An Introduction to Sparse Linear Algebra: Storage, Solvers, and Open Problems

Harnoor Sethi

July 13, 2026

Abstract

Sparse linear algebra is the study of linear systems $\mathbf{Ax} = \mathbf{b}$ in which the matrix $\mathbf{A}$ contains overwhelmingly many zero entries. Such systems arise throughout applied mathematics and engineering—from the discretization of partial differential equations on large grids, to network analysis, circuit simulation, and structural mechanics—and their efficient solution is a cornerstone of scientific computing. In this survey we trace the development of sparse matrix methods from their mathematical foundations through to the current frontiers of research. We begin by defining sparsity and its practical consequences, discuss the principal storage formats (COO, CSR, CSC, and friends), then survey direct solvers based on matrix factorizations and the reordering heuristics that make them tractable. We then turn to iterative Krylov subspace methods, which remain a central tool for the largest systems. We conclude by sketching several open problems that are actively discussed today, including unresolved questions about Krylov convergence, finite-precision behavior, and sparse computation at scale.

1 Introduction

Linear systems of the form

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

are ubiquitous across science and engineering. When the dimension n is modest, classical dense methods based on Gaussian elimination or QR factorization—requiring $O(n^3)$ arithmetic operations and $O(n^2)$ storage—are fully adequate. However, many problems of genuine scientific interest produce systems in which n ranges from 10^5 to 10^9 or beyond. At those scales, dense methods are completely infeasible, both in time and in memory.

The saving grace is that such large systems typically arise from *local* physical interactions: each unknown depends on only a bounded number of its neighbors. Discretizing the Laplace operator $-\Delta$ on a uniform $N \times N$ grid, for instance, yields a matrix of dimension $n = N^2$. The resulting matrix has $O(n)$ non-zeros out of $O(n^2)$ possible entries; that is, it is *sparse*. Sparse linear algebra is the art of exploiting this structure to reduce storage from $O(n^2)$ to $O(\mathrm{nnz}(\mathbf{A}))$ and solve time from $O(n^3)$ to something far more tractable [1–3].

The goal of this paper is introductory rather than exhaustive. Topics such as GMRES, restarted Krylov methods, and modern solver theory are included to show where the basic ideas lead, but they are not treated as deep specialist subjects. Instead, the emphasis is on the main vocabulary, examples, and mathematical intuition needed to understand why sparse linear algebra differs from dense linear algebra.

2 Sparsity: Definition and Origins

Definition 2.1. Let $\mathbf{A} \in \mathbb{R}^{m\times n}$. The *number of non-zeros* of $\mathbf{A}$ is

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

The *sparsity* of $\mathbf{A}$ is the fraction of zero entries,

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

We call $\mathbf{A}$ *sparse* when $\mathrm{sp}(\mathbf{A})$ is sufficiently close to 1 such that exploiting the zeros produces a significant computational benefit.

In practice, the term "sparse" is used when $\mathrm{nnz}(\mathbf{A}) = O(n)$ or $O(n \log n)$ as $n \to \infty$, in contrast to a *dense* matrix with $\mathrm{nnz}(\mathbf{A}) = \Theta(n^2)$ [1].

2.1 Where Sparse Matrices Arise

Important sources of sparse matrices include:

- **Finite element discretizations.** In structural mechanics and fluid dynamics, each element of a mesh interacts only with adjacent elements, leading to sparsity determined by the mesh connectivity [3].
- **Network and graph problems.** Adjacency and Laplacian matrices of real-world networks (social graphs, power grids, citation networks) are sparse because each node has a bounded degree.
- **Circuit simulation.** Modified nodal analysis of electronic circuits yields sparse matrices whose structure reflects the circuit topology.
- **Operations research.** Linear programming in its standard form involves constraint matrices that are typically highly sparse.
- **Machine learning.** Recommender systems, natural language processing with bag-of-words representations, and kernel methods all give rise to large sparse matrices [20].

A thorough taxonomy of application areas is given by the Berkeley Parallel Computing Laboratory's pattern language [2].

3 Storage Formats for Sparse Matrices

The first practical challenge of sparse linear algebra is *storage*. A naive dense $n \times n$ array has n^2 total elements. For a matrix with a substantial number of zeroes, reaching a significantly more efficient storage is possible by only recording non-zero entries in the storage. There are a number of formats that have emerged. [1,2].

3.1 Coordinate Format (COO)

The simplest sparse storage format is the *coordinate* (COO) format, which stores three arrays of length nnz:

$$\texttt{row}[k], \quad \texttt{col}[k], \quad \texttt{val}[k], \qquad k = 0, 1, \ldots, \text{nnz} - 1,$$

where $A_{\texttt{row}[k],\,\texttt{col}[k]} = \texttt{val}[k]$. COO is easy to construct incrementally and is the natural output of an assembly process, but its access patterns are irregular and it is poorly suited for arithmetic operations such as matrix–vector multiplication without further processing.

An example of this format is as follows:

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

The matrix contains only four nonzero entries, so nnz $= 4$. Its COO representation is

$$\texttt{row} = \begin{bmatrix} 0 & 1 & 2 & 2 \end{bmatrix}, \qquad \texttt{col} = \begin{bmatrix} 0 & 1 & 0 & 2 \end{bmatrix}, \qquad \texttt{val} = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}.$$

3.2 Compressed Sparse Row (CSR)

The *Compressed Sparse Row* (CSR) format (also called CRS or Yale format) is by far the most widely used sparse matrix representation. It is obtained from COO by sorting the entries row-by-row and compressing the row index array into a shorter *row pointer* array [1,4]:

- $\texttt{val}[0 : \text{nnz} - 1]$: the non-zero values in row-major order;
- $\texttt{col\_ind}[0 : \text{nnz} - 1]$: the column index of each stored value;
- $\texttt{row\_ptr}[0 : m]$: $\texttt{row\_ptr}[i]$ is the index in $\texttt{val}$ of the first non-zero in row i.

