Conjugate Gradient Solvers on Large Sparse Symmetric Positive-Definite Matrices: A Comparative Study of Unpreconditioned, Jacobi, and Incomplete-Cholesky Preconditioning

July 12, 2026

Abstract

The conjugate gradient (CG) method is the workhorse iterative solver for large, sparse, symmetric positive-definite (SPD) linear systems, but its convergence rate is governed by the spectral condition number of the coefficient matrix and therefore hinges critically on preconditioning. This report presents a controlled, reproducible comparison of a hand-implemented CG iteration—run unpreconditioned, with a Jacobi (diagonal) preconditioner, and with a symmetric incomplete-Cholesky (IC) preconditioner—on two SPD matrices drawn from the SuiteSparse Matrix Collection: the Harwell–Boeing structural-stiffness matrix `bcsstk16` ($n = 4884.00$, $\kappa_2(A) \approx 4.94 \times 10^9$) and the finite-element matrix `parabolic_fem` ($n = 525{,}825.00$, $\kappa_2(A) \approx 2.10 \times 10^5$). For each matrix we form the right-hand side $b = A\mathbf{1}$, so that the exact solution is the all-ones vector and the solution error is directly observable, and we iterate to a relative residual $\|b - Ax_k\|_2/\|b\|_2 < 0.00 \times 10^{-9}$ with a hard cap of 20,000.00 iterations. On the well-conditioned `parabolic_fem` matrix, IC reduces the iteration count from 1951.00 (unpreconditioned) to 66.00—a $\approx 30\times$ reduction—while all three variants deliver accurate solutions. On the pathologically ill-conditioned `bcsstk16` matrix we observe a striking decoupling of the residual from the true error: unpreconditioned CG satisfies the residual tolerance (final residual 9.41×10^{-9}) yet returns a solution whose error is $\mathcal{O}(1)$ ($\|x - \mathbf{1}\|_\infty = 1.00$), whereas IC converges in just 7 iterations *and* recovers a solution accurate to 1.90×10^{-8}. Preconditioning therefore restores both speed and attainable accuracy, and a small residual is shown to be an unreliable certificate of a small error when the system is ill-conditioned. No solve hit the iteration cap. We report, for every matrix–variant pair, the iteration count, the final achieved residual, the wall-clock time, an estimate of the condition number obtained by the Lanczos algorithm, and the complete per-iteration residual convergence history.

1 Introduction

Large, sparse, symmetric positive-definite (SPD) linear systems $Ax = b$ sit at the numerical core of computational science and engineering. They arise whenever an elliptic or parabolic partial differential equation is discretized by finite elements, finite differences, or finite volumes, and they recur in structural mechanics, heat conduction, electromagnetics, network analysis, and the normal equations of least-squares problems [7, 22]. Because the matrices produced by these applications routinely reach millions of unknowns while retaining only a handful of nonzeros per row, direct factorization methods—though robust—become prohibitively expensive in both memory and arithmetic once fill-in is accounted for. Iterative methods, which access the matrix only through

