An Introduction to Sparse Linear Algebra

Storage, Solvers, and Open Problems

Harnoor Sethi

July 7, 2026

The problem: solving $\mathbf{Ax} = \mathbf{b}$

Linear systems

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

are ubiquitous across science and engineering.

- Small n: classical **dense** methods (Gaussian elimination, QR) work fine.
  - $O(n^3)$ arithmetic, $O(n^2)$ storage.
- But many real problems have $n = 10^5$ to 10^9 or more.
- At that scale, dense methods are **infeasible** in both time and memory.

Locality gives sparsity

Large systems usually come from *local* interactions: each unknown depends on only a bounded number of neighbors.

Model example: discretizing $-\Delta$ on an $N \times N$ grid

- Matrix dimension $n = N^2$.
- Only $O(n)$ non-zeros out of $O(n^2)$ possible entries $\Rightarrow$ **sparse**.

The goal

Exploit the zeros to cut

$$\text{storage } O(n^2) \to O(\mathrm{nnz}(\mathbf{A})), \qquad \text{solve time } O(n^3) \to \text{much less.}$$

Defining sparsity

Definition

Let $\mathbf{A} \in \mathbb{R}^{m \times n}$. The *number of non-zeros* is

$$\operatorname{nnz}(\mathbf{A}) = \big|\{(i,j) : A_{ij} \neq 0\}\big|.$$

The *sparsity* is the fraction of zero entries,

$$\operatorname{sp}(\mathbf{A}) = 1 - \frac{\operatorname{nnz}(\mathbf{A})}{mn}.$$

$\mathbf{A}$ is *sparse* when $\operatorname{sp}(\mathbf{A})$ is close enough to 1 that exploiting the zeros gives a real computational benefit.

In practice "sparse" means $\operatorname{nnz}(\mathbf{A}) = O(n)$ or $O(n \log n)$ as $n \to \infty$, versus a *dense* matrix with $\operatorname{nnz}(\mathbf{A}) = \Theta(n^2)$.

Where sparse matrices arise

- **Finite element discretizations.** Each mesh element interacts only with adjacent elements; sparsity follows mesh connectivity.
- **Network / graph problems.** Adjacency and Laplacian matrices of social graphs, power grids, citation networks — bounded node degree.
- **Circuit simulation.** Modified nodal analysis mirrors circuit topology.
- **Operations research.** Standard-form LP constraint matrices are typically highly sparse.
- **Machine learning.** Recommender systems, bag-of-words NLP, and kernel methods produce large sparse matrices.

The storage challenge

A dense $n \times n$ array stores n^2 entries — wasteful when most are zero.

Idea

Store only the non-zero entries (plus enough bookkeeping to know where they go).

Several formats have emerged, each trading off construction cost, memory, and arithmetic efficiency:

$$\text{COO} \quad \longrightarrow \quad \text{CSR/CSC} \quad \longrightarrow \quad \text{BCSR}, \text{DIA}, \text{ELL}.$$

Coordinate format (COO)

Store three arrays of length nnz, with $A_{\mathtt{row}[k],\,\mathtt{col}[k]} = \mathtt{val}[k]$:

$$\mathtt{row}[k], \quad \mathtt{col}[k], \quad \mathtt{val}[k], \qquad k = 0, \ldots, \mathrm{nnz} - 1.$$

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 3 & 0 & 4 \end{bmatrix}, \quad \mathrm{nnz} = 4.$$

$$\begin{aligned} \mathtt{row} &= [\,0,\ 1,\ 2,\ 2\,] \\ \mathtt{col} &= [\,0,\ 1,\ 0,\ 2\,] \\ \mathtt{val} &= [\,1,\ 2,\ 3,\ 4\,] \end{aligned}$$

+ Easy to build incrementally; natural output of assembly.
− Irregular access; poor for matrix–vector products without conversion.

Compressed Sparse Row (CSR)

The most widely used format (a.k.a. CRS / Yale). Sort COO row-by-row and *compress* the row index into a pointer array:

- `val`: non-zero values in row-major order;
- `col_ind`: column index of each stored value;
- `row_ptr`[i]: index in `val` of the first non-zero in row i.