For example, consider the sparse matrix introduced earlier,

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

The non-zero entries are read row-by-row, giving

$$\texttt{val} = [1,\ 2,\ 3,\ 4],$$

with corresponding column indices

$$\texttt{col\_ind} = [0,\ 1,\ 0,\ 2].$$

The row pointer array records where each row begins:

$$\texttt{row\_ptr} = [0,\ 1,\ 2,\ 4].$$

Thus, row 0 occupies position 0, row 1 occupies position 1, and row 2 occupies positions 2 through 3 in the `val` and `col_ind` arrays. The final entry $\texttt{row\_ptr}[3] = 4$ equals nnz, indicating the end of the stored data. [2, 14].

3.3 Compressed Sparse Column (CSC)

The *Compressed Sparse Column* (CSC) format is the transpose analogue of CSR: columns are the unit of compression. It stores `val`, `row_ind` (row index of each non-zero), and `col_ptr` (column pointer). CSC is natural for the computation $\mathbf{y} \leftarrow \mathbf{A}^\top \mathbf{x}$ and appears in column-oriented factorization codes.

3.4 Block and Diagonal Formats

When the non-zeros of $\mathbf{A}$ cluster into small dense blocks—a common occurrence in finite-element methods where multiple degrees of freedom are associated with each mesh node—*Block CSR* (BCSR) can dramatically reduce index overhead and improve cache utilization. The *diagonal* (DIA) format stores each occupied diagonal in a contiguous array and is especially efficient for regular stencil matrices; the *ELLPACK* (ELL) format pads each row to the same length for SIMD and GPU efficiency [2, 14].

The choice of storage format is a non-trivial, problem-specific decision that interacts with the target hardware architecture and the operations to be performed. Automatic format selection is itself a research topic [15].

4 Direct Solvers

A *direct solver* computes the solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ by factoring $\mathbf{A}$ explicitly—typically via LU, Cholesky (when $\mathbf{A}$ is symmetric positive definite), or QR decomposition—and then solving two triangular systems. For sparse matrices the central difficulty is *fill-in*: the factors L and U can have many more non-zeros than $\mathbf{A}$ itself, destroying sparsity and inflating both storage and computation.

4.1 The Fill-In Problem

Fill-in can be severe: even a matrix with $O(n)$ non-zeros can produce factors with $O(n^{3/2})$ or even $O(n^2)$ non-zeros, depending on structure. For the $N \times N$ matrix ($n = N^2$), Gaussian elimination produces $O(n^{3/2})$ fill-in and requires $O(n^2)$ operations [5].

An example of a direct solve using QR decomposition is the system

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

where we are solving for x in

$$Ax = b$$

Computing the QR factorization gives

$$A = QR,$$

where

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

Multiplying by Q^T yields

$$Rx = Q^T b = \begin{bmatrix} \frac{5}{\sqrt{2}} \\ \frac{5}{\sqrt{6}} \\ \frac{2}{\sqrt{3}} \end{bmatrix}.$$

The upper-triangular system is then solved by back substitution to obtain

$$x = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}.$$

An equivalent direct method is LU decomposition. We factor $A = LU$ where L is lower triangular with ones on the diagonal and U is upper triangular, then solve two triangular systems in sequence. Using the same matrix,

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

Step 1 — eliminate column 1. The pivot is $A_{11} = 1$. Only row 2 has a nonzero below it in column 1, so we compute the multiplier

$$\ell_{21} = \frac{A_{21}}{A_{11}} = \frac{1}{1} = 1,$$

and subtract ℓ_{21} times row 1 from row 2:

$$\text{row } 2 \leftarrow [1, 0, 1] - 1 \cdot [1, 1, 0] = [0, -1, 1].$$

Row 3 already has a zero in column 1, so no work is needed there. The matrix after step 1 is

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

Step 2 — eliminate column 2. The pivot is $A^{(1)}_{22} = -1$. Row 3 has a nonzero below it in column 2, so we compute

$$\ell_{32} = \frac{A^{(1)}_{32}}{A^{(1)}_{22}} = \frac{1}{-1} = -1,$$

and subtract ℓ_{32} times row 2 from row 3:

$$\text{row } 3 \leftarrow [0, 1, 1] - (-1) \cdot [0, -1, 1] = [0, 0, 2].$$

The matrix after step 2 is upper triangular — this is U:

$$U = \begin{bmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 2 \end{bmatrix}.$$

Assembling L. L is the identity with the multipliers placed below the diagonal in the column where they were computed:

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

One can verify that $LU = A$.

Forward substitution — solve $Ly = b$.

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 3 \end{bmatrix}.$$

Row by row:

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

Back substitution — solve $Ux = y$.

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix}.$$

Row by row from the bottom:

$$x_3 = \frac{4}{2} = 2,$$
$$x_2 = \frac{1 - 1 \cdot x_3}{-1} = \frac{1-2}{-1} = 1,$$
$$x_1 = 2 - 1 \cdot x_2 - 0 \cdot x_3 = 2 - 1 = 1.$$

The solution is

$$x = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix},$$

which matches the QR result exactly, as expected. Like QR decomposition, LU is a direct method: it reaches the solution in a finite number of algebraic steps. The advantage of LU over QR is efficiency — Q is a dense orthogonal matrix, whereas L preserves whatever sparsity A has (modulo fill-in), making LU the standard choice for large sparse systems. Unlike iterative methods, both LU and QR decomposition computes the solution in a finite sequence of algebraic operations and is therefore classified as a direct method. For smaller matrices, this is feasible. But for larger and sparse matrices, without any permutations, this is rather inefficient.