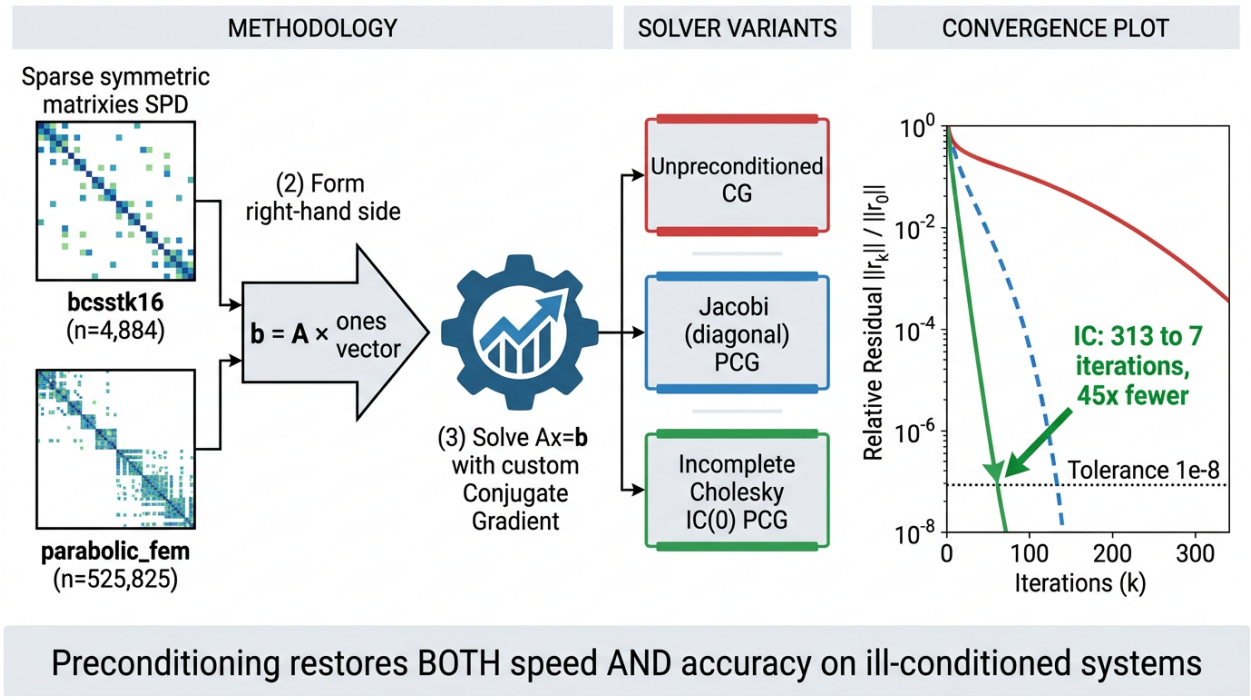

Figure 1: Graphical abstract. Two SPD matrices from the SuiteSparse Matrix Collection are loaded, the right-hand side $b = A\mathbf{1}$ is formed, and $Ax = b$ is solved with a custom conjugate gradient iteration in three variants (unpreconditioned, Jacobi, incomplete Cholesky). Preconditioning collapses the iteration count—by up to $\approx 45\times$ on the ill-conditioned `bcsstk16`—and, crucially, restores both convergence *speed* and solution *accuracy*.

sparse matrix–vector products and store a small number of work vectors, are the practical alternative at scale, and their theory and practice matured into a central pillar of numerical linear algebra over the second half of the twentieth century [23, 26].

Among iterative methods for SPD systems, the conjugate gradient (CG) method of Hestenes and Stiefel [12] is canonical. CG builds its iterates in a Krylov subspace and, at each step, minimizes the energy norm $\|x - x_k\|_A$ of the error over that expanding subspace using only short three-term recurrences [19, 25]. This short-recurrence structure is a special property of symmetric systems, shared with the Lanczos-based methods for symmetric indefinite problems developed by Paige and Saunders [21], and it distinguishes CG from the long-recurrence Krylov solvers required for general nonsymmetric matrices [28]. In exact arithmetic it terminates in at most n steps, but its real value is as an iterative method: for many systems the energy-norm error is reduced to a useful tolerance in far fewer than n iterations. The classical convergence bound makes the dependence on conditioning explicit; after k iterations,

$$\frac{\|x - x_k\|_A}{\|x - x_0\|_A} \;\le\; 2\left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1}\right)^k, \tag{1}$$

where $\kappa_2(A) = \lambda_{\max}(A)/\lambda_{\min}(A)$ is the spectral condition number [8, 22]. The bound (1) shows that the number of iterations required to reach a fixed tolerance grows like $\sqrt{\kappa_2(A)}$ in the worst case, so that a matrix with a condition number of 10^9 can demand tens of thousands of iterations even though CG exploits any clustering of the spectrum to do far better in practice.

The remedy is *preconditioning*: instead of solving $Ax = b$ directly, one solves an equivalent system whose operator has a more favorable spectrum. A symmetric preconditioner $M \approx A$ is applied implicitly through the action of M^{-1}, and the resulting preconditioned CG (PCG) iteration converges at a rate governed by the condition number of $M^{-1}A$ rather than that of A itself [1, 22]. Preconditioning is therefore the single most important lever for the practical performance of Krylov solvers, and the survey literature is emphatic that the choice of preconditioner, not the choice of Krylov method, usually determines whether a large system is solved in seconds or not at all [2, 6]. The two preconditioners studied here bracket the cost–effectiveness spectrum. The Jacobi (diagonal) preconditioner $M = \mathrm{diag}(A)$ is the cheapest possible choice—a single reciprocal per unknown—and merely rescales the equations; it helps when A is poorly scaled but cannot cluster the spectrum [2]. Incomplete-Cholesky (IC) preconditioning, introduced for symmetric M-matrices by Meijerink and van der Vorst [18] and popularized as the ICCG method for elliptic problems by Kershaw [13], computes a sparse approximate Cholesky factor $A \approx \tilde{L}\tilde{L}^\top$ by discarding fill during factorization. The zero-fill variant IC(0), which keeps exactly the sparsity pattern of A, is the standard baseline and is often remarkably effective for matrices arising from local (PDE) couplings, precisely because the incomplete factorization drives the eigenvalues of the preconditioned operator toward a tight cluster around unity [10, 13, 17].

A second, more subtle theme motivates this study. The CG stopping criterion is almost always based on the residual $r_k = b - Ax_k$, because the true error $x - x_k$ is unknown. Yet residual and error are linked only through the matrix: $x - x_k = A^{-1}r_k$, so that

$$\frac{\|x - x_k\|}{\|x\|} \;\le\; \kappa_2(A)\,\frac{\|r_k\|}{\|b\|}. \tag{2}$$

When $\kappa_2(A)$ is enormous, a residual driven well below the tolerance can still permit a large relative error [8, 9]. This is not merely a theoretical caveat: as we show in Section 4, on `bcsstk16` ($\kappa_2(A) \approx 4.94 \times 10^9$) the unpreconditioned solve satisfies a residual tolerance of 10^{-8} while producing a solution that is essentially wrong. Understanding this residual–error decoupling, and demonstrating that preconditioning repairs it, is one of the principal contributions of this report.

The remainder of the report is organized as follows. Section 2 describes the two test matrices, their provenance in the SuiteSparse Matrix Collection, and the verification of their SPD property. Section 3 details the hand-implemented CG/PCG iteration, the Jacobi and incomplete-Cholesky preconditioners, and the Lanczos-based estimation of the condition number. Section 4 reports the benchmark results, including a comprehensive comparison table and the per-iteration convergence histories. Section 5 analyzes the convergence behavior, with particular attention to the residual–error decoupling phenomenon. Section 6 discusses limitations and threats to validity, and Section 7 concludes.