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 3 & 0 & 4 \end{bmatrix}$$

$$\texttt{val} = [1,\ 2,\ 3,\ 4]$$
$$\texttt{col\_ind} = [0,\ 1,\ 0,\ 2]$$
$$\texttt{row\_ptr} = [0,\ 1,\ 2,\ 4]$$

Row 2 occupies positions 2–3; the final $\texttt{row\_ptr}[3] = 4 = \mathrm{nnz}$ marks the end of the data.

CSC and specialized formats

Compressed Sparse Column (CSC)

Transpose analogue of CSR — compress by *columns*: `val`, `row_ind`, `col_ptr`. Natural for $\mathbf{y} \leftarrow \mathbf{A}^\top \mathbf{x}$ and column-oriented factorizations.

Block & diagonal formats

- **BCSR**: small dense blocks (multiple DOFs per node in FEM) — cuts index overhead, improves cache use.
- **DIA**: stores each occupied diagonal contiguously — great for regular stencils.
- **ELL**: pads each row to equal length — SIMD/GPU friendly.

Takeaway

Format choice is problem- and hardware-specific — *automatic format selection* is itself a research topic.

Direct solvers and the fill-in problem

A *direct solver* factors $\mathbf{A}$ explicitly (LU, Cholesky if SPD, or QR) and then solves two triangular systems.

The central difficulty: fill-in

The factors L and U can have **many more** non-zeros than $\mathbf{A}$, destroying sparsity and inflating storage and work.

- Even a matrix with $O(n)$ non-zeros can produce factors with $O(n^{3/2})$ or $O(n^2)$ non-zeros.
- For the $N \times N$ grid ($n = N^2$), naive Gaussian elimination gives $O(n^{3/2})$ fill-in and $O(n^2)$ operations.

Worked example: solving via QR

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \qquad b = \begin{bmatrix} 2 \\ 3 \\ 3 \end{bmatrix}, \qquad \text{solve } Ax = b.$$

Factor $A = QR$ (columns of Q come from Gram–Schmidt — next slide):

$$Q = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{3}} \\ 0 & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{3}} \end{bmatrix}, \qquad R = \begin{bmatrix} \sqrt{2} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \sqrt{\frac{3}{2}} & \frac{1}{\sqrt{6}} \\ 0 & 0 & \frac{2}{\sqrt{3}} \end{bmatrix}.$$

Since $Q^\top Q = I$, the system $QRx = b$ becomes $Rx = Q^\top b$:

$$Rx = Q^\top b = \left[\tfrac{5}{\sqrt{2}},\ \tfrac{5}{\sqrt{6}},\ \tfrac{2}{\sqrt{3}}\right]^\top, \qquad \text{back-substitute} \Rightarrow x = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}.$$

Where Q and R come from: Gram–Schmidt

Take the columns of A and orthonormalize them left to right. Let $a_1 = (1,1,0)$, $a_2 = (1,0,1)$, $a_3 = (0,1,1)$.

$$q_1 = \frac{a_1}{\|a_1\|} = \tfrac{1}{\sqrt{2}}(1,1,0),$$

$$\tilde{q}_2 = a_2 - (q_1^\top a_2)\, q_1 = (1,0,1) - \tfrac{1}{2}(1,1,0) = (\tfrac{1}{2}, -\tfrac{1}{2}, 1), \quad q_2 = \tfrac{1}{\sqrt{6}}(1,-1,2),$$

$$\tilde{q}_3 = a_3 - (q_1^\top a_3)\, q_1 - (q_2^\top a_3)\, q_2 = (-\tfrac{2}{3}, \tfrac{2}{3}, \tfrac{2}{3}), \qquad q_3 = \tfrac{1}{\sqrt{3}}(-1,1,1).$$

The projection coefficients $r_{ij} = q_i^\top a_j$ (with $r_{ii} = \|\tilde{q}_i\|$) are exactly the entries of R:

$$r_{11} = \sqrt{2}, \quad r_{12} = r_{13} = \tfrac{1}{\sqrt{2}}, \quad r_{22} = \sqrt{\tfrac{3}{2}}, \quad r_{23} = \tfrac{1}{\sqrt{6}}, \quad r_{33} = \tfrac{2}{\sqrt{3}},$$

so $A = QR$ with the Q, R from the previous slide.

Worked example: computing the LU factorization

Factor $A = LU$ with L unit lower-triangular. Eliminate below the diagonal one column at a time, recording each multiplier.

