Iterative Methods Cheat Sheet

Based on Prof. P. F. Antonietti, *Numerical Linear Algebra, A.Y. 2025–2026*

Jacobi Method

Applicable: A strictly diagonally dominant (by rows or columns).
Splitting: $A = D - E - F$, D diagonal.
Iteration:

$$x^{(k+1)} = D^{-1}(E+F)x^{(k)} + D^{-1}b$$

Iteration matrix: $B_J = D^{-1}(E+F)$.
Convergence: $\rho(B_J) < 1$; if A is strictly diagonally dominant $\Rightarrow$ convergence.
Cost: $\mathcal{O}(n^2)$ (or $\mathcal{O}(n)$ if A sparse).
Idea: Update all components using values from previous iteration.
Fully parallelizable.

Gauss–Seidel Method

Applicable: A SPD or strictly diagonally dominant.
Splitting: $A = (D-E) - F$.
Iteration:

$$x^{(k+1)} = (D-E)^{-1}Fx^{(k)} + (D-E)^{-1}b$$

Iteration matrix: $B_{GS} = (D-E)^{-1}F$.
Convergence: $\rho(B_{GS}) < 1$; sufficient if A SPD or strictly diagonally dominant.
Relation: For tridiagonal A, $\rho^2(B_J) = \rho(B_{GS})$.
Idea: Use new values $x_j^{(k+1)}$ as soon as computed.

Stationary Richardson Method

Applicable: A SPD, parameter $\alpha \in \mathbb{R}$.
Iteration:

$$x^{(k+1)} = (I - \alpha A)x^{(k)} + \alpha b$$

Iteration matrix: $B_\alpha = I - \alpha A$.
Convergence: $0 < \alpha < \frac{2}{\lambda_{\max}(A)}$.
Spectral radius:

$$\rho(B_\alpha) = \max_i |1 - \alpha\lambda_i(A)|.$$

Optimal parameter:

$$\alpha_{\text{opt}} = \frac{2}{\lambda_{\max}(A) + \lambda_{\min}(A)}, \quad \rho_{\text{opt}} = \frac{K(A)-1}{K(A)+1}.$$

Idea: Update along residual $r^{(k)} = b - Ax^{(k)}$ proportionally to its size.

Gradient Method (Steepest Descent)

Applicable: A SPD. Equivalent to minimizing $\Phi(x) = \frac{1}{2}x^TAx - b^Tx$.
Iteration:

$$r^{(k)} = b - Ax^{(k)}, \quad \alpha_k = \frac{r^{(k)T}r^{(k)}}{r^{(k)T}Ar^{(k)}}, \quad x^{(k+1)} = x^{(k)} + \alpha_k r^{(k)}.$$

Convergence: Same rate as Richardson with α_{opt}:

$$\rho_{\text{opt}} = \frac{K(A)-1}{K(A)+1}.$$

Idea: Direction of steepest descent of $\Phi(x)$.

Conjugate Gradient (CG)

Applicable: A SPD.
Idea: Directions $\boldsymbol{d}^{(k)}$ are A-conjugate: $(\boldsymbol{d}^{(i)}, A\boldsymbol{d}^{(j)}) = 0$ for $i \neq j$.
Algorithm:

$$\begin{aligned}
r^{(0)} &= b - Ax^{(0)}, \quad \boldsymbol{d}^{(0)} = r^{(0)}, \\
\alpha_k &= \frac{r^{(k)T}r^{(k)}}{\boldsymbol{d}^{(k)T}A\boldsymbol{d}^{(k)}}, \quad x^{(k+1)} = x^{(k)} + \alpha_k \boldsymbol{d}^{(k)}, \\
r^{(k+1)} &= r^{(k)} - \alpha_k A\boldsymbol{d}^{(k)}, \\
\beta_k &= \frac{r^{(k+1)T}r^{(k+1)}}{r^{(k)T}r^{(k)}}, \quad \boldsymbol{d}^{(k+1)} = r^{(k+1)} + \beta_k \boldsymbol{d}^{(k)}.
\end{aligned}$$

Error bound (in A-norm):

$$\frac{\|\vec{e}^{(k)}\|_A}{\|\vec{e}^{(0)}\|_A} \leq \frac{2\,c^k}{1+c^{2k}}, \quad c = \frac{\sqrt{K(A)}-1}{\sqrt{K(A)}+1}.$$

Convergence: Exact in $\leq n$ steps (in exact arithmetic).

BiConjugate Gradient (BiCG)

Applicable: General nonsymmetric A.
Idea: Two coupled systems $Ax = b$ and $A^T\hat{x} = \hat{b}$, residuals $r^{(k)}$, $\hat{r}^{(k)}$ biorthogonal in dual Krylov spaces.
Algorithm:

