Lec 13: Iterative Linear Solver

15-369/769: Numerical Methods (Foundations, ML & Visual Computing)

Instructor: Minchen Li

Iterative Linear Solver

Motivation: Iterative Methods for Linear Systems

- We revisit solving $A\mathbf{x} = \mathbf{b}$, now via **iterative** methods.
- Reformulate as a minimization problem:

$$\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2^2, \quad \text{or alternatively,} \quad \min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T A\mathbf{x} - b^T\mathbf{x} \text{ when } A \text{ is SPD}$$

- Motivation:
  - Direct methods (LU, QR, Cholesky) $\Rightarrow O(n^3)$ time and large memory.
  - Iterative methods exploit sparsity and require only matrix-vector products ($A\mathbf{v}$).
- Iterative solvers are crucial for large-scale systems where exact solutions are expensive.

Iterative Linear Solver

When to Use Iterative Methods

- **Sparse matrices:** Gaussian elimination induces *fill-in* — intermediate steps create nonzeros in originally empty entries.

$$\text{Storage cost: } O(n^2) \text{ (dense) vs. } O(n) \text{ (sparse format).}$$

- **Efficiency:** Each iteration only computes $A\mathbf{v}$ — time proportional to nonzeros in A.
- **Goal:** Often prefer approximate but fast solutions.
- **Example:** In optimization or simulation, a fast approximate linear solve per iteration is sufficient.

Table of Content

- Gradient Descent
- Conjugate Gradients
- Preconditioning
- Other Iterative Algorithms

Table of Content

- Gradient Descent
- Conjugate Gradients
- Preconditioning
- Other Iterative Algorithms

Gradient Descent

Problem Setup

We study the case $A\mathbf{x} = \mathbf{b}$ where:

1. $A \in \mathbb{R}^{n \times n}$ is square.
2. A is symmetric: $A^\top = A$.
3. A is positive definite: $\mathbf{x}^\top A\mathbf{x} > 0$ for all $\mathbf{x} \neq 0$.

- Then A defines a convex quadratic function:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{b}^\top \mathbf{x} + c.$$

- Gradient: $\nabla f(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$.
- Solving $\nabla f(\mathbf{x}) = 0 \Rightarrow A\mathbf{x} = \mathbf{b}$.

Gradient Descent

The Algorithm

1. Compute search direction:
$$\mathbf{d}_k = -\nabla f(\mathbf{x}_{k-1}) = \mathbf{b} - A\mathbf{x}_{k-1}.$$
2. Update:
$$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{d}_k,$$
where α_k ensures $f(\mathbf{x}_k) < f(\mathbf{x}_{k-1})$, computed via line search.
3. Repeat until convergence.

Intuition: Move along the steepest descent direction of $f(\mathbf{x})$, adjusting the step size α_k to minimize f along that direction.

Gradient Descent

Optimal Step Size

For quadratic $f(\mathbf{x})$, define:

$$g(\alpha) = f(\mathbf{x} + \alpha\mathbf{d}) = \frac{1}{2}(\mathbf{x} + \alpha\mathbf{d})^\top A(\mathbf{x} + \alpha\mathbf{d}) - \mathbf{b}^\top(\mathbf{x} + \alpha\mathbf{d}) + c.$$

Differentiate w.r.t. α:

$$\frac{dg}{d\alpha} = \alpha\mathbf{d}^\top A\mathbf{d} + \mathbf{d}^\top(A\mathbf{x} - \mathbf{b}).$$

Set derivative to zero $\Rightarrow$

$$\alpha = \frac{\mathbf{d}^\top(\mathbf{b} - A\mathbf{x})}{\mathbf{d}^\top A\mathbf{d}}.$$

For $\mathbf{d}_k = \mathbf{b} - A\mathbf{x}_k$:

$$\boxed{\alpha_k = \frac{\|\mathbf{d}_k\|_2^2}{\mathbf{d}_k^\top A\mathbf{d}_k}.}$$

Gradient Descent

Convergence Analysis

function LINEAR-GRADIENT-DESCENT($A, \mathbf{b}$)
  $\mathbf{x} \leftarrow \mathbf{0}$
  for $k \leftarrow 1, 2, 3, \ldots$
    $\mathbf{d} \leftarrow \mathbf{b} - A\mathbf{x}$ ▷ Search direction is residual
    $\alpha \leftarrow \frac{\|\mathbf{d}\|_2^2}{\mathbf{d}^\top A\mathbf{d}}$ ▷ Line search formula
    $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{d}$ ▷ Update solution vector $\mathbf{x}$