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

Step 1 — column 1. Pivot $A_{11} = 1$, multiplier $\ell_{21} = A_{21}/A_{11} = 1$:

$$\text{row } 2 \leftarrow [1,0,1] - 1 \cdot [1,1,0] = [0,-1,1] \;\Rightarrow\; \begin{bmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

Step 2 — column 2. Pivot -1, multiplier $\ell_{32} = 1/(-1) = -1$:

$$\text{row } 3 \leftarrow [0,1,1] - (-1) \cdot [0,-1,1] = [0,0,2] \;\Rightarrow\; U = \begin{bmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 2 \end{bmatrix}.$$

Placing the multipliers below the diagonal gives $L = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix}$, and one checks $LU = A$.

Worked example: solving with L and U

With $A = LU$, solve $Ly = b$ then $Ux = y$ (two triangular solves), $b = [2, 3, 3]^\top$.

Forward: $Ly = b$

$$
\begin{aligned}
y_1 &= 2, \\
y_2 &= 3 - y_1 = 1, \\
y_3 &= 3 + y_2 = 4.
\end{aligned}
$$

Back: $Ux = y$

$$
\begin{aligned}
x_3 &= \tfrac{4}{2} = 2, \\
x_2 &= \tfrac{1 - x_3}{-1} = 1, \\
x_1 &= 2 - x_2 = 1.
\end{aligned}
$$

Same answer $x = [1, 1, 2]^\top$ as QR. LU is preferred for sparse systems: L preserves A's sparsity (modulo fill-in), whereas Q is a *dense* orthogonal matrix.

Direct vs. iterative — and why ordering matters

Both LU and QR reach the answer in a **finite** sequence of algebraic steps: they are *direct* methods.

- For small matrices this is entirely fine.
- For large sparse matrices, factoring *without* a smart ordering is inefficient — fill-in explodes.

Key idea

Permuting rows/columns *before* factorization can drastically reduce fill-in — and the effect is best understood through graphs.

Graphs can represent matrices

For a symmetric $\mathbf{A}$, build an undirected graph G:

- one **vertex** per row/column index i;
- an **edge** (i, j) exactly when $A_{ij} \neq 0$ $(i \neq j)$.

The sparsity pattern of $\mathbf{A}$ *is* the adjacency structure of G.

$$\begin{bmatrix} * & * & 0 \\ * & * & * \\ 0 & * & * \end{bmatrix} \longleftrightarrow 1 - 2 - 3$$

a path (tridiagonal)

$$\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \longleftrightarrow \begin{array}{c} 1 \\ / \ \backslash \\ 2 - 3 \end{array}$$

a triangle (dense 3×3)

More non-zeros $\Leftrightarrow$ more edges. Fill-in during elimination will show up as *new* edges.

Removing a vertex of a graph

Definition

A position (i, j) with $A_{ij} = 0$ but $L_{ij} \neq 0$ or $U_{ij} \neq 0$ is a *fill-in entry*.

Eliminating vertex k (one step of Gaussian elimination) means:

1. remove k and all edges touching it;
2. connect *every pair* of k's former neighbors that were not already joined.

Each edge added in step 2 is a **fill-in entry**.

Example. On the path $1 - 2 - 3 - 4$ (tridiagonal A), eliminate vertex 2. Its neighbors $\{1, 3\}$ get a new edge $(1, 3)$:

$$\begin{bmatrix} * & * & 0 & 0 \\ * & * & * & 0 \\ 0 & * & * & * \\ 0 & 0 & * & * \end{bmatrix} \xrightarrow{\text{eliminate } 2} 1 - 3 - 4 \quad \Rightarrow \quad \text{fill at } (1,3), (3,1).$$

The Schur complement: removing an index cleanly

Eliminating pivot k updates every remaining entry by

$$A_{ij} \leftarrow A_{ij} - \frac{A_{ik}\, A_{kj}}{A_{kk}}.$$

In block form, split off the pivot block A_{11}:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \implies \text{Schur complement } \; S = A_{22} - A_{21} A_{11}^{-1} A_{12}.$$

What S is

S is the system you are left with after *exactly* eliminating the unknowns in block 1 — it "simulates removing those indices" while keeping the solution on the rest unchanged.

New non-zeros in S (where A_{22} had zeros) are precisely the **fill** — the new edges of graph elimination.