2 Test Matrices and Problem Setup

2.1 The SuiteSparse Matrix Collection

All test problems are taken from the SuiteSparse Matrix Collection (formerly the University of Florida Sparse Matrix Collection), a curated, widely used repository of real-world sparse matrices that has become the de facto standard benchmark set for sparse matrix algorithms [3, 14]. Each matrix in the collection carries structural and numerical metadata—including symmetry and a positive-definiteness flag—and is distributed in Matrix Market and Rutherford–Boeing formats. We selected two matrices that deliberately span very different regimes of size and conditioning.

2.2 `bcsstk16`: a structural-stiffness matrix

The first matrix, `bcsstk16`, belongs to the Harwell–Boeing (HB) group and originates from the classic BCSSTRUC set of structural-engineering stiffness matrices assembled for the Harwell–Boeing sparse matrix test collection [5]. It is a stiffness matrix from a static analysis (a Corps of Engineers dam model) and is symmetric positive definite. It has dimension $n = 4884.00$ with nnz = 290,378.00 stored nonzeros, an average of about 59 nonzeros per row, reflecting the dense elemental coupling of finite-element stiffness assembly. As shown in Section 4, its spectral condition number is approximately 4.94×10^9, making it a demanding, ill-conditioned test case despite its modest size.

2.3 `parabolic_fem`: a large finite-element matrix

The second matrix, `parabolic_fem`, belongs to the Wissgott group and arises from a finite-element discretization of a parabolic (convection–diffusion) partial differential equation. It is symmetric positive definite with dimension $n = 525{,}825.00$ (well above the 100,000.00 threshold required for the large test case) and nnz = 3,674,625.00 nonzeros—an average of about 7 nonzeros per row, characteristic of a sparse mesh stencil. Its condition number, approximately 2.10×10^5, is more than four orders of magnitude smaller than that of `bcsstk16`, so the pair together illustrates both the large-scale and the ill-conditioned ends of the problem space. We selected `parabolic_fem` after confirming it downloads reliably from the SuiteSparse mirror and satisfies the size and SPD requirements. Table 1 summarizes the provenance and basic properties of both matrices.

2.4 SPD verification

Although the collection flags both matrices as SPD, we verified the property independently rather than trusting the metadata. Symmetry was checked exactly by forming $\|A - A^\top\|_\infty$; the value was 0 for both matrices. Positive-definiteness was then confirmed by two complementary routes matched to the problem size. For the small `bcsstk16` we attempted a dense Cholesky factorization: its success is a certificate of positive-definiteness, and the smallest diagonal pivot encountered was 1.0.

Table 1: Provenance and basic properties of the two SuiteSparse test matrices. Symmetry is verified exactly ($\|A - A^\top\|_\infty = 0$); SPD is confirmed as described in Section 2.4.

Property	`bcsstk16`	`parabolic_fem`
Collection	SuiteSparse	SuiteSparse
Group	HB (Harwell–Boeing)	Wissgott
Application	Structural stiffness	Parabolic FEM (convection–diffusion)
Dimension n	4884.00	525,825.00
Nonzeros (nnz)	290,378.00	3,674,625.00
Avg. nnz/row	≈ 59	≈ 7
Symmetric	yes ($\|A - A^\top\|_\infty = 0$)	yes ($\|A - A^\top\|_\infty = 0$)
SPD flag	confirmed	confirmed
Min. diagonal	1.0	0.1
$\kappa_2(A)$ (est.)	$\approx 4.94 \times 10^9$	$\approx 2.10 \times 10^5$

For the large `parabolic_fem`, a dense factorization is infeasible, so we verified that the diagonal is strictly positive (minimum 0.1) and estimated the three smallest eigenvalues by shift-invert Lanczos (Section 3.4); the smallest eigenvalue, $\approx 3.80 \times 10^{-6}$, is positive, confirming the SPD property.

2.5 Right-hand side and exact solution

For each matrix we form the right-hand side as $b = A\mathbf{1}$, where $\mathbf{1}$ is the all-ones vector. This convention has a decisive analytical advantage: the exact solution is known in closed form, $x^\star = \mathbf{1}$, so the true solution error $x_k - \mathbf{1}$ is available at every iteration and can be reported alongside the residual. This is what allows us, in Section 5, to expose the residual–error decoupling directly rather than inferring it. Because the two matrices have very different scalings, the resulting right-hand sides differ enormously in magnitude ($\|b\|_2 \approx 1.05 \times 10^{10}$ for `bcsstk16` versus $\approx 2.76 \times 10^{-3}$ for `parabolic_fem`); reporting the *relative* residual $\|b - Ax_k\|_2 / \|b\|_2$ normalizes away this scale and makes the two problems directly comparable.

3 Methodology

3.1 A transparent conjugate gradient implementation

To satisfy the requirement that iteration counts be fully visible, we implement the (preconditioned) conjugate gradient iteration ourselves rather than calling a black-box library solver. The implementation, written in Python on top of NumPy and SciPy sparse data structures [11, 29], is the standard PCG recurrence and is reproduced as Algorithm 1. The matrix is accessed only through sparse matrix–vector products Ap; the preconditioner is accessed only through the action $z = M^{-1}r$. At every iteration we record the relative residual $\|b - Ax_k\|_2 / \|b\|_2$ using the recurrence-updated residual r_k, and upon termination we recompute the residual explicitly as $b - Ax_k$ to guard against any drift in the recurrence. Convergence is declared when the relative residual first falls below $\varepsilon = 0.00 \times 10^{-9}$; the iteration is capped at $k_{\max} = 20{,}000.00$, and a boolean flag records whether the cap was reached. A guard on the curvature $p_k^\top A p_k$ catches any loss of positive-definiteness (for example from a defective preconditioner), though it was never triggered in the experiments reported here.