- Gradient descent always decreases $f(\mathbf{x}_k)$:

$$f(\mathbf{x}_k) < f(\mathbf{x}_{k-1}).$$

- Define relative error ratio:

$$R_k := \frac{f(\mathbf{x}_k) - f(\mathbf{x}^*)}{f(\mathbf{x}_{k-1}) - f(\mathbf{x}^*)}.$$

- If $R_k < \beta < 1$ for some fixed β (possibly depending on A), then $f(\mathbf{x}_k) \to f(\mathbf{x}^*)$ as $k \to \infty$.

Gradient Descent

Convergence Analysis: Derivation of R_k

Expand one step of iteration:

$$f(\mathbf{x}_k) = f(\mathbf{x}_{k-1}) - \frac{(\mathbf{d}_k^\top \mathbf{d}_k)^2}{2\mathbf{d}_k^\top A\mathbf{d}_k}.$$

Also, from $A\mathbf{x}^* = \mathbf{b}$:

$$f(\mathbf{x}_{k-1}) - f(\mathbf{x}^*) = \frac{1}{2}\mathbf{d}_k^\top A^{-1}\mathbf{d}_k.$$

Substitute and simplify:

$$R_k = 1 - \frac{(\mathbf{d}_k^\top \mathbf{d}_k)^2}{\mathbf{d}_k^\top A\mathbf{d}_k\, \mathbf{d}_k^\top A^{-1}\mathbf{d}_k}.$$

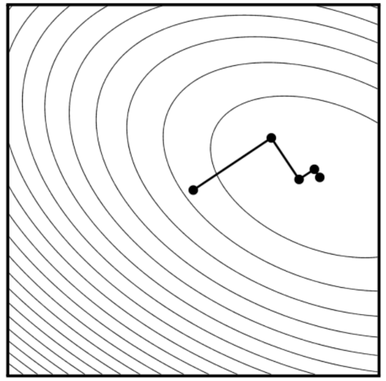
(The magnitude of $\mathbf{d}_k$ does not matter here, and $\max_{\|\mathbf{d}\|=1} \mathbf{d}^T A\mathbf{d} = \sigma_{max}$.)

Using eigenvalue bounds:

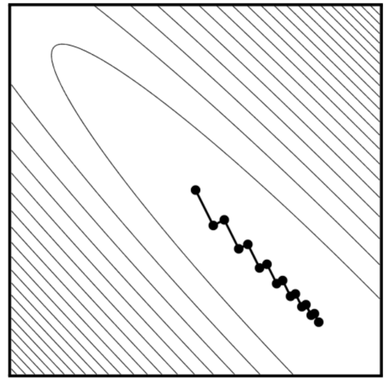
$$R_k \leq 1 - \frac{\sigma_{\min}}{\sigma_{\max}} = 1 - \frac{1}{\mathrm{cond}(A)}.$$

Gradient Descent

Conditioning and Convergence Speed



Well conditioned A



Poorly conditioned A

The convergence rate of gradient descent depends on the conditioning of A.

$$R_k = 1 - \frac{1}{\text{cond}(A)}, \quad \text{where } \text{cond}(A) = \frac{\sigma_{\text{max}}}{\sigma_{\text{min}}}.$$

- **Well-conditioned** A ($\text{cond}(A) \approx 1$): fast convergence.
- **Poorly-conditioned** A: slow convergence (zig-zagging along narrow valleys).
- Gradient descent always converges for SPD A, but rate deteriorates with high $\text{cond}(A)$.

Table of Content

- Gradient Descent
- Conjugate Gradients
- Preconditioning
- Other Iterative Algorithms

Conjugate Gradients

Motivation

- Solving $A\mathbf{x} = \mathbf{b}$ directly (e.g., Gaussian elimination) takes $O(n^3)$ time.
- Gradient descent takes $O(n^2)$ per iteration due to matrix-vector products.
- If gradient descent requires $> n$ iterations, it becomes slower than direct methods.
- **Conjugate Gradients (CG)**:
  - Guaranteed to converge in at most n steps.
  - Each step costs $O(n^2)$ for dense A.
  - Often converges much faster in practice, especially for well-conditioned A.