Schur complement: a worked example

$$A = \begin{bmatrix} 4 & 2 & 1 \\ 2 & 3 & 0 \\ 1 & 0 & 2 \end{bmatrix} \qquad \longleftrightarrow \qquad 2 - 1 - 3$$

(vertex 1 joins 2 and 3; entry $A_{23} = 0$, so 2–3 has no edge). Eliminate index 1: pivot $A_{11} = 4$, $A_{12} = [2,\ 1]$, $A_{22} = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$.

$$S = A_{22} - A_{21}A_{11}^{-1}A_{12} = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} - \tfrac{1}{4}\begin{bmatrix} 2 \\ 1 \end{bmatrix}\begin{bmatrix} 2 & 1 \end{bmatrix} = \begin{bmatrix} 2 & -\frac{1}{2} \\ -\frac{1}{2} & \frac{7}{4} \end{bmatrix}.$$

The off-diagonal $S_{23} = -\frac{1}{2} \neq 0$ is a **fill-in entry** — exactly the new edge $(2, 3)$ created when vertex 1 was eliminated.

Running example: a tridiagonal / path graph

$$A = \begin{bmatrix} * & * & 0 & 0 & 0 & 0 \\ * & * & * & 0 & 0 & 0 \\ 0 & * & * & * & 0 & 0 \\ 0 & 0 & * & * & * & 0 \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \end{bmatrix} \qquad \longleftrightarrow \qquad 1 - 2 - 3 - 4 - 5 - 6$$

Degree sequence:

$$d(1) = d(6) = 1, \qquad d(2) = d(3) = d(4) = d(5) = 2.$$

The two endpoints have a single neighbor, so eliminating either one introduces **no fill**. We now trace two ordering strategies on this graph.

Theorem: graph fill = matrix fill

Theorem

Let $\mathbf{A}$ *be symmetric with graph* $G = (V, E)$, $(i, j) \in E \iff A_{ij} \neq 0$ *for* $i \neq j$. *Let* $\sigma = (\sigma_1, \ldots, \sigma_n)$ *be an elimination ordering and* $G^{(k)}$ *the elimination graph after removing* $\sigma_1, \ldots, \sigma_{k-1}$. *Then* (σ_i, σ_j) $(i < j)$ *is a fill-in entry of the Cholesky factor* L *if and only if it is a fill edge produced during graph elimination.*

Consequence

Graph elimination produces *no* fill edges $\iff$ the factor L has *no* fill-in entries. Minimizing fill is a purely combinatorial problem.

Proof sketch (induction on elimination step k)

Let $A^{(k)}$ be the Schur complement after $k-1$ steps. The update

$$A_{ij}^{(k+1)} = A_{ij}^{(k)} - \frac{A_{i\sigma_k}^{(k)} A_{\sigma_k j}^{(k)}}{A_{\sigma_k \sigma_k}^{(k)}}$$

drives everything.

- **Base** ($k = 1$): $A^{(1)} = A$, $G^{(1)} = G$ — edges are exactly the non-zeros by construction.
- **Step:** assume edges of $G^{(k)}$ = non-zeros of $A^{(k)}$. A new non-zero appears in $A_{ij}^{(k+1)}$ exactly when $A_{ij}^{(k)} = 0$ but both $A_{i\sigma_k}^{(k)} \neq 0$ and $A_{\sigma_k j}^{(k)} \neq 0$ — i.e. i, j are both neighbors of σ_k.

That is precisely when graph elimination adds the fill edge (i, j). So the new non-zeros and the new fill edges coincide, completing the induction. Accumulated non-zeros = fill-in of L. □

Minimum Degree (MD) ordering

Heuristic: repeatedly eliminate the vertex of smallest current degree — it connects the fewest neighbors, so it creates the fewest fill edges. (Modern codes use Approximate Minimum Degree, AMD.)

On the path 1–2–3–4–5–6**:**

1. degrees $1, 6 : 1$; pick 1, neighbor $\{2\}$ — **no fill**.
2. on 2–3–4–5–6: pick 6, neighbor $\{5\}$ — **no fill**.
3. on 2–3–4–5: pick 2, neighbor $\{3\}$ — **no fill**.
4. on 3–4–5: pick 5, neighbor $\{4\}$ — **no fill**.
5. on 3–4: eliminate 3 then 4 — **no fill**.