$$\begin{aligned}
r^{(0)} &= b - Ax^{(0)}, \quad \hat{r}^{(0)} = \hat{b} - A^T\hat{x}^{(0)}, \\
\boldsymbol{d}^{(0)} &= r^{(0)}, \quad \hat{\boldsymbol{d}}^{(0)} = \hat{r}^{(0)}, \\
\alpha_k &= \frac{\hat{r}^{(k)T}r^{(k)}}{\hat{\boldsymbol{d}}^{(k)T}A\boldsymbol{d}^{(k)}}, \\
x^{(k+1)} &= x^{(k)} + \alpha_k \boldsymbol{d}^{(k)}, \quad \hat{x}^{(k+1)} = \hat{x}^{(k)} + \alpha_k \hat{\boldsymbol{d}}^{(k)}, \\
r^{(k+1)} &= r^{(k)} - \alpha_k A\boldsymbol{d}^{(k)}, \quad \hat{r}^{(k+1)} = \hat{r}^{(k)} - \alpha_k A^T\hat{\boldsymbol{d}}^{(k)}, \\
\beta_k &= \frac{\hat{r}^{(k+1)T}r^{(k+1)}}{\hat{r}^{(k)T}r^{(k)}}, \\
\boldsymbol{d}^{(k+1)} &= r^{(k+1)} + \beta_k \boldsymbol{d}^{(k)}, \quad \hat{\boldsymbol{d}}^{(k+1)} = \hat{r}^{(k+1)} + \beta_k \hat{\boldsymbol{d}}^{(k)}.
\end{aligned}$$

Remarks: Requires multiplications by A and A^T; each iteration ≈ 2 CG costs.
Unstable $\Rightarrow$ BiCGSTAB improves stability.

GMRES

Applicable: Any square A. Projection method.
Idea: Find $x^{(k)} \in x^{(0)} + \mathcal{K}_k(A, r^{(0)})$ minimizing $\|b - Ax^{(k)}\|_2$.
Algorithm (Arnoldi):

1. Compute $\boldsymbol{q_k}$ with a suitable method
2. Build $\boldsymbol{Q_k}$ as the n x k matrix formed by $\boldsymbol{q_1}, \boldsymbol{q_2}, ..., \boldsymbol{q_k}$,
3. Find $\boldsymbol{y_k}$ which minimizes $\|\boldsymbol{r}^{(k)}\|_2$
4. Compute $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(0)} + Q_k\boldsymbol{y}^{(k)}$

Convergence: Nonstationary; monotone residual decrease in exact arithmetic.
If A SPD:

$$\|r^{(k)}\|_2 \leq \left[\frac{[K_2(A)]^2 - 1}{[K_2(A)]^2}\right]^{k/2}\|r^{(0)}\|_2,$$

If $A_s = \frac{(A+A^T)}{2}$ is SPD:

$$\|r^{(k)}\|_2 \leq \left[1 - \frac{\lambda^2_{\min}(A_S)}{\lambda_{\max}(A^TA)}\right]^{k/2}\|r^{(0)}\|_2,$$

Power Method

Applicable: A has a unique eigenvalue λ_1 of maximum modulus, $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$.
Algorithm:

$$\begin{aligned} \boldsymbol{y}^{(k+1)} &= A\boldsymbol{x}^{(k)}, \\ \boldsymbol{x}^{(k+1)} &= \frac{\boldsymbol{y}^{(k+1)}}{\|\boldsymbol{y}^{(k+1)}\|}, \\ \nu^{(k+1)} &= [\boldsymbol{x}^{(k+1)}]^H A\boldsymbol{x}^{(k+1)}. \end{aligned}$$

Convergence: $\boldsymbol{x}^{(k)} \to$ (multiple of eigenvector v_1), $\nu^{(k)} \to \lambda_1$.
Rate: depends on $\dfrac{|\lambda_2|}{|\lambda_1|}$:

$$\|\boldsymbol{e}^{(k)}\| = \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right).$$

Faster convergence if $|\lambda_2/\lambda_1|$ small.

Deflation Methods

Goal: Once (λ_1, v_1) is known, remove it to compute $\lambda_2, \ldots, \lambda_n$.
Idea: Build S s.t. $Sv_1 = \alpha e_1$ (e_1 first canonical vector). Then:

$$SAS^{-1} = \begin{pmatrix} \lambda_1 & b^T \\ 0 & B \end{pmatrix}, \quad B \in \mathbb{C}^{(n-1)\times(n-1)}.$$

Eigenvalues of B are $\lambda_2, \ldots, \lambda_n$. Compute them (e.g. via Power Method) iteratively.
Example: S can be chosen as a Householder transformation.

Inverse Power Method

Goal: Compute smallest eigenvalue of A.
Observation: Eigenvalues of A^{-1} are $1/\lambda_i$.
Algorithm:

$$\begin{aligned} A\boldsymbol{z}^{(k+1)} &= \boldsymbol{q}^{(k)}, \\ \boldsymbol{q}^{(k+1)} &= \frac{\boldsymbol{z}^{(k+1)}}{\|\boldsymbol{z}^{(k+1)}\|}, \\ \sigma^{(k+1)} &= [\boldsymbol{q}^{(k+1)}]^H A\boldsymbol{q}^{(k+1)}. \end{aligned}$$

Converges to the eigenvector associated with $\lambda_n = \min_i \lambda_i(A)$.

Inverse Power Method with Shift

Goal: Compute eigenvalue of A closest to given μ.
Idea: Let $M_\mu = A - \mu I$. Eigenvalues of M_μ^{-1} are $(\lambda_i - \mu)^{-1}$, so the largest eigenvalue of M_μ^{-1} corresponds to λ_i closest to μ.
Algorithm:

$$\begin{aligned} M_\mu\boldsymbol{z}^{(k+1)} &= \boldsymbol{q}^{(k)}, \\ \boldsymbol{q}^{(k+1)} &= \frac{\boldsymbol{z}^{(k+1)}}{\|\boldsymbol{z}^{(k+1)}\|}, \\ \nu^{(k+1)} &= [\boldsymbol{q}^{(k+1)}]^H A\boldsymbol{q}^{(k+1)}. \end{aligned}$$

Remarks: Shifting accelerates convergence when a good μ is chosen.