4.2 Graph-Based Reordering

Definition 4.1. Let $\mathbf{A} = LU$ be a factorization. A position (i, j) with $A_{ij} = 0$ but $L_{ij} \neq 0$ or $U_{ij} \neq 0$ is called a *fill-in entry*.

The key insight is that the fill-in pattern of $\mathbf{A} = LU$ is determined by the *elimination graph* of $\mathbf{A}$: permuting the rows and columns of $\mathbf{A}$ before factorization corresponds to relabeling the graph's vertices, which can drastically reduce fill-in.

For a sparse symmetric matrix we construct a graph by creating one vertex for each row (equivalently, column) and connecting vertices i and j whenever $A_{ij} \neq 0$. Throughout this section we use the running example

$$A = \begin{bmatrix} * & * & 0 & 0 & 0 & 0 \\ * & * & * & 0 & 0 & 0 \\ 0 & * & * & * & 0 & 0 \\ 0 & 0 & * & * & * & 0 \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \end{bmatrix},$$

a 6×6 tridiagonal matrix, which corresponds to the path graph

$$1 - 2 - 3 - 4 - 5 - 6.$$

Because A is tridiagonal, the only edges are $(i,\, i+1)$ for $i = 1, \ldots, 5$. We will trace how each ordering strategy handles this graph, including what fill-in (if any) is produced during elimination.

For context, eliminating vertex k from the current graph means

1. removing k and all edges incident to it, and

2. adding an edge between every pair of k's neighbors that are not already connected.

Each edge added in step 2 corresponds to a fill-in entry that appears in the factor L (or U). For the path graph $1-2-3-4-5-6$ the degree sequence is

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

so the endpoints have only one neighbor each; eliminating either introduces *no* fill.

Theorem 4.2. *Let* $\mathbf{A}$ *be a symmetric matrix with associated undirected graph* $G = (V, E)$*, where* $(i, j) \in E$ *if and only if* $A_{ij} \neq 0$ *for* $i \neq j$*. Let* $\sigma = (\sigma_1, \ldots, \sigma_n)$ *be an elimination ordering and let* $G^{(k)}$ *denote the elimination graph after vertices* $\sigma_1, \ldots, \sigma_{k-1}$ *have been eliminated. Then a position* (σ_i, σ_j) *with* $i < j$ *is a fill-in entry of the Cholesky factor* L *if and only if the edge* (σ_i, σ_j) *is a fill edge produced during the graph elimination process.*

Proof. We prove both directions by induction on the elimination step k.

Setup. At step k we eliminate vertex σ_k. Let $A^{(k)}$ denote the Schur complement of A after $k-1$ steps of Gaussian elimination, so $A^{(1)} = A$ and $A^{(k+1)}$ is obtained from $A^{(k)}$ by eliminating the pivot row and column corresponding to σ_k. Concretely, for $i, j \notin \{\sigma_1, \ldots, \sigma_k\}$,

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

The graph $G^{(k)}$ has vertex set $V \setminus \{\sigma_1, \ldots, \sigma_{k-1}\}$ and edge set that includes the original edges of $A^{(k)}$ plus any fill edges added in previous steps. By the inductive hypothesis (established below), the edges of $G^{(k)}$ correspond exactly to the nonzero off-diagonal entries of $A^{(k)}$.

Base case ($k = 1$). $A^{(1)} = A$ and $G^{(1)} = G$, so the edges of $G^{(1)}$ are precisely the nonzeros of A by construction.

Inductive step. Assume the edges of $G^{(k)}$ correspond exactly to the nonzero off-diagonal entries of $A^{(k)}$. We show the same holds for $G^{(k+1)}$.

When vertex σ_k is eliminated, the update formula shows that $A_{ij}^{(k+1)}$ can be nonzero in two ways:

1. $A_{ij}^{(k)} \neq 0$: the entry was already nonzero, so (i, j) is already an edge of $G^{(k)}$, and no new edge is needed.

2. $A_{ij}^{(k)} = 0$ but $A_{i\sigma_k}^{(k)} \neq 0$ and $A_{\sigma_k j}^{(k)} \neq 0$: the elimination introduces a new nonzero. Since $A^{(k)}$ is symmetric, this requires $i \neq \sigma_k$ and $j \neq \sigma_k$ to both be neighbors of σ_k in $G^{(k)}$.

Case 2 is exactly the condition under which the graph elimination rule adds a fill edge between i and j: both i and j are neighbors of σ_k and $(i, j) \notin E(G^{(k)})$. Therefore:

- every new nonzero in $A^{(k+1)}$ corresponds to a fill edge added to $G^{(k+1)}$, and

- every fill edge added to $G^{(k+1)}$ corresponds to a new nonzero in $A^{(k+1)}$.

Together with Case 1, the edges of $G^{(k+1)}$ correspond exactly to the nonzero off-diagonal entries of $A^{(k+1)}$, completing the induction.

Since the nonzeros that accumulate in $A^{(k)}$ across all steps are exactly the fill-in entries of L (the off-diagonal positions where $L_{ij} \neq 0$ but $A_{ij} = 0$), and we have shown these correspond one-to-one with fill edges in the graph elimination, the two notions of fill-in coincide. In particular, graph elimination produces no fill edges if and only if the factor L has no fill-in entries. □

4.2.1 Minimum Degree Ordering

The Minimum Degree (MD) heuristic repeatedly eliminates the vertex with the smallest current degree, on the grounds that low-degree vertices connect fewer neighbors and therefore create fewer fill edges. Modern implementations typically use the Approximate Minimum Degree (AMD) algorithm [8].