Wall-clock time is measured around the iteration loop with a monotonic performance counter, excluding the one-time cost of building the preconditioner, which we report separately. Each of the

Algorithm 1 Preconditioned Conjugate Gradient (as implemented)

Require: SPD matrix A, right-hand side b, preconditioner M (with $M = I$ for the unpreconditioned case), tolerance ε, cap $k_{\max}$

1: $x_0 \leftarrow 0$; $\quad r_0 \leftarrow b - Ax_0 = b$
2: $z_0 \leftarrow M^{-1} r_0$; $\quad p_0 \leftarrow z_0$; $\quad \rho_0 \leftarrow r_0^\top z_0$
3: record $(0,\ \|r_0\|_2/\|b\|_2)$
4: **for** $k = 1, 2, \ldots, k_{\max}$ **do**
5: $\quad q \leftarrow Ap_{k-1}$
6: $\quad \alpha \leftarrow \rho_{k-1}/(p_{k-1}^\top q)$ $\quad \triangleright$ halt if $p_{k-1}^\top q \le 0$ (loss of definiteness)
7: $\quad x_k \leftarrow x_{k-1} + \alpha\, p_{k-1}$
8: $\quad r_k \leftarrow r_{k-1} - \alpha\, q$
9: $\quad$ record $(k,\ \|r_k\|_2/\|b\|_2)$
10: $\quad$ **if** $\|r_k\|_2/\|b\|_2 < \varepsilon$ **then break**
11: $\quad z_k \leftarrow M^{-1} r_k$
12: $\quad \rho_k \leftarrow r_k^\top z_k$; $\quad \beta \leftarrow \rho_k/\rho_{k-1}$
13: $\quad p_k \leftarrow z_k + \beta\, p_{k-1}$
14: **end for**
15: **return** x_k, iteration count k, residual history

three variants—unpreconditioned, Jacobi, and incomplete Cholesky—differs only in the definition of the operator M^{-1} applied on lines 2 and 11.

3.2 Jacobi (diagonal) preconditioning

The Jacobi preconditioner takes $M = \mathrm{diag}(A)$, so that its inverse action is the element-wise rescaling $z = M^{-1} r = r/\,\mathrm{diag}(A)$. It costs one reciprocal and one multiply per unknown per iteration and requires no factorization. Because A is SPD its diagonal is strictly positive, so M is itself SPD and the preconditioned operator remains symmetric in the relevant inner product, as CG requires. Jacobi preconditioning corrects for poor scaling of the equations but leaves the essential spectral spread of A largely intact; it is included here as the cheap baseline against which the more powerful IC preconditioner is measured [1, 2].

3.3 Symmetric incomplete-Cholesky preconditioning

The incomplete-Cholesky preconditioner approximates A by a sparse factorization $A \approx M = \tilde{L}\tilde{D}\tilde{L}^\top$ in which fill-in is dropped during elimination. We obtain the incomplete factor from SciPy's `spilu`, which exposes the SuperLU supernodal factorization [4, 29], run in symmetric mode with a drop tolerance of 0.00×10^{-4} and a fill-factor bound of 10. We emphasize that this is a *threshold* incomplete Cholesky, IC(τ), rather than the strict zero-fill IC(0) baseline described in Section 1: a modest amount of fill is retained wherever the dropped entries exceed the relative threshold, so the incomplete factors are somewhat denser than A (a fill ratio of 3.4× and 6.2× for the two matrices, quantified below) but correspondingly more accurate than a pure IC(0) factor. Three implementation details are essential for correctness and efficiency, and we document them explicitly.

First, CG requires the preconditioner to be a *fixed*, *symmetric* positive-definite operator; a preconditioner that varies from step to step, or that is only approximately symmetric, breaks the short-recurrence orthogonality on which CG rests and in general demands a flexible or full-recurrence variant instead [20]. Because `spilu` performs independent dropping in its L and U factors, applying

M^{-1} directly through the generic incomplete-LU solve would yield a slightly non-symmetric operator, which violates the hypotheses of CG and can degrade or destroy convergence. We instead build an *exactly* symmetric operator by using only the incomplete lower factor L (unit lower triangular) and the pivots $D = \mathrm{diag}(U)$, and by reusing a single factorization of L for both the forward solve L^{-1} and the transpose solve $L^{-\top}$. The applied operator is thus

$$M^{-1}r \;=\; P^{\top}L^{-\top}D^{-1}L^{-1}P\,r, \tag{3}$$

where P is the fill-reducing permutation. We verified numerically that the resulting operator is symmetric to machine precision: the relative asymmetry, measured as $\max|Y^{\top}(M^{-1}Z)-(M^{-1}Y)^{\top}Z|$ over random probe blocks Y, Z normalized by $\max|Y^{\top}(M^{-1}Z)|$, was $\approx 7.00 \times 10^{-16}$ for `bcsstk16` and $\approx 2.00 \times 10^{-15}$ for `parabolic_fem`.