QR Factorization

Definition: $A = QR$ where Q orthogonal/unitary ($Q^TQ = I$), R upper triangular.
Reduced form: $A = \hat{Q}\hat{R}$ with $\hat{Q} \in \mathbb{R}^{m\times n}$, $\hat{R} \in \mathbb{R}^{n\times n}$.
Full form: $Q \in \mathbb{R}^{m\times m}$, $R \in \mathbb{R}^{m\times n}$.
Existence & uniqueness: Always exists for $A \in \mathbb{C}^{m\times n}$, $m \geq n$, unique if $\det(R) > 0$.

Gram-Schmidt Orthogonalisation

Given $A = [a_1, a_2, \ldots, a_n]$. **Classical GS:**

$$\begin{aligned} w_j &= a_j - \sum_{i=1}^{j-1} (q_i^T a_j) q_i, \\ q_j &= \frac{w_j}{\|w_j\|}, \quad r_{ij} = q_i^T a_j. \end{aligned}$$

Modified GS: replace a_j by w_j in inner products: more stable numerically.
Produces: $A = \hat{Q}\hat{R}$.

QR Algorithm (Basic Form)

Goal: Compute eigenvalues of A.
Algorithm:

1. $A^{(0)} = A$, $U^{(0)} = I$.
2. For $k = 1, 2, \ldots$:

$$\begin{aligned} A^{(k-1)} &= Q^{(k)}R^{(k)} \quad \text{(QR factorization)} \\ A^{(k)} &= R^{(k)}Q^{(k)} \\ U^{(k)} &= U^{(k-1)}Q^{(k)} \end{aligned}$$

3. Stop when subdiagonal entries $\to 0$.

Convergence: If $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$, then

$$a_{ij}^{(k)} = \mathcal{O}\left(\left(\frac{\lambda_i}{\lambda_j}\right)^k\right), \quad i > j.$$

Slow when eigenvalues are close. Cost per iteration: $\mathcal{O}(n^3)$.

QR Algorithm Improvements

Hessenberg Reduction: Transform A to upper Hessenberg $H = U^HAU$, so QR steps cost $\mathcal{O}(n^2)$. Structure preserved at each iteration.
Shifts: Accelerate convergence.
Deflation: Once a diagonal element converges, deflate the matrix.

Schur Decomposition & Schur Vectors

If $A \in \mathbb{C}^{n\times n}$, $\exists$ unitary U s.t.

$$U^HAU = T,$$

where T is upper triangular, with diagonal entries = eigenvalues of A. Columns of U are **Schur vectors**:

$$Au_k = \lambda_k u_k + \sum_{i=1}^{k-1} t_{ik}u_i, \quad Au_k \in \text{span}\{u_1, \ldots, u_k\}.$$

u_1 is an eigenvector; $\{u_1, \ldots, u_k\}$ form an invariant subspace. Schur decomposition is unique up to ordering of eigenvalues.

Over-determined linear systems

The mathematical problem is: given $\boldsymbol{A} \in \mathbb{R}$,$m \geq n$, and $\boldsymbol{b} \in \mathbb{R}^m$find $\boldsymbol{x} \in \mathbb{R}^n$ such that: $A\boldsymbol{x} = \boldsymbol{b}$

Solution in the least-square sense

Given $A \in \mathbb{R}^{m\times n}$, we say that $\boldsymbol{x}^* \in \mathbb{R}^n$ is a solution of the linear system $A\boldsymbol{x} = \boldsymbol{b}$ in the least-squares sense if

$$\Phi(\boldsymbol{x}^*) = \min_{y\in\mathbb{R}^n} \Phi(\boldsymbol{y}) \quad \text{where} \quad \Phi(\boldsymbol{y}) = \|A\boldsymbol{y} - \boldsymbol{b}\|_2^2$$

The solution can be found by imposing that the gradient of the function $\Phi()$ must be equal to zero at $\boldsymbol{x}^*$.
So $\boldsymbol{x}^*$ must be the solution of the square system:

$$A^T A\boldsymbol{x}^* = A^T\boldsymbol{b} \quad \text{System of normal equations}$$

Solving with QR factorization

Let $A \in \mathbb{R}^{m\times n}$, with $m \geq n$, be a full rank matrix. Then the unique solution in the least-square sense $\boldsymbol{x}^*$ of $A\boldsymbol{x} = \boldsymbol{b}$ is given by $\boldsymbol{x}^* = \hat{R}^{-1}\hat{Q}^T\boldsymbol{b}$, where $\hat{R} \in \mathbb{R}^{n\times n}$ and $\hat{Q} \in \mathbb{R}^{m\times n}$ are the matrices of the reduced QR factorization of A.
The minimum of $\Phi()$ is given by

$$\Phi(\boldsymbol{x}^*) = \sum_{i=n+1}^{m} [(Q^T b)_i]^2$$

Solution with SVD and Generalized Inverse

If A does not have full rank, the techniques based on normal equations or QR factorization fail. In this case, if $\boldsymbol{x}^*$ is a least-squares solution, then $\boldsymbol{x}^* + \boldsymbol{z}$, with $\boldsymbol{z} \in \ker(A)$, is also a solution. To ensure uniqueness, we choose the solution with **minimum Euclidean norm**, i.e.

