}

Theorem 4 (CG Convergence Rate)**.** *For SPD matrix A with condition number $\kappa = \kappa(A) = \lambda_{\max}/\lambda_{\min}$:*

$$\|x_k - x^*\|_A \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \|x_0 - x^*\|_A$$

4.2 GMRES Method

For general nonsymmetric systems, GMRES (Generalized Minimal Residual) minimizes the residual norm.

Theorem 5 (GMRES Optimality)**.** *The GMRES iterate x_k minimizes the residual norm:*

$$x_k = \arg\min_{x \in x_0 + \mathcal{K}_k(A, r_0)} \|b - Ax\|_2$$

GMRES uses the Arnoldi process to build an orthonormal basis $V_k = [v_1, v_2, \ldots, v_k]$ for $\mathcal{K}_k(A, r_0)$ and solves the least squares problem:

$$\min_y \|\beta e_1 - \bar{H}_k y\|_2$$

where $\bar{H}_k$ is the $(k+1) \times k$ upper Hessenberg matrix from Arnoldi.

5 Preconditioning

Definition 6 (Preconditioning)**.** *Instead of solving $Ax = b$, solve the preconditioned system:*

$$M^{-1}Ax = M^{-1}b$$

where $M \approx A$ is chosen so that $M^{-1}A$ has better spectral properties than A.

5.1 Common Preconditioners

1. **Diagonal (Jacobi):** $M = \text{diag}(A)$
2. **Incomplete LU (ILU):** $M = \tilde{L}\tilde{U}$ where $\tilde{L}\tilde{U}$ is a sparse approximation to the LU factorization
3. **SSOR:** $M = \frac{\omega}{2-\omega}(D - \omega L)D^{-1}(D - \omega U)$