Step-by-step elimination on the example.

Rehashing the example: $A = \begin{bmatrix} * & * & 0 & 0 & 0 & 0 \\ * & * & * & 0 & 0 & 0 \\ 0 & * & * & * & 0 & 0 \\ 0 & 0 & * & * & * & 0 \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \end{bmatrix}$ is the matrix we are using.

1. **Graph:** $1 - 2 - 3 - 4 - 5 - 6$, degrees: $d(1) = 1, d(6) = 1, d(2) = d(3) = d(4) = d(5) = 2$.
   Minimum-degree vertices: 1 and 6 (both degree 1). Pick vertex 1.
   Neighbors of 1: $\{2\}$. No pair of neighbors to connect. **Fill: none.**

2. **Graph:** $2 - 3 - 4 - 5 - 6$, degrees: $d(2) = 1, d(6) = 1, d(3) = d(4) = d(5) = 2$.
   Minimum-degree: $\{2, 6\}$, pick vertex 6.
   Neighbors of 6: $\{5\}$. **Fill: none.**

3. **Graph:** $2 - 3 - 4 - 5$, degrees: $d(2) = 1, d(5) = 1, d(3) = d(4) = 2$.
   Minimum-degree: $\{2, 5\}$, pick vertex 2.
   Neighbors of 2: $\{3\}$. **Fill: none.**

4. **Graph:** $3 - 4 - 5$, degrees: $d(3) = 1, d(5) = 1, d(4) = 2$.
   Minimum-degree: $\{3, 5\}$, pick vertex 5.
   Neighbors of 5: $\{4\}$. **Fill: none.**

5. **Graph:** $3 - 4$. Eliminate 3, then 4. **Fill: none.**

The resulting MD ordering is $(1, 6, 2, 5, 3, 4)$ (one of several valid choices due to ties). Because the path graph's endpoints always have degree 1, every elimination step removes a leaf and creates no fill. The graph analysis therefore guarantees that the factors L and U have no fill-in entries beyond the nonzero pattern already present in $PAP^\top$ — no extra storage is needed beyond A itself.

4.2.2 Nested Dissection

Nested dissection takes a global view of the graph. It finds a small *separator* whose removal partitions the graph into roughly equal-sized independent subgraphs, recurses on each subgraph, and places the separator vertices last in the ordering. Because the separator is eliminated after all subgraph vertices, interactions between the two halves are postponed and fill-in between them is avoided.

Step-by-step on the example.

1. **Level 0 — full graph:** $1-2-3-4-5-6$.
   Choose vertex 3 (or 4) as a separator; it splits the graph into
   $$S_L = \{1,2\}, \qquad S_R = \{4,5,6\}, \qquad \text{separator} = \{3\}.$$
   Place the separator last at this level: $(\ldots, 3)$.

2. **Recurse on $S_L = \{1,2\}$:**
   Both vertices have degree 1 in S_L; pick vertex 1 as a trivial separator, yielding ordering $(1,2)$ for the left subgraph.

3. **Recurse on $S_R = \{4,5,6\}$:**
   Symmetric to S_L; ordering $(6,5,4)$.

4. **Assemble:** left block, right block, then separator:
   $$(1,\ 2,\ 6,\ 5,\ 4,\ 3).$$

The permuted matrix $PAP^\top$ with ordering $(1,2,6,5,4,3)$ has the block structure

$$PAP^\top = \begin{bmatrix} * & * & & & & 0 \\ * & * & & & & * \\ & & * & * & & 0 \\ & & * & * & * & 0 \\ & & & * & * & * \\ 0 & * & 0 & 0 & * & * \end{bmatrix},$$

where blank entries are structural zeros. The two diagonal blocks (rows/columns $\{1,2\}$ and $\{4,5,6\}$ in the new numbering) are decoupled except through the final row and column corresponding to the separator vertex 3. Eliminating the two blocks first produces no fill between them; the separator row is processed last and the resulting fill is confined to the small separator block.

For regular $\sqrt{n} \times \sqrt{n}$ grid graphs, nested dissection achieves $O(n^{3/2})$ fill-in in two dimensions and $O(n^2)$ in three dimensions [9], which is asymptotically optimal.

4.2.3 Reverse Cuthill–McKee Ordering

The Reverse Cuthill–McKee (RCM) algorithm reduces matrix *bandwidth* rather than directly minimizing fill-in. A narrow bandwidth concentrates nonzeros near the diagonal, which tends to reduce fill during factorization. The algorithm performs a breadth-first search (BFS) from a peripheral (high-eccentricity) starting vertex and then reverses the BFS order.

Step-by-step on the example.

1. **Choose a starting vertex.** Vertex 1 has degree 1 and eccentricity 5 (the maximum possible), so it is a natural peripheral vertex.

2. **BFS from vertex** 1**.**

Level	Vertices visited	Reason
0	1	starting vertex
1	2	neighbor of 1 not yet visited
2	3	neighbor of 2 not yet visited
3	4	neighbor of 3 not yet visited
4	5	neighbor of 4 not yet visited
5	6	neighbor of 5 not yet visited

   Cuthill–McKee order: $(1, 2, 3, 4, 5, 6)$.

3. **Reverse.**

$$\text{RCM order: } (6, 5, 4, 3, 2, 1).$$

The permuted matrix with this ordering is simply the original tridiagonal matrix with rows and columns relabeled in reverse order,

$$PAP^\top = \begin{bmatrix} * & * & 0 & 0 & 0 & 0 \\ * & * & * & 0 & 0 & 0 \\ 0 & * & * & * & 0 & 0 \\ 0 & 0 & * & * & * & 0 \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \end{bmatrix},$$

so the bandwidth is unchanged at 1 (one sub- and superdiagonal). For this simple path graph RCM offers no structural benefit, but on unstructured sparse graphs with irregular connectivity the reversal step consistently tightens the bandwidth by pulling strongly connected vertices closer to the diagonal.