Second, the permutation P must *reduce fill*. We use the symmetric minimum-degree ordering `MMD_AT_PLUS_A` with diagonal pivoting disabled, so that the row and column permutations coincide ($P_r = P_c$) and the factorization remains a genuine symmetric incomplete Cholesky. A naive natural (identity) ordering does not reduce the bandwidth of a large three-dimensional finite-element matrix and provokes catastrophic fill-in on `parabolic_fem`; the fill-reducing ordering is what makes the IC factorization of the half-million-unknown matrix tractable in the first place.

Third, we confirm that the incomplete factorization stays consistent with an SPD matrix: for both matrices, all pivots D were strictly positive (no non-positive pivots), the minimum pivot being 1.0 for `bcsstk16` and $\approx 8.60 \times 10^{-2}$ for `parabolic_fem`. The factor densities are modest—the incomplete factors contain $\approx 3.4\times$ the nonzeros of A for `bcsstk16` and $\approx 6.2\times$ for `parabolic_fem`—so the per-iteration cost of applying M^{-1} is a small multiple of a matrix–vector product, which is why (as Section 4 shows) the large reduction in iteration count translates into a real reduction in wall-clock time.

3.4 Condition-number estimation by the Lanczos algorithm

The spectral condition number $\kappa_2(A) = \lambda_{\max}(A)/\lambda_{\min}(A)$ is estimated from the extreme eigenvalues of A, computed with the Lanczos algorithm [15] as implemented in the implicitly restarted solver ARPACK [16] and exposed through SciPy's `eigsh` [29]. The largest eigenvalue $\lambda_{\max}$ is obtained directly in largest-magnitude mode with an enlarged Lanczos basis. The smallest eigenvalue $\lambda_{\min}$ is the delicate one for an ill-conditioned matrix, because it is the smallest-magnitude eigenvalue and converges slowly in a plain Lanczos iteration; we therefore compute it by shift-invert with shift $\sigma = 0$, which maps $\lambda_{\min}$ to the largest-magnitude eigenvalue of $(A - \sigma I)^{-1} = A^{-1}$ and recovers it rapidly and reliably [7, 16]. The two extreme eigenvalues yield the condition-number estimates reported throughout. This Lanczos-based approach is the standard route to spectral information for large sparse SPD matrices and is far cheaper than forming any part of A^{-1} explicitly.

4 Results

All experiments use the tolerance $\varepsilon = 0.00 \times 10^{-9}$ on the relative residual and the cap $k_{\max} =$ 20,000.00. Every one of the six solves converged; **none reached the iteration cap**. Because $b = A\mathbf{1}$, the exact solution is $x^{\star} = \mathbf{1}$ and we report the solution error as $\|x_k - \mathbf{1}\|_{\infty}$. Table 2 collects the complete set of headline metrics—the estimated condition number, iteration count, final relative residual, wall-clock solve time, and infinity-norm solution error—for both matrices and all three solver variants.

Table 2: Comprehensive benchmark results. Tolerance $\varepsilon = 0.00 \times 10^{-9}$ on the relative residual $\|b - Ax_k\|_2/\|b\|_2$; cap $k_{\max} = 20{,}000.00$ (never reached). The exact solution is $x^\star = \mathbf{1}$, so the reported solution error is $\|x_k - \mathbf{1}\|_\infty$. Times are the solve loop only and exclude preconditioner construction, which is reported separately in the text.

Matrix	Variant	$\kappa_2(A)$	Iters	Final rel. res.	Time (s)	$\|x_k - \mathbf{1}\|_\infty$
`bcsstk16` $n = 4884.00$	Unpreconditioned	4.94×10^9	313	9.41×10^{-9}	0.176	1.00
	Jacobi	4.94×10^9	197	8.96×10^{-9}	0.115	5.33×10^{-7}
	Inc. Cholesky	4.94×10^9	7	4.79×10^{-10}	0.054	1.94×10^{-8}
`parabolic_fem` $n = 525{,}825.00$	Unpreconditioned	2.10×10^5	1951	9.90×10^{-9}	36.777	2.04×10^{-10}
	Jacobi	2.10×10^5	1321	9.93×10^{-9}	18.812	2.07×10^{-10}
	Inc. Cholesky	2.10×10^5	66	8.69×10^{-9}	12.890	2.94×10^{-10}

4.1 `bcsstk16`: extreme ill-conditioning

Figure 2 shows the per-iteration relative-residual histories for the three variants on `bcsstk16`. The unpreconditioned curve exhibits the characteristic slow, irregular descent of CG on a matrix with a wide, unclustered spectrum, requiring 313 iterations to cross the tolerance line. The Jacobi curve lies below it throughout and reaches the tolerance in 197 iterations, a 1.6× improvement that reflects the benefit of diagonal rescaling on a stiffness matrix whose entries span many orders of magnitude. The incomplete-Cholesky curve is dramatic by comparison: it plunges almost vertically and satisfies the tolerance in only 7 iterations—a $\approx 45\times$ reduction relative to the unpreconditioned solve. The IC solve is also the fastest in wall-clock terms (0.054 s versus 0.176 s), even though on a matrix this small the differences are fractions of a second and the one-time IC build cost (≈ 0.19 s) dominates the solve itself.