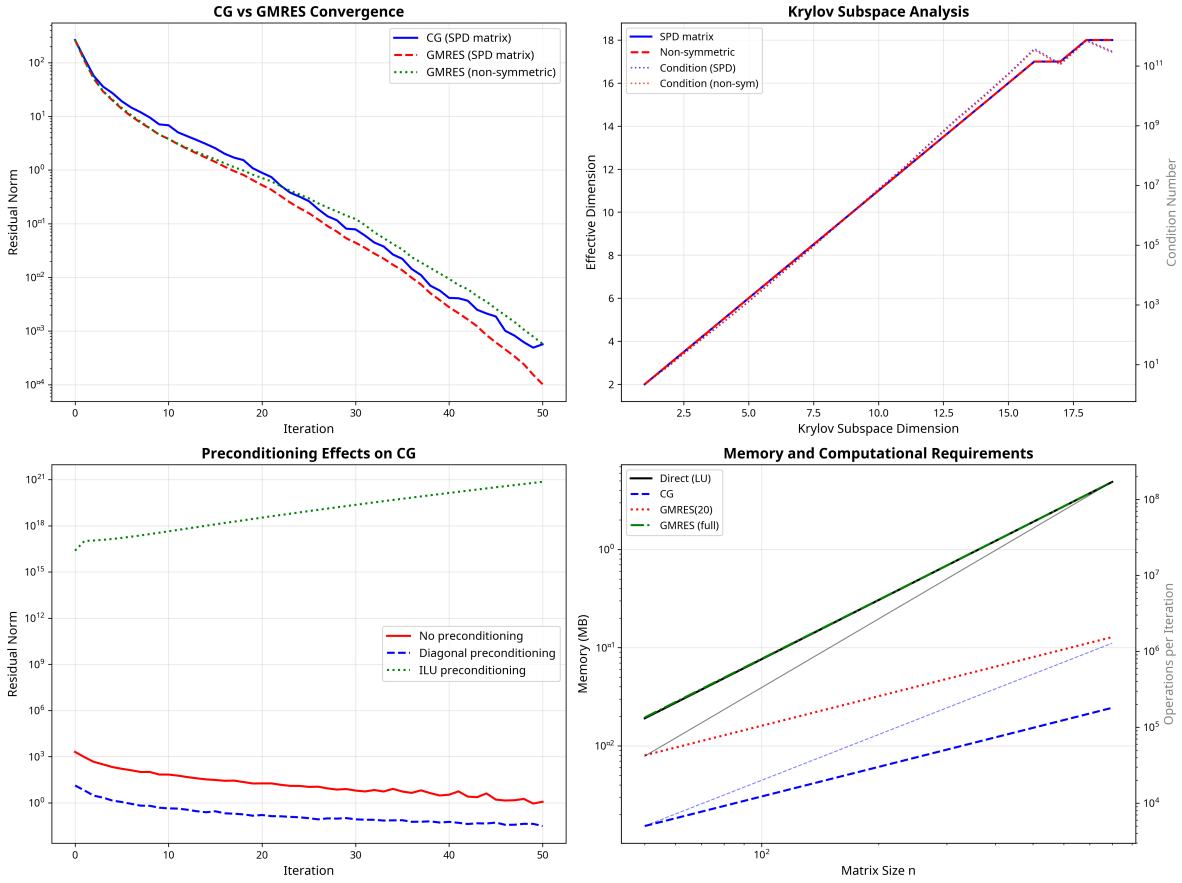

Figure 2: Analysis of Krylov subspace methods comparing CG and GMRES convergence behavior, subspace dimension growth, preconditioning effects, and computational requirements. The figure illustrates the superior convergence of CG for SPD systems and the general applicability of GMRES.

Theorem 6 (Effect of Preconditioning on CG)**.** *For preconditioned CG with SPD preconditioner M:*

$$\|x_k - x^*\|_A \leq 2 \left(\frac{\sqrt{\kappa(M^{-1}A)} - 1}{\sqrt{\kappa(M^{-1}A)} + 1} \right)^k \|x_0 - x^*\|_A$$

6 Multigrid Methods

Multigrid methods achieve optimal $O(n)$ complexity by exploiting the complementary properties of different solution components.

6.1 Two-Grid Principle

1. **Smoothing:** Apply a few iterations of a classical method (e.g., Gauss-Seidel) to reduce high-frequency error components

2. **Coarse Grid Correction:** Transfer the residual to a coarser grid, solve exactly, and transfer the correction back