For this tridiagonal example all four orderings produce zero fill, but they differ in how the nonzeros are distributed. Natural order and RCM preserve the narrow bandwidth, while MD and nested dissection spread nonzeros further from the diagonal. The advantage of MD and nested dissection becomes apparent on matrices whose sparsity graphs are not paths: for two-dimensional grid problems, the natural (row-major) ordering produces $O(n^{3/2})$ fill, whereas nested dissection keeps fill at $O(n^{3/2})$ with a much smaller constant, and AMD approaches nested dissection performance in practice.

5 Iterative Methods: Krylov Subspaces

Direct solvers compute $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ exactly via factorization, at the cost of fill-in that can be mitigated but often never fully eliminated by reordering. *Iterative* methods take a different

approach: rather than factoring $\mathbf{A}$, they generate a sequence of approximations $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ that converge toward the solution, using $\mathbf{A}$ only through matrix–vector products $\mathbf{A}$. For a sparse matrix this is a single $O(\mathrm{nnz}(\mathbf{A}))$ operation, so an iterative method never needs to store anything beyond $\mathbf{A}$ itself and a handful of length-n vectors—no fill-in is possible because no factorization is ever formed. The most important family of such methods is built on *Krylov subspaces*.

5.1 Definition

Definition 5.1. Let $A \in \mathbb{R}^{n\times n}$ and let $v \in \mathbb{R}^n$ be a nonzero vector. The *kth Krylov subspace* generated by A and v is

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

In the context of solving $\mathbf{Ax} = \mathbf{b}$ from an initial guess $\mathbf{x}_0$, the generating vector is taken to be the initial residual $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$, so that the subspaces of interest are $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ for $k = 1, 2, \ldots$. Each subspace is built entirely from repeated multiplication by $\mathbf{A}$; no inverse or factorization of $\mathbf{A}$ is ever required to construct it.

5.2 The Logic of Krylov Subspace Methods

The governing idea is to replace the linear system $\mathbf{Ax} = \mathbf{b}$ with an equivalent *optimization* problem. When $\mathbf{A}$ is symmetric positive definite (SPD), define the quadratic functional

$$\phi(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^\top \mathbf{Ax} - \mathbf{b}^\top \mathbf{x}.$$

Its gradient is $\nabla\phi(\mathbf{x}) = \mathbf{Ax} - \mathbf{b}$, so $\nabla\phi(\mathbf{x}) = 0$ if and only if $\mathbf{Ax} = \mathbf{b}$; moreover, since $\mathbf{A}$ is SPD, ϕ is strictly convex and has a unique global minimizer, which is exactly $\mathbf{x}^\star = \mathbf{A}^{-1}\mathbf{b}$. Solving $\mathbf{Ax} = \mathbf{b}$ is therefore equivalent to minimizing ϕ over $\mathbb{R}^n$.

A Krylov subspace method exploits this by minimizing ϕ *one direction at a time* over a nested, growing sequence of subspaces:

$$\mathbf{x}_0 + \mathcal{K}_1(\mathbf{A}, \mathbf{r}_0) \ \subset\ \mathbf{x}_0 + \mathcal{K}_2(\mathbf{A}, \mathbf{r}_0) \ \subset\ \cdots \ \subset\ \mathbf{x}_0 + \mathcal{K}_n(\mathbf{A}, \mathbf{r}_0) = \mathbb{R}^n,$$

where at step k the method picks $\mathbf{x}_k$ to be the point in $\mathbf{x}_0 + \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ that minimizes ϕ. Because each subspace contains the previous one, the method never loses progress made in earlier steps; because $\mathcal{K}_n(\mathbf{A}, \mathbf{r}_0) = \mathbb{R}^n$ (generically), the exact solution $\mathbf{x}^\star$ is guaranteed to be reached within n steps. This is the *Conjugate Gradient (CG)* method, applicable whenever $\mathbf{A}$ is SPD [6].

Rather than minimizing ϕ over the full growing subspace from scratch at each step (which would require re-solving an increasingly large problem), CG achieves this incrementally: at step k it moves from $\mathbf{x}_{k-1}$ along a single new search direction $\mathbf{p}_{k-1}$, chosen to be $\mathbf{A}$-conjugate to all previous directions ($\mathbf{p}_i^\top \mathbf{A}\mathbf{p}_j = 0$ for $i \neq j$). This conjugacy is what allows the one-dimensional line minimization performed at each step to also be the minimizer over the *entire* subspace so far, using only a short three-term recurrence and $O(n)$ storage.

5.3 The Optimal Step Size

At each step, having fixed a search direction $\mathbf{p}_{k-1}$, CG must choose how far to move along it. The following result shows that this reduces to an ordinary single-variable minimization.