Ordering $(1, 6, 2, 5, 3, 4)$. Every step removes a leaf $\Rightarrow$ zero fill: L needs no storage beyond $PAP^\top$.

Nested Dissection

Find a small *separator* whose removal splits the graph into balanced independent pieces; recurse on each piece; place separator vertices *last*. Postponing cross-piece interactions keeps fill from bridging the two halves.

On the example: separator $\{3\}$ splits into $S_L = \{1, 2\}$, $S_R = \{4, 5, 6\}$, giving the order $(1, 2, 6, 5, 4, 3)$.

Why it scales (George, 1973)

For $\sqrt{n} \times \sqrt{n}$ grid graphs, nested dissection achieves $O(n^{3/2})$ fill-in in 2D and $O(n^2)$ in 3D — asymptotically **optimal**. In practice AMD approaches this quality on many problems.

Iterative solvers

Direct solvers compute $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ exactly, but pay for fill-in. *Iterative* methods instead generate approximations $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \cdots \to \mathbf{x}^\star$ using $\mathbf{A}$ only through matrix–vector products.

- Each product costs $O(\mathrm{nnz}(\mathbf{A}))$ for a sparse $\mathbf{A}$.
- Storage: just $\mathbf{A}$ plus a handful of length-n vectors.
- **No factorization $\Rightarrow$ no fill-in, ever.**

The dominant family is built on *Krylov subspaces*.

Krylov subspaces

Definition

For $A \in \mathbb{R}^{n \times n}$ and nonzero $v \in \mathbb{R}^n$, the *kth Krylov subspace* is

$$\mathcal{K}_k(A, v) = \text{span}\{v, Av, A^2v, \ldots, A^{k-1}v\}.$$

For $\mathbf{Ax} = \mathbf{b}$ with initial guess $\mathbf{x}_0$, take $v = \mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$ (the initial residual) and work in $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$.

Every subspace is built purely from repeated multiplication by $\mathbf{A}$ — no inverse and no factorization are ever needed.

Why repeated multiplication by $\mathbf{A}$ suffices

Start with $\mathbf{r}_0$ and keep multiplying: $\mathbf{r}_0,\ \mathbf{A}\mathbf{r}_0,\ \mathbf{A}^2\mathbf{r}_0, \ldots$

- Generically each new vector $\mathbf{A}^k\mathbf{r}_0$ is **linearly independent** of the earlier ones, so $\dim \mathcal{K}_k = k$ grows by one each step — until the subspace becomes $\mathbf{A}$-invariant (after at most n steps).
- By Cayley–Hamilton, $\mathbf{A}^{-1}$ equals a polynomial in $\mathbf{A}$ of degree $\leq n-1$, so

$$\mathbf{x}^\star = \mathbf{x}_0 + \mathbf{A}^{-1}\mathbf{r}_0 = \mathbf{x}_0 + q(\mathbf{A})\,\mathbf{r}_0$$

  for some polynomial q — i.e. the solution is a **linear combination** of $\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \ldots, \mathbf{A}^{n-1}\mathbf{r}_0$.

Consequence

The exact solution lies in $\mathbf{x}_0 + \mathcal{K}_n(\mathbf{A}, \mathbf{r}_0)$: we can find it using matrix–vector products alone. Methods that search for $\mathbf{x}$ inside these subspaces are exactly the **Krylov subspace methods** (CG, GMRES, MINRES, BiCG, ...) — essentially every mainstream iterative solver.

The logic: turn solving into optimizing

When $\mathbf{A}$ is symmetric positive definite (SPD), define

$$\phi(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} - \mathbf{b}^\top \mathbf{x}, \qquad \nabla\phi(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}.$$

So $\nabla\phi(\mathbf{x}) = 0 \iff \mathbf{A}\mathbf{x} = \mathbf{b}$, and since $\mathbf{A}$ is SPD, ϕ is strictly convex with unique minimizer $\mathbf{x}^\star = \mathbf{A}^{-1}\mathbf{b}$.

Minimize over a growing nested chain of subspaces

$$\mathbf{x}_0 + \mathcal{K}_1 \subset \mathbf{x}_0 + \mathcal{K}_2 \subset \cdots \subset \mathbf{x}_0 + \mathcal{K}_n = \mathbb{R}^n.$$

Never lose earlier progress; reach $\mathbf{x}^\star$ within n steps. This is the **Conjugate Gradient (CG)** method.