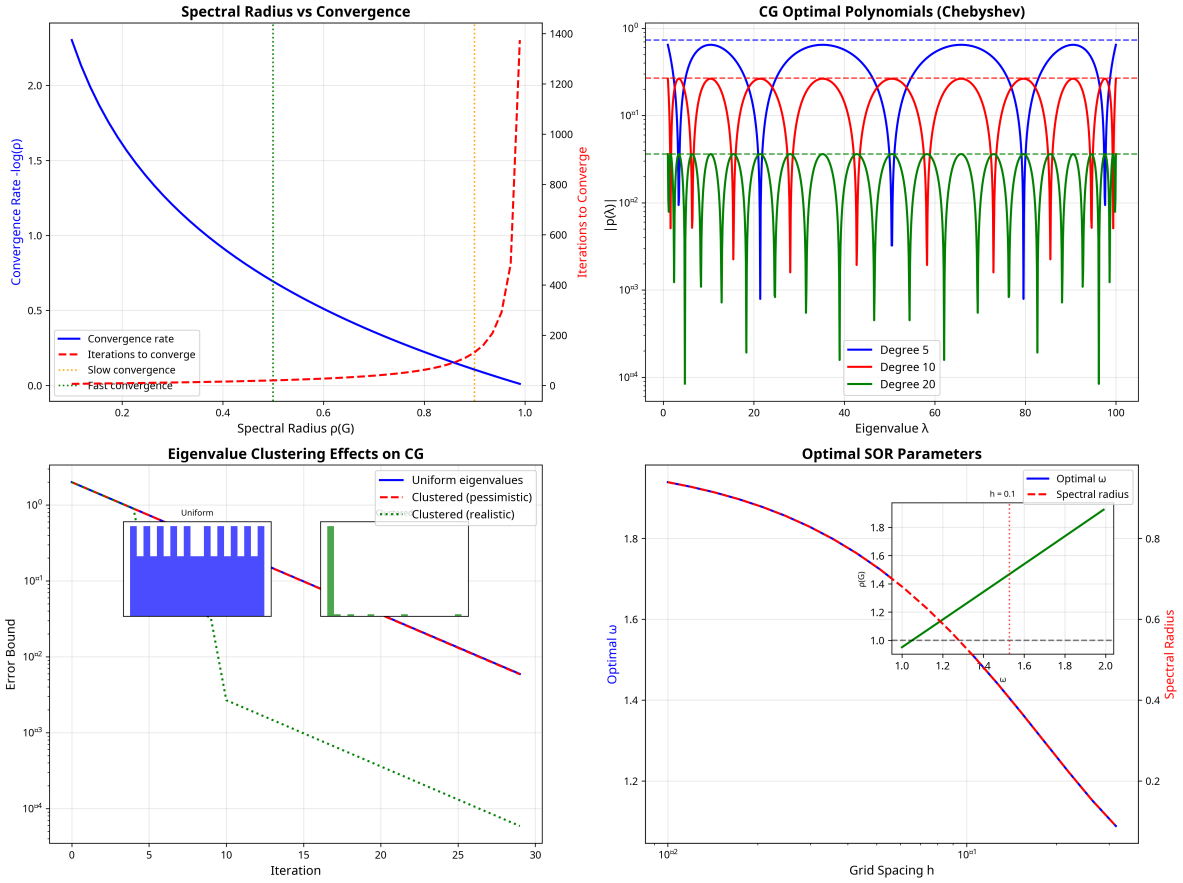

Figure 3: Theoretical convergence analysis showing spectral radius relationships, polynomial approximation theory, eigenvalue clustering effects, and optimal SOR parameters. The figure demonstrates how eigenvalue distribution affects convergence rates and the importance of preconditioning for clustering eigenvalues.

3. **Post-smoothing:** Apply additional smoothing iterations

Theorem 7 (Multigrid Convergence)**.** *Under appropriate smoothing and approximation properties, multigrid methods achieve convergence rates independent of the grid size, resulting in $O(n)$ total complexity.*

7 Practical Implementation

7.1 Method Selection Guidelines

- **Small dense systems ($n < 1000$):** Direct methods (LU, Cholesky)
- **Large SPD systems:** Preconditioned CG
- **Large general systems:** Preconditioned GMRES with restart
- **Elliptic PDEs:** Multigrid methods
- **Very large sparse systems:** Iterative methods with problem-specific preconditioners