Conjugate Gradients

Energy Interpretation

- Recall the quadratic energy function:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{b}^\top \mathbf{x} + c.$$

- Using the true solution $A\mathbf{x}^* = \mathbf{b}$:

$$f(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x}^*)^\top A(\mathbf{x} - \mathbf{x}^*) + \text{const.}$$

- Hence $f(\mathbf{x})$ measures the distance between $\mathbf{x}$ and $\mathbf{x}^*$ in the **A-norm**:

$$\|\mathbf{v}\|_A^2 := \mathbf{v}^\top A\mathbf{v}.$$

Conjugate Gradients

A-Norm

- For symmetric positive definite A, Cholesky factorization gives $A = LL^\top$.
- Then:

$$f(\mathbf{x}) = \frac{1}{2}\|L^\top(\mathbf{x} - \mathbf{x}^*)\|_2^2 + \text{const.}$$

- This confirms that f measures a "distance" in A-norm between $\mathbf{x}$ and $\mathbf{x}^*$.
- Define change of variables:

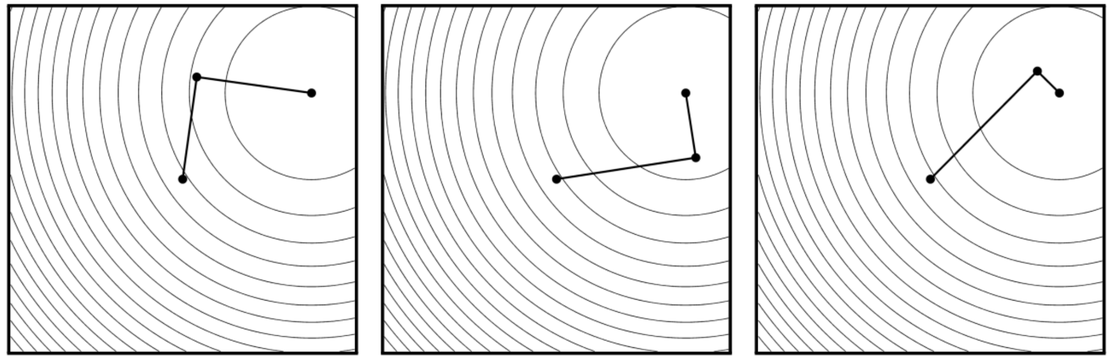
$$\mathbf{y} = L^\top \mathbf{x}, \quad \mathbf{y}^* = L^\top \mathbf{x}^*.$$

  Then, minimizing $f(\mathbf{x})$ is equivalent to minimizing

$$\bar{f}(\mathbf{y}) = \frac{1}{2}\|\mathbf{y} - \mathbf{y}^*\|_2^2.$$

Conjugate Gradients

Orthogonality and Conjugacy



- **Proposition 11.1:** If $\{\mathbf{w}_1, \ldots, \mathbf{w}_n\}$ are orthogonal in $\mathbb{R}^n$, then $\bar{f}$ is minimized in at most n steps by optimizing sequentially along each $\mathbf{w}_i$.

- In original coordinates, orthogonality becomes: $0 = \mathbf{w}_i \cdot \mathbf{w}_j = \mathbf{v}_i^\top A \mathbf{v}_j \Rightarrow \mathbf{v}_i, \mathbf{v}_j$ are **A-conjugate**.

- **Proposition 11.2:** If $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ are A-conjugate, then

  $$f \text{ is minimized in at most } n \text{ steps by line search along } \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n.$$

  Idea: Conjugate gradient constructs a sequence of A-conjugate directions that guarantees convergence in $\leq n$ steps.

Conjugate Gradients

Orthogonality and Conjugacy (cont.)

- If we express the solution $\mathbf{x}^*$ using the A-conjugate vectors: $\mathbf{x}^* = \sum_i \alpha_i \mathbf{v}_i$, then an algorithm computing an α_j in each iteration j can solve the problem exactly in n iterations.
- For any j, left-multiply $\mathbf{v}_j$ to $A\mathbf{x}^* = \mathbf{b}$, we see $\mathbf{v}_j^T A \sum_i \alpha_i \mathbf{v}_i = \mathbf{v}_j^T b \;\Rightarrow\; \mathbf{v}_j^T A \alpha_j \mathbf{v}_j = \mathbf{v}_j^T b$, which gives $\alpha_j = \mathbf{v}_j^T b / (\mathbf{v}_j^T A \mathbf{v}_j)$.
- Thus, given the set of A-conjugate vectors $\mathbf{v}_j$, such solver is straightforward to build.