Theorem 5.2. *Let $\mathbf{A}$ be SPD, let $\mathbf{x}_{k-1}$ be the current iterate with residual $\mathbf{r}_{k-1} = \mathbf{b} - \mathbf{A}\mathbf{x}_{k-1}$, and let $\mathbf{p}_{k-1}$ be a nonzero search direction. Define $\mathbf{x}(\alpha) = \mathbf{x}_{k-1} + \alpha\mathbf{p}_{k-1}$. Then $\phi(\mathbf{x}(\alpha))$ 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. Define $f(\alpha) = \phi(\mathbf{x}_{k-1} + \alpha\mathbf{p}_{k-1})$. Expanding ϕ and using symmetry of $\mathbf{A}$ (so that $\mathbf{x}_{k-1}^\top \mathbf{A}\mathbf{p}_{k-1} = \mathbf{p}_{k-1}^\top \mathbf{A}\mathbf{x}_{k-1}$),

$$\begin{aligned} f(\alpha) &= \tfrac{1}{2}(\mathbf{x}_{k-1} + \alpha\mathbf{p}_{k-1})^\top \mathbf{A}(\mathbf{x}_{k-1} + \alpha\mathbf{p}_{k-1}) - \mathbf{b}^\top(\mathbf{x}_{k-1} + \alpha\mathbf{p}_{k-1}) \\ &= \underbrace{\tfrac{1}{2}\mathbf{x}_{k-1}^\top \mathbf{A}\mathbf{x}_{k-1} - \mathbf{b}^\top \mathbf{x}_{k-1}}_{\text{constant in } \alpha} + \alpha\big(\mathbf{p}_{k-1}^\top \mathbf{A}\mathbf{x}_{k-1} - \mathbf{b}^\top \mathbf{p}_{k-1}\big) \;+\; \tfrac{1}{2}\alpha^2\, \mathbf{p}_{k-1}^\top \mathbf{A}\mathbf{p}_{k-1}. \end{aligned}$$

This is a quadratic in the scalar α. Differentiating termwise,

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

Since $\mathbf{r}_{k-1} = \mathbf{b} - \mathbf{A}\mathbf{x}_{k-1}$, we have $\mathbf{A}\mathbf{x}_{k-1} - \mathbf{b} = -\mathbf{r}_{k-1}$, so

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

Because $\mathbf{A}$ is SPD, $\mathbf{p}_{k-1}^\top \mathbf{A}\mathbf{p}_{k-1} > 0$ for $\mathbf{p}_{k-1} \neq 0$, so f is a strictly convex parabola in α and the unique critical point $f'(\alpha) = 0$ is its global minimum. Solving gives

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

□

At the first step $\mathbf{p}_0 = \mathbf{r}_0$, so Theorem 5.2 reduces to $\alpha_0 = \dfrac{\mathbf{r}_0^\top \mathbf{r}_0}{\mathbf{p}_0^\top \mathbf{A}\mathbf{p}_0}$. Together with the update rules

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

this constitutes the full Conjugate Gradient recurrence: each step costs one matrix–vector product and a fixed number of dot products, with no growth in storage as k increases—in sharp contrast to the fill-in growth of direct methods.

5.4 A Worked Example in Three Dimensions

Consider the sparse (tridiagonal) SPD system

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

$\mathbf{A}$ is SPD, since its leading principal minors are 2, 3, and 4, all positive. Direct solution gives $\mathbf{x}^\star = \left[\tfrac{1}{2},\ 0,\ \tfrac{3}{2}\right]^\top$. We now reach this same point using only Krylov subspaces $\mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3$, via CG, starting from $\mathbf{x}_0 = \mathbf{0}$.

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

$$\mathbf{A}\mathbf{p}_0 = [4, 8, 8]^\top, \qquad \mathbf{r}_0^\top \mathbf{r}_0 = 14, \qquad \mathbf{p}_0^\top \mathbf{A}\mathbf{p}_0 = 44, \qquad \alpha_0 = \tfrac{14}{44} = \tfrac{7}{22}.$$

$$\mathbf{x}_1 = \alpha_0 \mathbf{p}_0 = \left[\tfrac{7}{22},\ \tfrac{7}{11},\ \tfrac{21}{22}\right]^\top, \qquad \mathbf{r}_1 = \mathbf{r}_0 - \alpha_0 \mathbf{A}\mathbf{p}_0 = \left[-\tfrac{3}{11},\ -\tfrac{6}{11},\ \tfrac{5}{11}\right]^\top.$$

Step 2 ($k = 2$, subspace $\mathcal{K}_2(\mathbf{A}, \mathbf{r}_0)$).

$$\beta_0 = \frac{\mathbf{r}_1^\top \mathbf{r}_1}{\mathbf{r}_0^\top \mathbf{r}_0} = \frac{70/121}{14} = \frac{5}{121}, \qquad \mathbf{p}_1 = \mathbf{r}_1 + \beta_0 \mathbf{p}_0 = \tfrac{1}{121}\,[-28,\ -56,\ 70]^\top.$$

$$\mathbf{A}\mathbf{p}_1 = \tfrac{1}{121}[-112,\ -70,\ 84]^\top, \qquad \mathbf{p}_1^\top \mathbf{A}\mathbf{p}_1 = \tfrac{12936}{14641}, \qquad \alpha_1 = \frac{\mathbf{r}_1^\top \mathbf{r}_1}{\mathbf{p}_1^\top \mathbf{A}\mathbf{p}_1} = \tfrac{55}{84}.$$

$$\mathbf{x}_2 = \mathbf{x}_1 + \alpha_1 \mathbf{p}_1 = \left[\tfrac{1}{6},\ \tfrac{1}{3},\ \tfrac{4}{3}\right]^\top, \qquad \mathbf{r}_2 = \mathbf{r}_1 - \alpha_1 \mathbf{A}\mathbf{p}_1 = \left[\tfrac{1}{3},\ -\tfrac{1}{6},\ 0\right]^\top.$$

Note $\mathbf{x}_2 \neq \mathbf{x}^\star$: two dimensions are not yet enough for this generic right-hand side, since $\mathcal{K}_2(\mathbf{A}, \mathbf{r}_0)$ is a proper subspace of $\mathbb{R}^3$.