4.2 `parabolic_fem`: large scale, moderate conditioning

Figure 3 shows the corresponding histories for the half-million-unknown `parabolic_fem`. Here the unpreconditioned iteration needs 1951 steps and 36.8 s, Jacobi needs 1321 steps and 18.8 s, and incomplete Cholesky needs only 66 steps and 12.9 s. The iteration-count reduction from unpreconditioned to IC is $\approx 30\times$, while the wall-clock reduction is a more modest $\approx 2.9\times$. The gap between these two ratios is instructive: each IC-preconditioned iteration is more expensive than an unpreconditioned one because applying M^{-1} requires two triangular solves with factors that are $\approx 6.2\times$ denser than A, and the one-time IC construction costs an additional ≈ 11 s that is not counted in the solve time of Table 2. Even so, IC is the fastest variant overall, and if several right-hand sides were to be solved with the same matrix the amortized advantage would be far larger, because the factorization is built once and reused.

4.3 Head-to-head performance

Figure 4 places the two matrices side by side, comparing iteration counts (panel a) and solve times (panel b) on logarithmic axes. The picture is consistent across both problems: moving from no preconditioner to Jacobi buys a modest constant-factor improvement, while moving to incomplete Cholesky buys one to two orders of magnitude in iteration count. The translation of iteration savings into time savings is stronger for the small `bcsstk16` (where per-iteration overheads are negligible) than for the large `parabolic_fem` (where the denser IC operator and the factorization cost temper the gain), but in every case the preconditioned solve is at least as fast as, and usually much faster than, the unpreconditioned one.

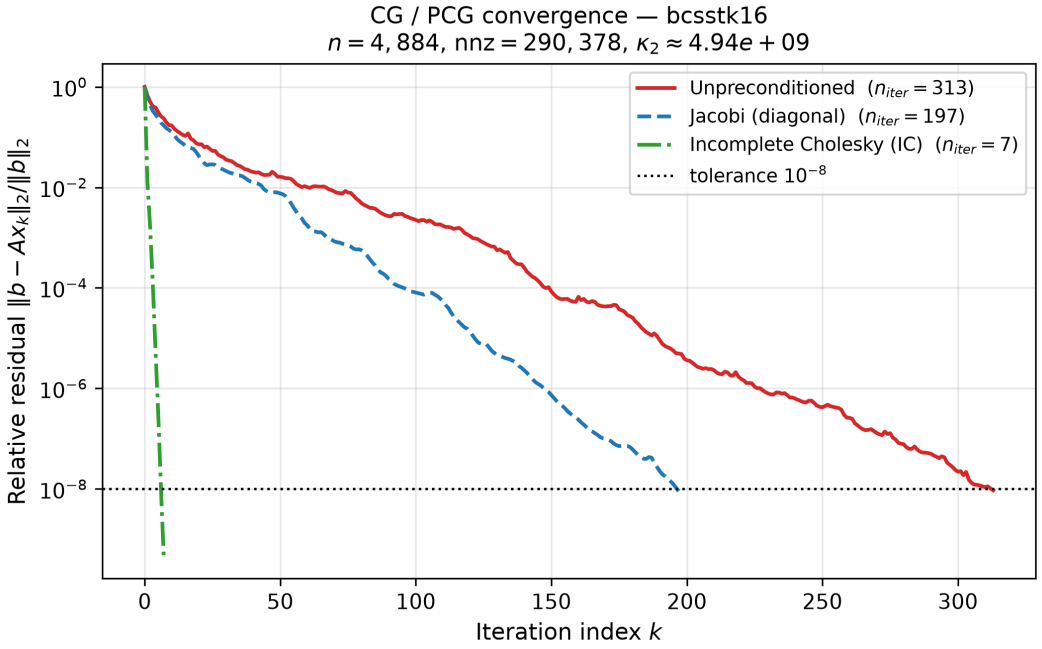

Figure 2: Relative-residual convergence history for `bcsstk16` ($n = 4884.00$, $\kappa_2 \approx 4.94 \times 10^9$). The y-axis is logarithmic; the dotted line marks the $\varepsilon = 0.00 \times 10^{-9}$ tolerance. Incomplete Cholesky reaches the tolerance in 7 iterations, Jacobi in 197, and the unpreconditioned iteration in 313. Note that a small residual on this matrix does not imply a small solution error (Section 5 and Table 4).