Conjugate Gradients

Gradient Descent vs Conjugate Gradients

- At iteration k, define residual $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k = -\nabla f(\mathbf{x}_k)$. In gradient descent: $\mathbf{v}_{k+1} = \mathbf{r}_k$ (steepest descent direction); In CG, $\mathbf{v}_{k+1}$ is chosen such that $\mathbf{v}_{k+1}^T A\mathbf{v}_i = 0, \forall i \leq k$.

- Update rule: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_{k+1}\mathbf{v}_{k+1}, \quad \alpha_{k+1} = \frac{\mathbf{v}_{k+1}^\top \mathbf{r}_k}{\mathbf{v}_{k+1}^\top A\mathbf{v}_{k+1}}. \quad \mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_{k+1}A\mathbf{v}_{k+1}.$

- **Proposition 11.3**: In gradient descent: $\text{span}\{\mathbf{r}_0, \ldots, \mathbf{r}_k\} = \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^k\mathbf{r}_0\}$.

- Gradient descent explores a Krylov subspace generated by A and $\mathbf{r}_0$. However, its iterates $\mathbf{x}_k$ may not minimize f over that subspace.

- CG ensures: $\mathbf{x}_k = \arg\min_{\mathbf{v} \in \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}} f(\mathbf{x}_0 + \mathbf{v})$, yielding an optimal approximation in each subspace expansion by avoiding undoing progress from previous iterations.

Conjugate Gradients

Generating A-Conjugate Directions

- Idea: iterative construction that maintains A-conjugacy:

$$\mathbf{v}_k \leftarrow A^{k-1}\mathbf{r}_0 - \sum_{i<k} \frac{\langle A^{k-1}\mathbf{r}_0, \mathbf{v}_i \rangle_A}{\langle \mathbf{v}_i, \mathbf{v}_i \rangle_A} \mathbf{v}_i,$$

$$\alpha_k \leftarrow \frac{\mathbf{v}_k^\top \mathbf{r}_{k-1}}{\mathbf{v}_k^\top A \mathbf{v}_k},$$

$$\mathbf{x}_k \leftarrow \mathbf{x}_{k-1} + \alpha_k \mathbf{v}_k,$$

$$\mathbf{r}_k \leftarrow \mathbf{r}_{k-1} - \alpha_k A \mathbf{v}_k.$$

- Equivalent to explicit Gram–Schmidt, costly and memory-heavy:
  - $A^{k-1}\mathbf{r}_0$ tends to align with the dominant eigenvector of A, making projection worse conditioned.
  - Must store all $\mathbf{v}_1, \ldots, \mathbf{v}_{k-1}$ for projection.

Conjugate Gradients

Improved Construction via Residuals

- To improve conditioning, replace $A^{k-1}\mathbf{r}_0$ by $\mathbf{r}_{k-1}$, which already encodes higher powers of A:

$$\boxed{\begin{aligned}
\mathbf{v}_k &\leftarrow \mathbf{r}_{k-1} - \sum_{i<k} \frac{\langle \mathbf{r}_{k-1}, \mathbf{v}_i \rangle_A}{\langle \mathbf{v}_i, \mathbf{v}_i \rangle_A} \mathbf{v}_i, \\
\alpha_k &\leftarrow \frac{\mathbf{v}_k^\top \mathbf{r}_{k-1}}{\mathbf{v}_k^\top A \mathbf{v}_k}, \\
\mathbf{x}_k &\leftarrow \mathbf{x}_{k-1} + \alpha_k \mathbf{v}_k, \\
\mathbf{r}_k &\leftarrow \mathbf{r}_{k-1} - \alpha_k A \mathbf{v}_k.
\end{aligned}}$$

- **Key Observation (Proposition 11.5)**: Most projection terms vanish:

$$\langle \mathbf{r}_k, \mathbf{v}_\ell \rangle_A = 0 \quad \forall\, \ell < k.$$

Thus, the projection in iteration k depends only on $\mathbf{v}_{k-1}$.