CG: incremental minimization via conjugacy

Re-solving a growing subproblem each step would be expensive. CG does it incrementally:

- At step k, move from $\mathbf{x}_{k-1}$ along one new direction $\mathbf{p}_{k-1}$.
- Choose directions to be $\mathbf{A}$-*conjugate*: $\mathbf{p}_i^\top \mathbf{A}\mathbf{p}_j = 0$ for $i \neq j$.

Why conjugacy works

It makes a single one-dimensional line minimization coincide with the minimizer over the *entire* subspace so far — using only a short three-term recurrence and $O(n)$ storage.

The optimal step size

Theorem

Let $\mathbf{A}$ *be SPD,* $\mathbf{r}_{k-1} = \mathbf{b} - \mathbf{A}\mathbf{x}_{k-1}$*, and* $\mathbf{p}_{k-1} \neq 0$*. For* $\mathbf{x}(\alpha) = \mathbf{x}_{k-1} + \alpha\mathbf{p}_{k-1}$*, the functional* $\phi(\mathbf{x}(\alpha))$ *is minimized at*

$$\alpha_{k-1} = \frac{\mathbf{p}_{k-1}^\top \mathbf{r}_{k-1}}{\mathbf{p}_{k-1}^\top \mathbf{A}\mathbf{p}_{k-1}}.$$

Proof idea. $f(\alpha) = \phi(\mathbf{x}_{k-1} + \alpha\mathbf{p}_{k-1})$ is a quadratic in α with

$$f'(\alpha) = -\mathbf{p}_{k-1}^\top \mathbf{r}_{k-1} + \alpha\, \mathbf{p}_{k-1}^\top \mathbf{A}\mathbf{p}_{k-1}.$$

Since $\mathbf{A}$ SPD $\Rightarrow \mathbf{p}_{k-1}^\top \mathbf{A}\mathbf{p}_{k-1} > 0$, f is a strictly convex parabola; setting $f'(\alpha) = 0$ gives the unique minimizer. □

The full CG recurrence

Starting from $\mathbf{p}_0 = \mathbf{r}_0$, so $\alpha_0 = \dfrac{\mathbf{r}_0^\top \mathbf{r}_0}{\mathbf{p}_0^\top \mathbf{A}\mathbf{p}_0}$:

$$\alpha_{k-1} = \frac{\mathbf{p}_{k-1}^\top \mathbf{r}_{k-1}}{\mathbf{p}_{k-1}^\top \mathbf{A}\mathbf{p}_{k-1}}, \qquad \mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_{k-1}\mathbf{p}_{k-1},$$

$$\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_{k-1}\mathbf{A}\mathbf{p}_{k-1}, \quad \beta_{k-1} = \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{r}_{k-1}^\top \mathbf{r}_{k-1}},$$

$$\mathbf{p}_k = \mathbf{r}_k + \beta_{k-1}\mathbf{p}_{k-1}.$$

Cost per step: **one** matrix–vector product plus a few dot products, with **no growth** in storage — in sharp contrast to the fill-in of direct methods.

Worked example: a 3×3 SPD system

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \qquad \mathbf{x}^\star = \left[\tfrac{1}{2},\, 0,\, \tfrac{3}{2}\right]^\top.$$

($\mathbf{A}$ SPD: leading minors $2, 3, 4 > 0$.) Start from $\mathbf{x}_0 = \mathbf{0}$.

Step 1 ($\mathcal{K}_1$). $\mathbf{r}_0 = \mathbf{p}_0 = [1, 2, 3]^\top$, $\mathbf{A}\mathbf{p}_0 = [4, 8, 8]^\top$,

$$\alpha_0 = \tfrac{14}{44} = \tfrac{7}{22}, \qquad \mathbf{r}_1 = \left[-\tfrac{3}{11},\, -\tfrac{6}{11},\, \tfrac{5}{11}\right]^\top.$$

Step 2 ($\mathcal{K}_2$). $\beta_0 = \tfrac{5}{121}$, $\alpha_1 = \tfrac{55}{84}$,

$$\mathbf{x}_2 = \left[\tfrac{1}{6},\, \tfrac{1}{3},\, \tfrac{4}{3}\right]^\top, \qquad \mathbf{r}_2 = \left[\tfrac{1}{3},\, -\tfrac{1}{6},\, 0\right]^\top \neq \mathbf{0}.$$

Two dimensions are not enough for this generic right-hand side.