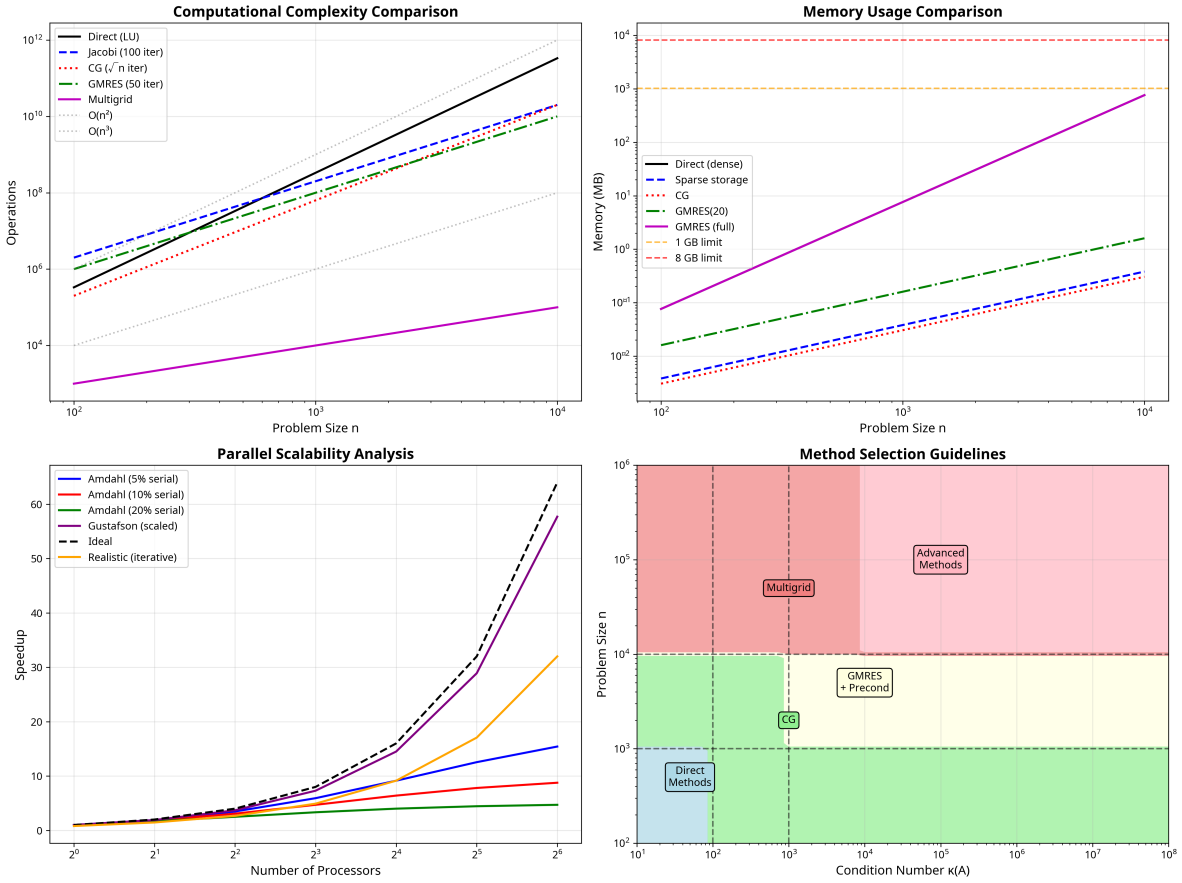

Figure 4: Practical implementation guide showing algorithm complexity comparison, memory usage patterns, parallel scalability analysis, and method selection guidelines. The figure provides decision trees for choosing appropriate methods based on problem characteristics and computational constraints.

7.2 Computational Considerations

- **Memory:** Classical methods require $O(n)$ storage, CG requires 4 vectors, GMRES requires $O(mn)$ for m iterations
- **Parallelization:** Matrix-vector products and inner products are easily parallelizable
- **Stopping criteria:** Use relative residual $\|r_k\|/\|r_0\| < \epsilon$ or energy norm for CG

7.3 Python Implementation Example

```
import numpy as np

def conjugate_gradient(A, b, x0, tol=1e-6, max_iter=1000):
    """
    Conjugate Gradient method for SPD systems

    Parameters:
    A: SPD matrix
    b: right-hand side vector
```

```
    x0: initial guess
    tol: convergence tolerance
    max_iter: maximum iterations

    Returns:
    x: solution vector
    iterations: number of iterations
    """
    x = x0.copy()
    r = b - A @ x
    p = r.copy()
    rsold = r.T @ r

    for k in range(max_iter):
        Ap = A @ p
        alpha = rsold / (p.T @ Ap)
        x = x + alpha * p
        r = r - alpha * Ap
        rsnew = r.T @ r

        if np.sqrt(rsnew) < tol:
            break

        beta = rsnew / rsold
        p = r + beta * p
        rsold = rsnew

    return x, k+1

# Example usage
n = 100
A = np.random.randn(n, n)
A = A.T @ A + np.eye(n)  # Make SPD
b = np.random.randn(n)
x0 = np.zeros(n)

x, iterations = conjugate_gradient(A, b, x0)
print(f"Converged in {iterations} iterations")
print(f"Residual norm: {np.linalg.norm(b - A @ x):.2e}")
```

8 Convergence Analysis Summary

Method	Convergence Rate	Memory	Conditions
Jacobi	$\rho(D^{-1}(L+U))$	$O(n)$	Diagonally dominant
Gauss-Seidel	$\rho((D-L)^{-1}U)$	$O(n)$	SPD or diag. dominant
SOR	$\omega_{opt}-1$	$O(n)$	Optimal ω
CG	$\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)$	$O(n)$	SPD matrix
GMRES	Problem dependent	$O(mn)$	Any nonsingular
Multigrid	$O(1)$	$O(n)$	Elliptic problems

Table 1: Comparison of iterative methods showing convergence rates, memory requirements, and applicability conditions.

The choice of iterative method depends critically on the matrix properties, problem size, and computational resources. Preconditioning is essential for practical performance on ill-conditioned systems, and modern implementations often combine multiple techniques for optimal efficiency.