Conjugate Gradients

The Algorithm

- Simplified residual-based iteration:

$$
\begin{aligned}
\mathbf{v}_k &\leftarrow \mathbf{r}_{k-1} - \frac{\langle \mathbf{r}_{k-1}, \mathbf{v}_{k-1} \rangle_A}{\langle \mathbf{v}_{k-1}, \mathbf{v}_{k-1} \rangle_A} \mathbf{v}_{k-1}, \\
\alpha_k &\leftarrow \frac{\mathbf{v}_k^\top \mathbf{r}_{k-1}}{\mathbf{v}_k^\top A \mathbf{v}_k}, \\
\mathbf{x}_k &\leftarrow \mathbf{x}_{k-1} + \alpha_k \mathbf{v}_k, \\
\mathbf{r}_k &\leftarrow \mathbf{r}_{k-1} - \alpha_k A \mathbf{v}_k.
\end{aligned}
$$

- Simplified Coefficients & Numerical Form:

$$
\begin{aligned}
x_{k+1} &= x_k + \alpha_k v_k \qquad (\alpha_k = \frac{r_k^\top r_k}{v_k^\top A v_k}) \\
r_{k+1} &= r_k - \alpha_k A v_k \\
v_{k+1} &= r_{k+1} + \beta_k v_k \qquad (\beta_k = \frac{r_{k+1}^\top r_{k+1}}{r_k^\top r_k})
\end{aligned}
$$

Numerical Advantages

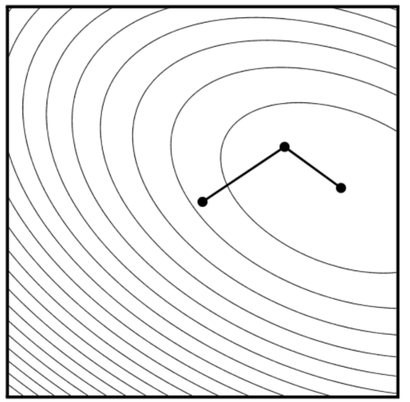
- Requires only one matrix-vector product per iteration.
- No need to store all previous directions.
- $\alpha_k, \beta_k \geq 0$ ensure stability under finite precision.

Conjugate Gradients

Convergence and Stopping Conditions

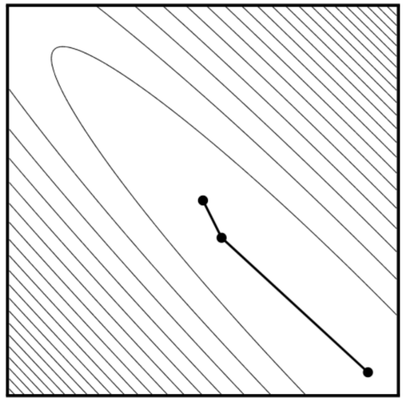
- CG converges as fast as gradient descent on $f(\mathbf{x})$, often much faster.
- Bound on convergence rate:

$$\frac{f(\mathbf{x}_k) - f(\mathbf{x}^*)}{f(\mathbf{x}_0) - f(\mathbf{x}^*)} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k, \quad \kappa = \text{cond}(A).$$



Well conditioned A

- Number of iterations $\propto \sqrt{\kappa}$ (vs. κ for gradient descent).
- Converges exactly in n steps if A has n unique eigenvalues.
- **Practical Stopping Criterion**: Stop when $\frac{\|\mathbf{r}_k\|}{\|\mathbf{r}_0\|} < \epsilon$.



Poorly conditioned A

Table of Content

- Gradient Descent
- Conjugate Gradients
- Preconditioning
- Other Iterative Algorithms

Preconditioning

Motivation

- Gradient Descent and Conjugate Gradient both converge for SPD matrix A.
- The convergence rate depends on $\text{cond}(A)$: smaller $\text{cond}(A) \Rightarrow$ faster convergence.
- To improve conditioning, we introduce a change of variables using an invertible matrix P:

$$A\mathbf{x} = \mathbf{b} \quad \Rightarrow \quad PA\mathbf{x} = P\mathbf{b}.$$

- If $P \approx A^{-1}$, then $\text{cond}(PA) \ll \text{cond}(A)$. P is called a **preconditioner**.
- **Challenges**
  - A may be SPD, but PA may not be symmetric or positive definite.
  - P must be easy to apply (approximate A^{-1} cheaply).