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \|A\boldsymbol{x} - \boldsymbol{b}\|_2^2 \quad \text{subject to} \quad \|\boldsymbol{x}\|_2 \text{ minimal.}$$

Singular Value Decomposition (SVD)

Any matrix $A \in \mathbb{R}^{m\times n}$ can be written as

$$A = U\Sigma V^T,$$

where

- $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices;
- $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m\times n}$ with $p = \min(m, n)$;
- $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$ are the **singular values** of A.

Computation of the SVD:

- The singular values σ_i are the square roots of the eigenvalues of $A^T A$:
$$A^T A v_i = \sigma_i^2 v_i.$$
- The right singular vectors v_i are the normalized eigenvectors of $A^T A$.
- The left singular vectors are obtained as
$$u_i = \frac{1}{\sigma_i} A v_i.$$
- In practice, instead of forming $A^T A$ explicitly (which squares the condition number), one first reduces A to **bidiagonal form** using Householder reflections:
$$U_1^T A V_1 = B,$$
where B is upper bidiagonal.
- Then, a symmetric QR algorithm is applied to $B^T B$ (or directly to B) to extract the singular values and vectors.

Moore–Penrose Pseudo-Inverse

Let $A = U\Sigma V^T$ be the SVD of A. The **Moore–Penrose pseudo-inverse** of A is defined as

$$A^\dagger = V\Sigma^\dagger U^T,$$

where

$$\Sigma^\dagger = \text{diag}\left(\frac{1}{\sigma_1}, \ldots, \frac{1}{\sigma_r}, 0, \ldots, 0\right),$$

and $r = \text{rank}(A)$.
The least-squares solution with minimum norm is then given by:

$$\boldsymbol{x}^* = A^\dagger\boldsymbol{b}.$$

If A is full rank and square, the pseudo-inverse coincides with the classical inverse:

$$A^\dagger = A^{-1}.$$

Multigrid Methods

Main idea and observations

A multigrid (MG) method is an iterative algorithm of the form

$$\boldsymbol{x}^{(k+1)} = MG(\boldsymbol{x}^{(k)}), \quad k \geq 0,$$

designed to efficiently solve large, typically sparse, linear systems

$$A_h\boldsymbol{x}_h = \boldsymbol{b}_h,$$

arising from the discretization of PDEs.
Main idea:

- Standard smoothers (e.g. Jacobi, Gauss–Seidel) reduce high-frequency error components rapidly, but smooth (low-frequency) components converge slowly.
- On a coarser grid, these smooth error components appear more oscillatory and can be reduced effectively.
- The multigrid method exploits this by recursively transferring the residual to coarser grids for correction.

MG methods operate on a hierarchy of grids $\mathcal{T}_h, \mathcal{T}_{2h}, \mathcal{T}_{4h}, \ldots$, where coarse-grid problems approximate the fine-grid problem at a lower cost.

Correction scheme

Let $\boldsymbol{x}_h^{(k)}$ be the current approximation and $\boldsymbol{r}_h^{(k)} = \boldsymbol{b}_h - A_h\boldsymbol{x}_h^{(k)}$ the residual. The correction scheme proceeds as:

1. Perform ν_1 relaxation sweeps on $A_h\boldsymbol{x}_h = \boldsymbol{b}_h$.
2. Compute the residual $\boldsymbol{r}_h = \boldsymbol{b}_h - A_h\boldsymbol{x}_h$.
3. Restrict $\boldsymbol{r}_h$ to the coarse grid: $\boldsymbol{r}_{2h} = I_{2h}^h\boldsymbol{r}_h$.
4. Solve the coarse residual equation: $A_{2h}\boldsymbol{e}_{2h} = \boldsymbol{r}_{2h}$.
5. Interpolate the coarse-grid correction: $\boldsymbol{e}_h = I_h^{2h}\boldsymbol{e}_{2h}$.
6. Correct the fine-grid approximation:
$$\boldsymbol{x}_h^{(k+1)} = \boldsymbol{x}_h^{(k)} + \boldsymbol{e}_h.$$

Restriction and interpolation operators

Restriction (fine → coarse): $I_{2h}^h : \mathcal{T}_h \to \mathcal{T}_{2h}$, e.g.

$$(I_{2h}^h\boldsymbol{w}_h)_i = w_{h,2i}.$$

Interpolation (coarse → fine): $I_h^{2h} : \mathcal{T}_{2h} \to \mathcal{T}_h$, e.g. in 1D:

$$v_{h,i} = \begin{cases} v_{2h,i} & \text{if } i \text{ is a common node,} \\ \dfrac{v_{2h,i^-} + v_{2h,i^+}}{2} & \text{otherwise.} \end{cases}$$

Galerkin condition:

$$A_{2h} = I_{2h}^h A_h I_h^{2h}, \qquad I_{2h}^h = c(I_h^{2h})^T,$$

where c is a scalar constant (often $c = 1$ or $c = \frac{1}{2}$).