Worked example: convergence in exactly $n = 3$ steps

Step 3 ($\mathcal{K}_3 = \mathbb{R}^3$).

$$\beta_1 = \tfrac{121}{504}, \qquad \mathbf{p}_2 = \tfrac{1}{36}[10, -10, 5]^\top, \qquad \alpha_2 = \tfrac{6}{5}.$$

$$\mathbf{x}_3 = \mathbf{x}_2 + \alpha_2 \mathbf{p}_2 = \left[\tfrac{1}{6} + \tfrac{1}{3},\ \tfrac{1}{3} - \tfrac{1}{3},\ \tfrac{4}{3} + \tfrac{1}{6}\right]^\top = \left[\tfrac{1}{2},\ 0,\ \tfrac{3}{2}\right]^\top = \mathbf{x}^\star.$$

Result

CG converged in exactly $n = 3$ steps, using only three matrix–vector products $\mathbf{A}\mathbf{p}_0, \mathbf{A}\mathbf{p}_1, \mathbf{A}\mathbf{p}_2$ and **no factorization** — versus the $O(n^{3/2})$ fill even a well-ordered direct factorization can incur at scale.

Beyond SPD: GMRES

CG needs **A** SPD. For *general* nonsymmetric **A** there is no convex quadratic ϕ to minimize, so **GMRES** (Generalized Minimal RESidual, Saad & Schultz 1986) works directly with the residual instead.

- At step k it picks the $\mathbf{x}_k \in \mathbf{x}_0 + \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ that minimizes $\|\mathbf{b} - \mathbf{A}\mathbf{x}_k\|_2$.
- It builds an orthonormal basis of $\mathcal{K}_k$ via the *Arnoldi* process (modified Gram–Schmidt on $\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \dots$), reducing each step to a small $(k+1) \times k$ least-squares problem.
- Cost and storage *grow* with k (it keeps the whole basis) $\Rightarrow$ often *restarted* as GMRES(m).
- Convergence for nonnormal **A** is subtle — descriptive bounds remain an active question.

Robust convergence for both CG and GMRES hinges on **preconditioning**.

Open problems: iterative-solver theory

A skim of directions actively discussed (e.g. 2025 Simons Institute workshop):

- **GMRES for nonsymmetric / nonnormal A.** No clean convergence theory like CG's; behavior depends on eigenvector conditioning, and descriptive bounds capturing observed fast convergence are open.
- **Preconditioning $\neq$ shrinking $\kappa(\mathbf{A})$.** CG's actual speed depends on the *whole* spectrum, not just the condition number — characterizing this precisely is open.
- **Finite-precision Krylov methods.** In floating point the computed residuals drift from $\mathbf{b} - \mathbf{A}\mathbf{x}_k$ and exact termination is lost; a full account (after Paige, Greenbaum) is still missing.

Open problems: algorithms and scale

- **Randomized alternatives.** When do *sketch-and-project* (randomized row-action) methods beat CG? Even CG vs. randomized coordinate descent on polynomially-decaying spectra is unresolved.
- **Sparse matvec at scale.** These kernels are memory-bound with low arithmetic intensity; with unknowns reaching tens of billions, data layout, partitioning, and GPU/distributed execution are open engineering problems.
- **Forsythe conjecture (1968).** Restarted steepest-descent/CG residual directions appear to oscillate between two limits for fixed restart dimension $s \geq 2$ — still not fully understood.

The thread

Nearly every open direction relaxes one assumption of the CG example: SPD $\rightarrow$ nonsymmetric, exact $\rightarrow$ finite precision, unrestarted $\rightarrow$ restarted/randomized.

Summary

- **Sparsity** from locality is what makes large-scale linear algebra tractable.
- **Storage formats** (COO $\rightarrow$ CSR/CSC $\rightarrow$ BCSR/DIA/ELL) trade off construction, memory, and arithmetic.
- **Direct solvers** are exact but fight fill-in; graphs, the Schur complement, and *reordering* (MD/AMD, nested dissection) tame it.
- **Iterative Krylov methods** (CG, GMRES) avoid factorization entirely — ideal for the very largest systems.
- **Preconditioning, randomization, and scale** are where much of today's research lives.

Thank you — questions?