Preconditioning

Conjugate Gradients with Preconditioning

- CG requires symmetry and positive definiteness. Using PA directly may violate these.
- Instead, assume P is SPD, then we can write $P^{-1} = EE^\top$ (Cholesky factorization).

$$E^{-1}AE^{-\top} \approx I \quad \Rightarrow \quad \text{CG on } E^{-1}AE^{-\top} \text{ is well-conditioned.}$$

- **Proposition 11.6:** PA and $E^{-1}AE^{-\top}$ have the same eigenvalues.
- **Proposition 11.7:** $\text{cond}(PA) \geq \text{cond}(E^{-1}AE^{-\top})$.
- Hence $E^{-1}AE^{-\top}$ provides a symmetric, better-conditioned system.
- Solve $E^{-1}AE^{-\top}\mathbf{y} = E^{-1}\mathbf{b}, \quad \mathbf{x} = E^{-\top}\mathbf{y}$.

Preconditioning

Preconditioned CG Iteration

- Equivalent form:

$$
\begin{aligned}
\beta_k &\leftarrow \frac{\mathbf{r}_{k-1}^\top \mathbf{r}_{k-1}}{\mathbf{r}_{k-2}^\top \mathbf{r}_{k-2}}, \\
\mathbf{v}_k &\leftarrow \mathbf{r}_{k-1} + \beta_k \mathbf{v}_{k-1}, \\
\alpha_k &\leftarrow \frac{\mathbf{r}_{k-1}^\top \mathbf{r}_{k-1}}{\mathbf{v}_k^\top E^{-1} A E^{-\top} \mathbf{v}_k}, \\
\mathbf{y}_k &\leftarrow \mathbf{y}_{k-1} + \alpha_k \mathbf{v}_k, \\
\mathbf{r}_k &\leftarrow \mathbf{r}_{k-1} - \alpha_k E^{-1} A E^{-\top} \mathbf{v}_k.
\end{aligned}
$$

- Using substitutions:
  $\tilde{\mathbf{r}}_k = E\mathbf{r}_k, \quad \tilde{\mathbf{v}}_k = E^{-\top}\mathbf{v}_k, \quad \mathbf{x}_k = E^{-\top}\mathbf{y}_k$, the iteration can be rewritten purely in terms of A and $P = E^{-\top}E^{-1}$:

$$
\begin{aligned}
\beta_k &\leftarrow \frac{\tilde{\mathbf{r}}_{k-1}^\top P \tilde{\mathbf{r}}_{k-1}}{\tilde{\mathbf{r}}_{k-2}^\top P \tilde{\mathbf{r}}_{k-2}}, \\
\tilde{\mathbf{v}}_k &\leftarrow P\tilde{\mathbf{r}}_{k-1} + \beta_k \tilde{\mathbf{v}}_{k-1}, \\
\alpha_k &\leftarrow \frac{\tilde{\mathbf{r}}_{k-1}^\top P \tilde{\mathbf{r}}_{k-1}}{\tilde{\mathbf{v}}_k^\top A \tilde{\mathbf{v}}_k}, \\
\mathbf{x}_k &\leftarrow \mathbf{x}_{k-1} + \alpha_k \tilde{\mathbf{v}}_k, \\
\tilde{\mathbf{r}}_k &\leftarrow \tilde{\mathbf{r}}_{k-1} - \alpha_k A \tilde{\mathbf{v}}_k.
\end{aligned}
$$

Preconditioning

Convergence and Interpretation

- Preconditioned CG converges according to $\text{cond}(E^{-1}AE^{-\top})$.
- The algorithm uses only matrix-vector products with A and P.
- **Goal:** find P such that $P \approx A^{-1}$ and P is cheap to apply.
- **Trade-off:** If applying P is too costly, preconditioning may not help overall.
- **Key Insight**: Preconditioning accelerates convergence by improving spectral clustering of A's eigenvalues.