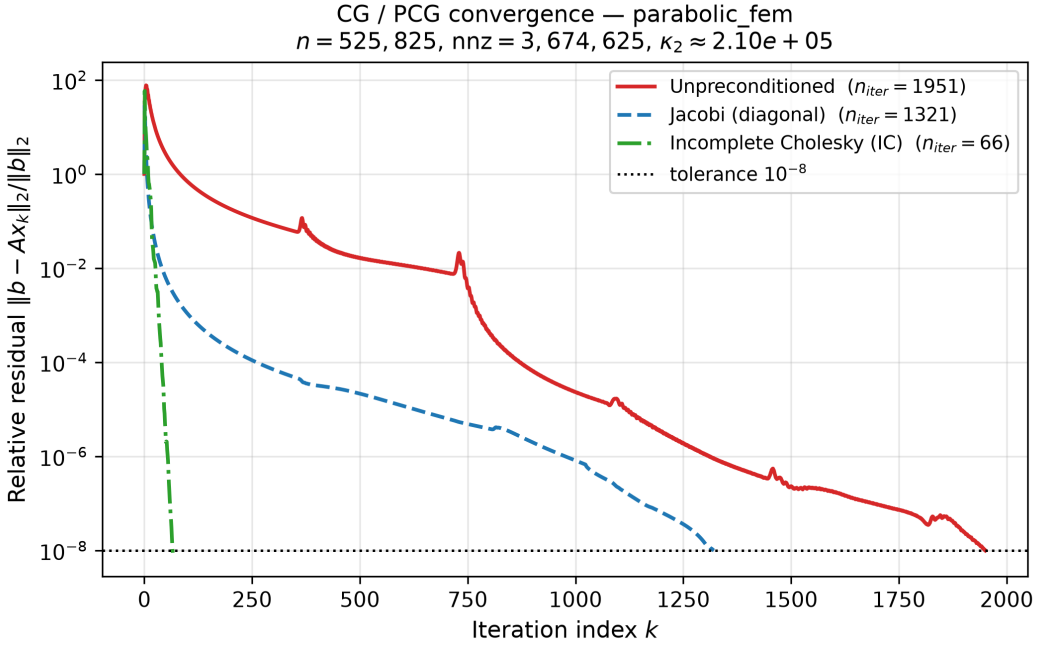

Figure 3: Relative-residual convergence history for `parabolic_fem` ($n = 525{,}825.00$, $\kappa_2 \approx 2.10 \times 10^5$). Incomplete Cholesky reaches the tolerance in 66 iterations, Jacobi in 1321, and the unpreconditioned iteration in 1951. All three variants also achieve accurate solutions (Table 2), in contrast to the ill-conditioned case of Figure 2.

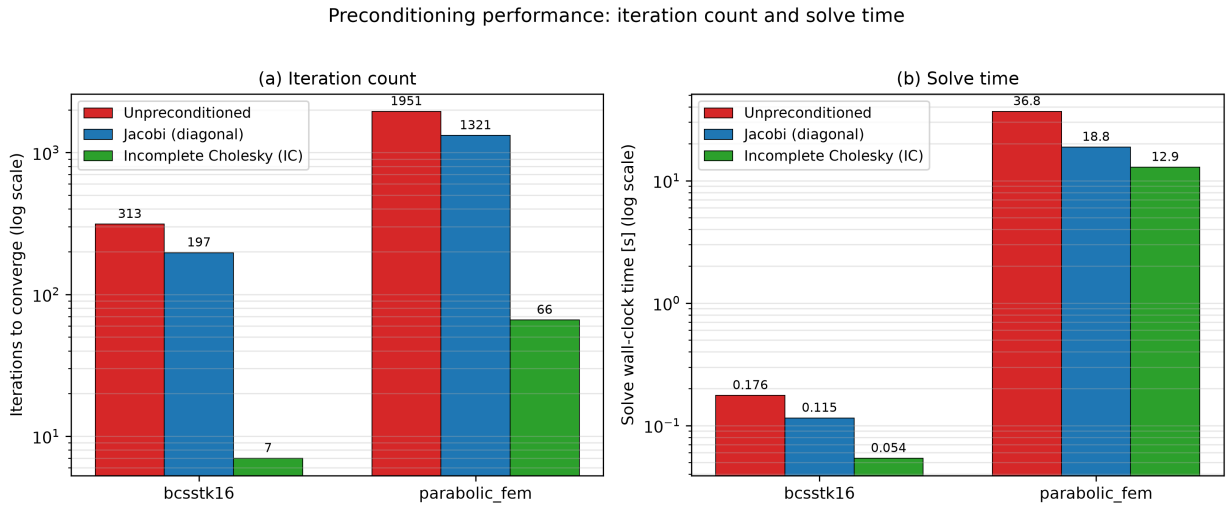

Figure 4: Head-to-head performance of the three solver variants on both matrices: (a) iteration count and (b) solve wall-clock time, both on logarithmic scales. Incomplete Cholesky reduces the iteration count by $\approx 45\times$ on `bcsstk16` and $\approx 30\times$ on `parabolic_fem`, and is the fastest variant on both matrices.

4.4 Per-iteration convergence histories

The complete per-iteration residual histories—pairs $(k,\ \|b - Ax_k\|_2/\|b\|_2)$ for every iteration of every solve—are delivered as machine-readable CSV files, one per matrix–variant combination (`<matrix>_<variant>_history.csv`), and are the data plotted in Figures 2–3. Each file begins at iteration 0 with relative residual 1.0 (the zero initial guess gives $r_0 = b$) and ends at the converged iteration with the final residual recorded in Table 2. Table 3 reproduces short excerpts to make the qualitative differences concrete: the incomplete-Cholesky residual on `bcsstk16` drops by roughly two orders of magnitude per iteration, whereas the unpreconditioned residual improves by only a few percent per step in its early phase.

Table 3: Excerpts from the per-iteration relative-residual histories for `bcsstk16`, illustrating the very different per-step contraction of the three variants. The complete histories for all six solves are provided as CSV files.

	Relative residual $\|b - Ax_k\|_2/\|b\|_2$		
Iter. k	Unpreconditioned	Jacobi	Inc. Cholesky
0	1.00	1.00	1.00
1	6.80×10^{-1}	5.98×10^{-1}	1.30×10^{-2}
2	5.99×10^{-1}	4.72×10^{-1}	1.67×10^{-3}
3	5.79×10^{-1}	4.06×10^{-1}	1.15×10^{-4}
4	5.62×10^{-1}	3.66×10^{-1}	5.52×10^{-6}
5	5.49×10^{-1}	3.35×10^{-1}	3.07×10^{-7}
6	5.30×10^{-1}	3.11×10^{-1}	1.05×10^{-8}
7	5.14×10^{-1}	2.90×10^{-1}	4.79e-10 (converged)
final	9.41e-9 (@313)	8.96e-9 (@197)	4.79e-10 (@7)