Step 3 ($k = 3$, subspace $\mathcal{K}_3(\mathbf{A}, \mathbf{r}_0) = \mathbb{R}^3$).

$$\beta_1 = \frac{\mathbf{r}_2^\top \mathbf{r}_2}{\mathbf{r}_1^\top \mathbf{r}_1} = \frac{5/36}{70/121} = \tfrac{121}{504}, \qquad \mathbf{p}_2 = \mathbf{r}_2 + \beta_1 \mathbf{p}_1 = \tfrac{1}{36}[10,\ -10,\ 5]^\top.$$

$$\mathbf{A}\mathbf{p}_2 = \tfrac{1}{36}[10,\ -5,\ 0]^\top, \qquad \mathbf{p}_2^\top \mathbf{A}\mathbf{p}_2 = \tfrac{25}{216}, \qquad \alpha_2 = \frac{\mathbf{r}_2^\top \mathbf{r}_2}{\mathbf{p}_2^\top \mathbf{A}\mathbf{p}_2} = \tfrac{6}{5}.$$

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

Convergence occurs in exactly $n = 3$ steps, matching the worst-case bound of Theorem 5.2's underlying guarantee ($\mathcal{K}_n(\mathbf{A}, \mathbf{r}_0) = \mathbb{R}^n$ in general position) exactly, using only three matrix–vector products $\mathbf{A}\mathbf{p}_0, \mathbf{A}\mathbf{p}_1, \mathbf{A}\mathbf{p}_2$ and no factorization of $\mathbf{A}$ whatsoever—in contrast to the $O(n^{3/2})$ fill-in that even a well-ordered direct factorization of a comparable tridiagonal system can incur at larger scale (Section 4).

5.5 GMRES

The *Generalized Minimal Residual* method (GMRES) is the standard Krylov-subspace analogue of CG for general nonsymmetric or indefinite linear systems [4, 7]. Its intent is simple: when the symmetry and positive-definiteness assumptions needed by CG are unavailable, GMRES still searches for an approximate solution inside an expanding Krylov subspace,

$$\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0) = \operatorname{span}\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{k-1}\mathbf{r}_0\},$$

but chooses the vector whose residual has the smallest Euclidean norm. In other words, if

$$\mathbf{x}_k \in \mathbf{x}_0 + \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0),$$

then GMRES imposes the minimization condition

$$\|\mathbf{b} - \mathbf{A}\mathbf{x}_k\|_2 = \min_{\mathbf{x} \in \mathbf{x}_0 + \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)} \|\mathbf{b} - \mathbf{A}\mathbf{x}\|_2.$$

Thus CG can be viewed as exploiting the special geometry of SPD matrices, while GMRES is designed to preserve the same Krylov idea in settings where that geometry is absent.

The price of this generality is storage. CG keeps only a few vectors because its search directions satisfy a short recurrence. GMRES, by contrast, typically builds an orthonormal basis of the full Krylov space using the Arnoldi process, so the memory and orthogonalization cost grow with k. In practice, this is often controlled by *restarted GMRES*, denoted GMRES(m), which runs m Krylov steps, restarts from the newest residual, and repeats. Restarting reduces memory but can slow or even stall convergence; this tension is one reason GMRES remains central in both practice and current research [21].

Example: a small nonsymmetric system. Let

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

This matrix is sparse and triangular, but it is not symmetric, so the CG theory developed above does not apply. Starting from $\mathbf{x}_0 = \mathbf{0}$, GMRES forms

$$\mathcal{K}_1(\mathbf{A}, \mathbf{b}) = \operatorname{span}\{\mathbf{b}\}, \qquad \mathcal{K}_2(\mathbf{A}, \mathbf{b}) = \operatorname{span}\{\mathbf{b}, \mathbf{A}\mathbf{b}\},$$

and at each step chooses the vector in $\mathbf{x}_0 + \mathcal{K}_k(\mathbf{A}, \mathbf{b})$ with the smallest residual norm. The point is not that this toy triangular system is hard to solve directly, but that it illustrates the core purpose of GMRES: the method keeps the sparse matrix accessible only through products with vectors, while replacing the SPD-specific CG optimality condition with residual minimization for general matrices.

6 Open Problems in Sparse Linear Algebra

The conjugate gradient method presented above is, in a precise sense, a solved problem: its convergence theory, optimality within Krylov subspaces, and finite termination are all rigorously established. But CG is the entry point into a much larger landscape of sparse linear algebra where fundamental questions remain open — spanning iterative solver theory, eigenvalue computation, low-rank approximation, and randomized methods. This section surveys several active directions, most of which were formalized as explicit open problems at a 2025 Simons Institute workshop bringing together the theoretical computer science and numerical linear algebra communities.

Iterative solvers beyond the symmetric positive-definite case. CG's short-recurrence structure and clean convergence bound rely essentially on A being symmetric positive definite. For nonsymmetric systems, methods like GMRES are used instead, but their convergence behavior is far less understood: it depends not just on eigenvalues but on the

conditioning of eigenvectors, and no fully satisfying convergence theory analogous to CG's exists. A central open problem is to find a descriptive convergence bound for GMRES applied to nonnormal systems that accurately captures its transient behavior early in the iteration — practitioners routinely observe fast convergence for such systems that current theory cannot fully explain, even though it can be linked heuristically to clustering of eigenvalues.

Randomized alternatives to Krylov methods. A growing body of work asks whether *sketch-and-project* methods (randomized row-action schemes built on random projections) can outperform classical Krylov solvers like CG on certain problem classes. The comparison is subtle: an open question asks what conditions on the eigenvalues of A ensure a sketch-and-project method converges to a given accuracy in fewer epochs than conjugate gradient, where an "epoch" is normalized to make the comparison fair (one CG epoch = one matrix-vector product; one sketch-and-project epoch = one pass over n randomly sampled rows). Even the simplest instance of this question — comparing plain CG against randomized coordinate descent on matrices with polynomially decaying eigenvalues — is unresolved.