Two-grid correction scheme

One iteration of the two-grid method:

$$\boldsymbol{x}_h^{(k+1)} = MG(\boldsymbol{x}_h^{(k)}, \boldsymbol{b}_h, \nu_1, \nu_2)$$

1. ν_1 pre-smoothing iterations on $A_h\boldsymbol{x}_h = \boldsymbol{b}_h$.
2. Compute residual $\boldsymbol{r}_h = \boldsymbol{b}_h - A_h\boldsymbol{x}_h$.
3. Restrict: $\boldsymbol{r}_{2h} = I_{2h}^h \boldsymbol{r}_h$.
4. Solve coarse problem $A_{2h}\boldsymbol{e}_{2h} = \boldsymbol{r}_{2h}$.
5. Interpolate and correct: $\boldsymbol{e}_h = I_h^{2h}\boldsymbol{e}_{2h}$, $\boldsymbol{x}_h \leftarrow \boldsymbol{x}_h + \boldsymbol{e}_h$.
6. ν_2 post-smoothing iterations.

V-cycle (single iteration)

One iteration of the V-cycle recursively applies the two-grid correction across all grid levels. Given the system $A_h\boldsymbol{x}_h = \boldsymbol{b}_h$ and an initial guess $\boldsymbol{x}_h^{(k)}$, the iteration

$$\boldsymbol{x}_h^{(k+1)} = MGV(\boldsymbol{x}_h^{(k)}, \boldsymbol{b}_h, \nu_1, \nu_2, j)$$

proceeds as follows:

- **Pre-smoothing:** perform ν_1 iterations of a relaxation method (e.g. weighted Jacobi or Gauss–Seidel) $\Rightarrow \boldsymbol{y}_h^{(\nu_1)} = S_h^{\nu_1}(\boldsymbol{x}_h^{(k)}, \boldsymbol{b}_h)$.
- **Compute the residual:** $\boldsymbol{r}_h = \boldsymbol{b}_h - A_h\boldsymbol{y}_h^{(\nu_1)}$.
- **Restriction:** transfer residual to the coarse grid $\boldsymbol{r}_{2h} = I_{2h}^h\boldsymbol{r}_h$.
- **Coarse-grid correction:** if j is the coarsest level, solve $A_{2h}\boldsymbol{e}_{2h} = \boldsymbol{r}_{2h}$ directly; otherwise compute $\boldsymbol{e}_{2h} = MGV(\boldsymbol{0}, \boldsymbol{r}_{2h}, \nu_1, \nu_2, j-1)$ recursively.
- **Interpolation and correction:** $\boldsymbol{e}_h = I_h^{2h}\boldsymbol{e}_{2h}$, $\boldsymbol{y}_h^{(\nu_1+1)} = \boldsymbol{y}_h^{(\nu_1)} + \boldsymbol{e}_h$.
- **Post-smoothing:** perform ν_2 additional relaxation steps $\Rightarrow \boldsymbol{y}_h^{(\nu_1+1+\nu_2)} = S_h^{\nu_2}(\boldsymbol{y}_h^{(\nu_1+1)}, \boldsymbol{b}_h)$.
- **Update solution:** $\boldsymbol{x}_h^{(k+1)} \leftarrow \boldsymbol{y}_h^{(\nu_1+1+\nu_2)}$.

Typical choice: $\nu_1 = \nu_2 = 1, 2$, or 3. Variants: V-cycle, W-cycle, and F-cycle (μ-cycle family).

Computational costs and convergence analysis

Storage cost: For spatial dimension d, the total storage is approximately

$$\text{cost} \approx \frac{2n^d}{1-2^{-d}} \text{ (in units of fine-grid unknowns).}$$

For $d = 1$: ≈ 2, and for $d = 2$: $\approx 4/3$ of the fine-grid problem.
Computational cost: Measured in Work Units (WU) = cost of one relaxation on the finest grid.

$$\text{Cost of V-cycle} \approx \frac{2}{1-2^{-d}} \text{ WU.}$$

Typical costs: ≈ 4 WU in 1D, 8 in 2D, and 16 in 3D.
Convergence: The convergence factor $\rho(MG) < 1$ is

$$\rho(MG) = \text{constant independent of } h \text{ and } n.$$

Hence, multigrid achieves optimal computational complexity:

$$O(n^d \log(n))$$

Algebraic Multigrid methods

Principle of AMG

AMG is based on MG principles but uses matrix coefficients instead of geometric information.
Classic AMG is based on the observation that the algebraic smooth error varies slowly in the direction of the matrix's relatively large (negative) coefficients.

Strong Connection

Given a threshold $\theta \in (0, 1)$, we say that i is strongly connected with j if

$$-a_{i,j} \geq \theta \max_{k \neq i}(-a_{i,k})$$

We can denote by S_i the set of vertices that i is strongly connected to by:

$$S_i = \{j \in N_i : i\text{strongly connects to } j\} \quad \text{where} \quad N_i = \{j \neq i : a_{i,j} \neq 0\}$$

This gives us the strngth matrix S, with S_i as its i-th row.

Standard coarsening

Algebraic smooth error veries slowly in the direction of strong connections, so we course in the direction of strong connections. This results in a C/F-splitting such that F-vertices strongly connect to neighboring C-vertices. Then the idea is to represent the values of a smooth vector at F-vertices as a linear combination of the values at C-vertices.
Algorithm:

1. Choose an independent set of C-vertices based on the graph of S;
2. Choose additional C-vertices in order to satisfy the interpolation requirements.