Preconditioning

Common Preconditioners

- The choice of P is problem-specific. Even rough approximations often improve convergence.
- **Examples:**
  - **Diagonal (Jacobi):** $P = \text{diag}(A)^{-1}$. Mitigates scaling differences between rows.
  - **Sparse Approximate Inverse:** $\min_{P \in S} \|AP - I\|_F, \quad$ with P sparse.
  - **Incomplete Cholesky:** approximate A^{-1} via partial factorization $A \approx L_* L_*^\top$ with limited fill-in.
  - **Domain Decomposition:** block-diagonalize A by graph partitioning, solving sub-blocks independently.
- Some methods provide theoretical bounds on $\text{cond}(PA)$, but most are heuristic – performance must be verified experimentally.

Table of Content

- Gradient Descent
- Conjugate Gradients
- Preconditioning
- Other Iterative Algorithms

Other Iterative Algorithms

Overview

- So far, we solved $A\mathbf{x} = \mathbf{b}$ assuming A is **square, symmetric, positive definite (SPD)**.
- Many practical systems are **asymmetric**, **indefinite**, and/or **rectangular**.
- In these cases, convergence theory is more involved. Here we summarize several important extensions and alternatives.

Other Iterative Algorithms

Splitting Methods

- Decompose
$$A = M - N \quad \Rightarrow \quad A\mathbf{x} = \mathbf{b} \iff M\mathbf{x} = N\mathbf{x} + \mathbf{b}.$$
- If M is easy to invert:
$$M\mathbf{x}_k = N\mathbf{x}_{k-1} + \mathbf{b}.$$
- Define iteration matrix $G = M^{-1}N$. Convergence requires $\rho(G) < 1$ (spectral radius).
- Popular choice: $M = \text{diag}(A)$.
- **Successive over-relaxation (SOR)** introduces a relaxation parameter to improve convergence.

Other Iterative Algorithms

CG on Normal Equations

- When A is not SPD, we can apply CG to the **normal equations**:

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

- **CGNR (Conjugate Gradient Normal Residual):** applies CG to $A^\top A\mathbf{x} = A^\top \mathbf{b}$.
- **CGNE (Conjugate Gradient Normal Error):** applies CG to $AA^\top \mathbf{y} = \mathbf{b}$, then $\mathbf{x} = A^\top \mathbf{y}$.
- Always convergent if A is full rank, but can be slow due to poor conditioning of $A^\top A$.

Other Iterative Algorithms

MINRES and LSQR/LSMR

- **MINRES**:
  - Apply to all symmetric A (not necessarily SPD).
  - Minimize
$$f(\mathbf{x}) = \tfrac{1}{2}\|\mathbf{b} - A\mathbf{x}\|_2^2.$$
- **LSQR / LSMR:**
  - Designed for least-squares systems $\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2$.
  - Better conditioning and stability than CGNR/CGNE.
  - Effective even when A is rectangular.

Other Iterative Algorithms

Generalized Krylov Subspace Methods

- Broader family for nonsymmetric or indefinite A:

  GMRES, QMR, BiCG, CGS, BiCGStab.

- Assume only that A is square and invertible.

- They minimize quadratic-like energies but require:
  - Storing basis vectors or search directions.
  - Possibly factoring intermediate matrices for stability.

- Trade-off: more generality $\Rightarrow$ more computation per iteration.

Other Iterative Algorithms

Non-Quadratic Optimization Extensions

- For minimizing nonlinear $f(\mathbf{x})$, CG can be generalized to nonlinear or quasi-Newton forms:

  Fletcher–Reeves, Hestenes–Stiefel, Polak–Ribière, Dai–Yuan.

- Each iteration updates the search direction via a formula using ∇f instead of residuals:

$$\mathbf{p}_k = -\nabla f_k + \beta_k \mathbf{p}_{k-1}.$$

- Effective when f is locally well approximated by a quadratic.
- Used widely in nonlinear optimization, machine learning, and graphics.

Other Iterative Algorithms

Practical Remarks and Rule of Thumb

- Most algorithms are nearly as easy to implement as CG or GD.
- Ready-made solvers (e.g. MATLAB, SciPy, PETSc) only need A and $\mathbf{b}$.
- Users must provide efficient routines for $A\mathbf{x}$ and $A^\top \mathbf{x}$.
- **Rule of Thumb**: The **fewer assumptions** a method makes about A, the **more iterations** it generally needs to converge.
- There are no universal rules – method selection depends on the structure of A.

Table of Content

- Gradient Descent
- Conjugate Gradients
- Preconditioning
- Other Iterative Algorithms