Finite-precision behavior of Krylov methods. In exact arithmetic, CG and the closely related Lanczos algorithm are mathematically equivalent and terminate in at most n steps. In floating-point arithmetic, both properties degrade — the computed residuals are not the true residuals $b - Ax_k$, and finite termination is no longer guaranteed. While partial explanations exist (dating to foundational work by C. Paige and A. Greenbaum), a full account of exactly which properties of A guarantee that the computed residual norms drop below machine precision, and precisely how many bits of precision CG needs to reach a target backward error in at most n steps, remain open.

Sparse matrix–vector product complexity at scale. On the computational side, as opposed to the numerical-analysis side, the central bottleneck for sparse linear algebra is that these computations are memory-bound with low arithmetic intensity: accelerating sparse matrix-vector and matrix-matrix products is a major current research focus, complicated by the sheer scale of the matrices involved — with unknown counts in some large-scale problems now reaching into the tens of billions. This creates open engineering and algorithmic questions around data layout, partitioning for parallelism, and exploiting sparsity structure across GPUs and distributed systems, in addition to the purely mathematical convergence questions above.

A concrete historical open problem: the Forsythe conjecture. Not all open questions are recent. In 1968, Forsythe studied a restarted variant of the steepest-descent/CG method and observed — but did not fully prove — that the normalized residual directions eventually oscillate between two limiting directions when the method is restarted with a fixed subspace dimension $s \geq 2$. This is exactly the mechanism the finite-termination proof above rules out for *unrestarted* CG (which keeps expanding $\mathcal{K}_k$ every step) — but for restarted variants, used in practice to bound memory costs, the analogous long-run behavior is still not fully understood beyond a few small cases of s.

Why this matters for the method in this paper. These open problems are not tangential to the CG algorithm derived above — they are direct extensions of the same object. The finite-termination proof shows $x_n = x^\star$ exactly *in exact arithmetic*, for symmetric positive-definite A, with no restart. Nearly every open direction above asks what happens when one of those three assumptions is relaxed: nonsymmetric A (GMRES theory), finite-precision arithmetic (loss of exact termination), or restarted/randomized variants (sketch-and-project, Forsythe's oscillation). Framed this way, the worked example given here is a clean base case sitting inside a much larger, still-unsettled theory.

References

[1] Wikipedia contributors, *Sparse matrix*, Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/wiki/Sparse_matrix, accessed 2025.

[2] Berkeley Parallel Computing Laboratory, *Sparse Linear Algebra*, Our Pattern Language, https://patterns.eecs.berkeley.edu/?page_id=202, accessed 2025.

[3] T. Betcke, *The Need for Sparse Linear Algebra—A PDE Example*, in *Techniques of High-Performance Computing: Lecture Notes*, https://tbetcke.github.io/hpc_lecture_notes/sparse_linalg_pde.html, accessed 2025.

[4] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003.

[5] T. A. Davis, *Direct Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2006.

[6] M. R. Hestenes and E. Stiefel, Methods of gradients for solving linear systems, *J. Res. Natl. Bur. Stand.*, **49**(6):409–436, 1952.

[7] Y. Saad and M. H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput.*, **7**(3):856–869, 1986.

[8] P. R. Amestoy, T. A. Davis, and I. S. Duff, Algorithm 837: AMD, an approximate minimum degree ordering algorithm, *ACM Trans. Math. Softw.*, **30**(3):381–388, 2004.

[9] A. George, Nested dissection of a regular finite element mesh, *SIAM J. Numer. Anal.*, **10**(2):345–363, 1973.

[10] A. Brandt, S. F. McCormick, and J. W. Ruge, Algebraic multigrid (AMG) for sparse matrix equations, in D. J. Evans (ed.), *Sparsity and Its Applications*, Cambridge Univ. Press, 1985.

[11] K. Stüben, A review of algebraic multigrid, *J. Comput. Appl. Math.*, **128**(1–2):281–309, 2001.

[12] N. Bell, L. N. Olson, J. Schroder, and B. Southworth, PyAMG: Algebraic multigrid solvers in Python, *J. Open Source Softw.*, **8**(87):5495, 2023.

[13] A. Toselli and O. Widlund, *Domain Decomposition Methods—Algorithms and Theory*, Springer, Berlin, 2005.

[14] J. Gao, B. Liu, W. Ji, and H. Huang, A systematic literature survey of sparse matrix–vector multiplication, *arXiv:2404.06047*, 2024.

[15] C. Stylianou and M. Weiland, Optimizing sparse linear algebra through automatic format selection and machine learning, *arXiv:2303.05098*, 2023.

[16] O. G. Ernst and M. J. Gander, Why it is difficult to solve Helmholtz problems with classical iterative methods, in *Numerical Analysis of Multiscale Problems*, Lecture Notes Comput. Sci. Eng. **83**, Springer, 2011, pp. 325–363.

[17] Various authors, Linear systems and eigenvalue problems: Open questions from a Simons workshop, *arXiv:2602.05394*, 2025.

[18] P.-G. Martinsson and J. A. Tropp, Randomized numerical linear algebra: Foundations and algorithms, *Acta Numerica*, **29**:403–572, 2020.

[19] H. Elman, D. Silvester, and A. Wathen, *Finite Elements and Fast Iterative Solvers*, Oxford Univ. Press, 2nd ed., 2014.

[20] T. A. Davis, *SuiteSparse: A Suite of Sparse Matrix Software*, https://people.engr.tamu.edu/davis/suitesparse.html, accessed 2025.

[21] M. Embree, How descriptive are GMRES convergence bounds?, *arXiv:2209.01231*, 2022.