In step (1) of the algorithm, the independent set can be choosed like this:

1. Start with a first vertex i to become a C-vertex
2. Then all vertices j which i strongly connect to become F-vertices
3. Next, another vertex from undecided vertices becomes a C-vertex, and all vertices which are strongly connected to it become F-vertices.
4. This procedure is repeated until all vertices become C or F-vertices.

Interpolation step

Let $e = (e_1, e_2, ..., e_i, ...)$ be the error.
A simple characterization of algebraic smooth error is $A\boldsymbol{e} \approx 0$.

So,

$$a_{i,i}e_i + \sum_{j \in N_i} a_{i,j}e_j \approx 0, \quad i \in F \quad (1)$$

We want to choose proper $w_{i,j}$ such that:

$$e_i \approx \sum_{j \in C} w_{i,j}e_j, \quad i \in F.$$

We define, for $i \in F$:

C_i : C-points strongly connected to i, $\quad (C_i = C \cap N_i)$,
$C_i^S = C \cap S_i$,
F_i : F-points strongly connected to i, $\quad (F_i = F \cap N_i)$,
N_i^W : all points weakly connected to i,
$(N_i^W = N_i \setminus (C_i^S \cup F_i^S))$

We can rewrite equation (1) as follows (replacing $\approx$ with $=$):

$$a_{i,i}e_i + \alpha \sum_{j \in C_i^S} a_{i,j}e_j = 0, \quad \alpha = \frac{\sum_{j \in N_i} a_{i,j}}{\sum_{j \in C_i^S} a_{i,j}}$$

So the direct formula is:

$$w_{i,j} = \alpha \frac{a_{i,j}}{a_{i,i}}, \quad \alpha = \frac{\sum_{j \in N_i} a_{i,j}}{\sum_{j \in C_i^S} a_{i,j}}$$

Domain Decomposition Methods

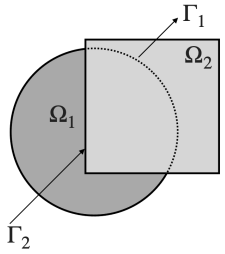

Figure 1: PDE in the form $Lu = f$ in $\Omega = \Omega_1 \cup \Omega_2$

Alternating Schwarz Method

Idea: Solve smaller problems on each subdomain iteratively, using the most recent solution from the neighboring subdomain to update the interface values.
Algorithm:

1. Start with an initial guess $u^{(0)}$ for the solution on the whole domain.
2. **Step 1: Solve on subdomain Ω_1.** Use the boundary conditions on the external boundary of Ω_1, and take the current solution from Ω_2 on the overlap region. This gives an updated solution in Ω_1.
3. **Step 2: Solve on subdomain Ω_2.** Use the boundary conditions on the external boundary of Ω_2, and take the newly computed solution from Ω_1 on the overlap region. This updates the solution in Ω_2.
4. **Step 3: Update the global solution.** Combine the updated solutions from Ω_1 and Ω_2 to form a new approximation over the whole domain.
5. Repeat steps 1–3 until the solution converges.

Intuition: Each subdomain problem is easier to solve than the full domain problem. By exchanging interface information iteratively, the solutions on each subdomain "communicate" and eventually converge to the solution on the full domain. The overlap ensures that information propagates between subdomains effectively.

Discretized Schwarz Method Setup

After discretization, we obtain an $n \times n$ symmetric positive definite linear system:

$$Ax = b, \quad A \in \mathbb{R}^{n \times n}, \quad x, b \in \mathbb{R}^n.$$

Let $\Omega = \Omega_1 \cup \Omega_2$ be split into two overlapping subdomains. For $i = 1, 2$, let

$$S_i \subset \{1, 2, \ldots, n\}$$

be the set of indices of interior grid points in Ω_i, with $n_i = |S_i|$. Because the subdomains overlap:

$$S_1 \cap S_2 \neq \emptyset \quad \text{and} \quad n_1 + n_2 > n.$$

Restriction matrices: For $i = 1, 2$, define the Boolean restriction matrix $R_i \in \mathbb{R}^{n_i \times n}$ such that, for any global vector $v \in \mathbb{R}^n$, the restricted vector

$$v_i = R_i v \in \mathbb{R}^{n_i}$$

contains only the components of v corresponding to indices in S_i.
Extension matrices: The transpose $R_i^T \in \mathbb{R}^{n \times n_i}$ expands a subdomain vector $v_i \in \mathbb{R}^{n_i}$ back to the global vector:

$$v = R_i^T v_i \in \mathbb{R}^n,$$

where the components corresponding to S_i are equal to v_i and all other components are zero.
Subdomain matrices: The principal submatrices corresponding to the subdomains are

$$A_1 = R_1 A R_1^T, \quad A_2 = R_2 A R_2^T, \quad A_i \in \mathbb{R}^{n_i \times n_i}.$$

These matrices are used in the Schwarz method to solve local problems on each subdomain.

Multiplicative Schwarz Method

Iteration:

$$\boldsymbol{x}^{(k+\frac{1}{2})} = \boldsymbol{x}^{(k)} + R_1^T A_1^{-1} R_1 (\boldsymbol{b} - A\boldsymbol{x}^{(k)}) \tag{1}$$
$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k+\frac{1}{2})} + R_2^T A_2^{-1} R_2 (\boldsymbol{b} - A\boldsymbol{x}^{(k+\frac{1}{2})}) \tag{2}$$

The error $\boldsymbol{e}^{(k)} = \boldsymbol{x} - \boldsymbol{x}^{(k)}$ updates as $\boldsymbol{e}^{(k+1)} = B_{MS}\boldsymbol{e}^{(k)}$, where:

$$B_{MS} = (I - R_2^T A_2^{-1} R_2 A)(I - R_1^T A_1^{-1} R_1 A)$$

Additive Schwarz Method

Iteration:

$$\boldsymbol{x}^{(k+\frac{1}{2})} = \boldsymbol{x}^{(k)} + R_1^T A_1^{-1} R_1 (\boldsymbol{b} - A\boldsymbol{x}^{(k)}) \tag{3}$$
$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k+\frac{1}{2})} + R_2^T A_2^{-1} R_2 (\boldsymbol{b} - A\boldsymbol{x}^{(k)}) \tag{4}$$

The error $\boldsymbol{e}^{(k)} = \boldsymbol{x} - \boldsymbol{x}^{(k)}$ updates as $\boldsymbol{e}^{(k+1)} = B_{AS}\boldsymbol{e}^{(k)}$, where:

$$B_{AS} = (R_2^T A_2^{-1} R_2 + R_1^T A_1^{-1} R_1) A$$

Schwarz Methods as preconditioners

If we eliminate the middle steps between $\boldsymbol{x}^{(k)}$ and $\boldsymbol{x}^{(k+1)}$ we find:
For the additive method:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + P_{ad}^{-1} \boldsymbol{r}^{(k)} \quad k \geq 0$$

with $P_{ad}^{-1} = (R_2^T A_2^{-1} R_2 + R_1^T A_1^{-1} R_1)$.
For the multiplicative method (adding an additional step with A_1^{-1} each iteration:

$$\boldsymbol{x}^{(k+\frac{1}{3})} = \boldsymbol{x}^{(k)} + R_1^T A_1^{-1} R_1 (\boldsymbol{b} - A\boldsymbol{x}^{(k)}) \tag{5}$$
$$\boldsymbol{x}^{(k+\frac{2}{3})} = \boldsymbol{x}^{(k+\frac{1}{2})} + R_2^T A_2^{-1} R_2 (\boldsymbol{b} - A\boldsymbol{x}^{(k+\frac{1}{3})}) \tag{6}$$
$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k+\frac{2}{3})} + R_1^T A_1^{-1} R_1 (\boldsymbol{b} - A\boldsymbol{x}^{(k+\frac{2}{3})}) \tag{7}$$

Preconditioners for Many Overlapping Subdomains

To achieve a higher degree of parallelism in the Schwarz method, we can apply the two-domain algorithm recursively or use many subdomains. However, as the number of subdomains p increases, convergence tends to degrade.
Additive Schwarz: The convergence rate can be restored by introducing a coarse grid correction that provides global coupling. The resulting *additive Schwarz preconditioner* takes the form:

$$P_{ad} = \sum_{i=0}^{p} R_i^T A_i^{-1} R_i.$$

In this formulation, all subdomain problems are solved *independently and in parallel*, making the additive method well-suited for parallel computing architectures.

Multiplicative Schwarz: The *multiplicative Schwarz iteration* for p subdomains is defined analogously to the two-domain case. Multiplicative Schwarz method typically converges faster than the additive one, but still requires a coarse grid correction for scalability.
The p subproblems must be solved sequentially within each iteration.
To improve parallelism, *coloring techniques* are used to identify groups of independent subdomains that can be solved simultaneously.
However, the degree of achievable parallelism depends on the number of subdomains per color.

Non Overlapping Subdomains

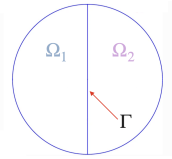

Figure 2: Non overlapping subdomains

We partition indices into three sets:

- S_1 corresponding to the interior nodes in Ω_1
- S_2 corresponding to the interior nodes in Ω_1
- S_Γ corresponding to the interface nodes in Γ

We obtain a symmetric block linear system:

$$\begin{pmatrix} A_{11} & 0 & A_{1\Gamma} \\ 0 & A_{22} & A_{2\Gamma} \\ A_{1\Gamma}^T & A_{2\Gamma}^T & A_{\Gamma\Gamma} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_\Gamma \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \boldsymbol{b}_\Gamma \end{pmatrix}$$

The LU factorization of A is given by

$$A = \begin{pmatrix} A_{11} & 0 & A_{1\Gamma} \\ 0 & A_{22} & A_{2\Gamma} \\ A_{1\Gamma}^T & A_{2\Gamma}^T & A_{\Gamma\Gamma} \end{pmatrix},$$

and can be factorized as

$$A = \underbrace{\begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ A_{1\Gamma}^T A_{11}^{-1} & A_{2\Gamma}^T A_{22}^{-1} & I \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} A_{11} & 0 & A_{1\Gamma} \\ 0 & A_{22} & A_{2\Gamma} \\ 0 & 0 & S \end{pmatrix}}_{U},$$

where the Schur complement S is

$$S = A_{\Gamma\Gamma} - A_{1\Gamma}^T A_{11}^{-1} A_{1\Gamma} - A_{2\Gamma}^T A_{22}^{-1} A_{2\Gamma}.$$

The Schur Complement System

Once the block LU factorization has been performed, we can express the interface problem in terms of the *Schur complement system*.

$$S\boldsymbol{x}_\Gamma = \tilde{\boldsymbol{b}}_\Gamma,$$

where

$$\tilde{\boldsymbol{b}}_\Gamma = \boldsymbol{b}_\Gamma - A_{1\Gamma}^T A_{11}^{-1} \boldsymbol{b}_1 - A_{2\Gamma}^T A_{22}^{-1} \boldsymbol{b}_2.$$

After computing the interface unknowns $\boldsymbol{x}_\Gamma$, the remaining subdomain unknowns can be recovered independently as

$$\boldsymbol{x}_1 = A_{11}^{-1}(\boldsymbol{b}_1 - A_{1\Gamma}\boldsymbol{x}_\Gamma), \qquad \boldsymbol{x}_2 = A_{22}^{-1}(\boldsymbol{b}_2 - A_{2\Gamma}\boldsymbol{x}_\Gamma).$$

These two local problems can be solved simultaneously once the interface solution is known.

Remarks:

- The Schur complement matrix S is generally *dense*, even when A is sparse, and is expensive to compute explicitly.
- In practice, if the Schur complement system

  $$S\boldsymbol{x}_\Gamma = \tilde{\boldsymbol{b}}_\Gamma$$

  is solved *iteratively*, then S need not be formed explicitly.
- A matrix-vector product with S implicitly involves solving local problems with A_{11}^{-1} and A_{22}^{-1}, which can be performed in parallel on each subdomain.
- The condition number of S is typically better than that of A, often scaling as $\mathcal{O}(h^{-1})$ instead of $\mathcal{O}(h^{-2})$ for mesh size h.
- Suitable *interface preconditioners* are still required to accelerate convergence of the iterative method.

Extension to Many Non-Overlapping Subdomains

To improve parallelism, the domain can be partitioned into many non-overlapping subdomains:

$$\Omega = \bigcup_{i=1}^{p} \Omega_i, \qquad \Gamma = \text{set of interface nodes.}$$

Let I be the set of interior node indices across all subdomains. The discrete system can then be written in block form:

$$\begin{pmatrix} A_{II} & A_{I\Gamma} \\ A_{I\Gamma}^T & A_{\Gamma\Gamma} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_I \\ \boldsymbol{x}_\Gamma \end{pmatrix} = \begin{pmatrix} \boldsymbol{b}_I \\ \boldsymbol{b}_\Gamma \end{pmatrix}.$$

The matrix A_{II} is block diagonal:

$$A_{II} = \begin{pmatrix} A_{11} & 0 & \cdots & 0 \\ 0 & A_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{pp} \end{pmatrix}.$$

Applying block LU factorization again yields the reduced Schur complement system:

$$S\boldsymbol{x}_\Gamma = \tilde{\boldsymbol{b}}_\Gamma,$$

where

$$S = A_{\Gamma\Gamma} - A_{I\Gamma}^T A_{II}^{-1} A_{I\Gamma}, \qquad \tilde{\boldsymbol{b}}_\Gamma = \boldsymbol{b}_\Gamma - A_{I\Gamma}^T A_{II}^{-1} \boldsymbol{b}_I.$$

This system can be solved iteratively without forming S explicitly. The interior unknowns can then be recovered via

$$\boldsymbol{x}_I = A_{II}^{-1}(\boldsymbol{b}_I - A_{I\Gamma}\boldsymbol{x}_\Gamma).$$

Because A_{II} is block diagonal, the computations involving A_{II}^{-1} can be performed on all subdomains *in parallel*.

Summary:

- The Schur complement system represents the coupling on the interface between subdomains.
- Its solution provides the interface unknowns $\boldsymbol{x}_\Gamma$.
- Once $\boldsymbol{x}_\Gamma$ is known, interior subdomain solutions can be computed independently.
- Parallelism is achieved because the inverses of the local matrices A_{ii} (blocks of A_{II}) are independent.

Direct Methods

0.1 LU factorization

Let A be a square matrix. An LU factorization refers to the factorization of A into two factors: a lower unitary trinagular matrix L and an upper triangular matrix U : $A = LU$
Theorem: If A is invertible, then it admits an LU factorization if and only if all its leading principal minors are nonzero.

Gaussian elimination

Algorithm:

1. For $k = 1, \ldots, n-1$:
   (a) For $i = k+1, \ldots, n$:
      i. Compute the multiplier:

$$l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$$

      ii. For $j = k+1, \ldots, n$, update the matrix entries:

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)}$$

      iii. Update the right-hand side:

$$b_i^{(k+1)} = b_i^{(k)} - l_{ik} b_k^{(k)}$$

After $n-1$ steps, we have:

$$A^{(n)} = U, \quad l_{ij} \to L, \quad \boldsymbol{b}^{(n)} = \boldsymbol{y}.$$

Total flops: $\sim \frac{2}{3}n^3$ **Sufficient conditions:**

- A strictly diagonally dominant by rows/columns
- A is SPD

Cholesky factorisation (A is SPD)

Let A be a SPD of order n. Then, there exists a unique upper triangular matrix R with real and positive diagonal entries such that $A = R^T R$
Algorithm: Let $r_{11} = \sqrt{a_{11}}$. For $j = 2, ..., n$

$$r_{ij} = \frac{1}{r_{ii}}(a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj}), \quad i = 1, ..., j-1 \tag{8}$$

$$r_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2} \tag{9}$$

Computational cost: $\frac{n^3}{3}$