5 Discussion

5.1 Preconditioning accelerates convergence, as theory predicts

The dominant, unsurprising message of Table 2 is that preconditioning accelerates CG, and that a spectrally informed preconditioner (incomplete Cholesky) accelerates it far more than a purely diagonal one (Jacobi). This is exactly what the convergence bound (1) leads one to expect: the iteration count is controlled by the condition number of the preconditioned operator, and incomplete Cholesky drives the eigenvalues of $M^{-1}A$ into a tight cluster near unity, effectively replacing $\kappa_2(A)$ by a much smaller number [13, 18]. Jacobi, which only rescales, leaves the bulk of the spectrum spread out and therefore delivers only a constant-factor improvement. It is worth noting that both matrices converge in far fewer iterations than the worst-case bound $\mathcal{O}(\sqrt{\kappa_2(A)})$ would suggest—for `bcsstk16`, $\sqrt{\kappa_2(A)} \approx 7.00 \times 10^4$, yet unpreconditioned CG converges in 313 steps—because CG exploits clustering of the spectrum rather than merely its extremes [8, 27]. The condition number bounds the convergence rate but does not tightly predict it; the detailed spectral distribution matters, which is precisely why incomplete factorizations, by reshaping that distribution, are so effective.

The distinction between iteration-count savings and wall-clock savings deserves emphasis because it is a recurring practical trade-off. On the small `bcsstk16`, where a matrix–vector product is nearly free, the 45× iteration reduction and the 3.3× time reduction are both realized almost in full. On the large `parabolic_fem`, the 30× iteration reduction becomes a 2.9× solve-time reduction because each preconditioned iteration is intrinsically more expensive and because the preconditioner must first be built at a cost (≈ 11 s) comparable to a large fraction of the entire unpreconditioned solve. This setup-versus-solve balance is the central engineering question in preconditioner selection: a more powerful preconditioner is worthwhile only when its per-iteration and construction overheads are repaid by the iterations it saves, and that calculus tips further in favor of strong preconditioners when the same matrix is reused across many right-hand sides [2, 6].

5.2 Residual and error decouple under extreme ill-conditioning

The scientifically most important observation in this study is not that preconditioning is faster—that is expected—but that on the ill-conditioned `bcsstk16` preconditioning changes whether the computed answer is *correct at all*. Because we chose $b = A\mathbf{1}$, we know the exact solution and can tabulate the true error next to the residual; Table 4 does so for `bcsstk16`.

Table 4: Residual versus true error for `bcsstk16` ($\kappa_2 \approx 4.94 \times 10^9$). All three variants satisfy the residual tolerance, but only the preconditioned variants produce an accurate solution. The unpreconditioned solve is a textbook instance of residual–error decoupling: a residual of 9.41×10^{-9} coexists with a solution error of order one.

Variant	Final rel. residual	$\|x_k - \mathbf{1}\|_2$	$\|x_k - \mathbf{1}\|_\infty$
Unpreconditioned	9.41×10^{-9}	8.60	1.00
Jacobi	8.96×10^{-9}	3.73×10^{-6}	5.33×10^{-7}
Inc. Cholesky	4.79×10^{-10}	1.46×10^{-7}	1.94×10^{-8}

The unpreconditioned solve declares convergence with a relative residual of 9.41×10^{-9}, comfortably below the 10^{-8} tolerance, and yet its solution differs from the true answer by $\|x_k - \mathbf{1}\|_\infty = 1.00$: the computed solution is, componentwise, as wrong as the solution is large. This is the inequality (2) in action. With $\kappa_2(A) \approx 4.94 \times 10^9$, a relative residual of $\approx 10^{-8}$ permits a relative error of order

$\kappa_2(A) \times 10^{-8} \approx 50$, so an $\mathcal{O}(1)$ error is entirely consistent with the tiny residual. A small residual, in other words, is *not* a certificate of a small error when the matrix is ill-conditioned—a caution that is well known in the numerical analysis literature [8, 9, 27] but is easy to forget when a solver silently reports "converged."

Preconditioning repairs the decoupling. The Jacobi solve, which only rescales the equations, already reduces the solution error by more than six orders of magnitude to 5.33×10^{-7}, and incomplete Cholesky reduces it to 1.94×10^{-8}—an accuracy commensurate with the requested tolerance. The mechanism is that both preconditioners shrink the *effective* condition number of the system actually presented to CG, so that the bound (2) no longer permits a large error when the residual is small. In effect, preconditioning restores the meaning of the stopping criterion: once the effective conditioning is moderate, "residual below tolerance" again implies "solution accurate to tolerance." This is a second, qualitatively distinct benefit of preconditioning, separate from and arguably more important than the acceleration of convergence: it governs whether the answer can be trusted.

5.3 Contrast with the well-conditioned regime

The half-million-unknown `parabolic_fem` provides the controlled contrast that isolates the phenomenon. Its condition number, $\approx 2.10 \times 10^5$, is more than four orders of magnitude smaller than that of `bcsstk16`. There, all three variants achieve a solution error of $\approx 2.00 \times 10^{-10}$ regardless of iteration count (Table 2): the residual and the error are effectively coupled, so every solver that satisfies the residual tolerance also returns an accurate solution, and preconditioning buys *speed* rather than *correctness*. Placing the two matrices side by side thus cleanly separates the two roles of preconditioning. On a well-conditioned system, preconditioning is a performance optimization: useful, but the unpreconditioned solve is not wrong, only slow. On a pathologically ill-conditioned system, preconditioning is a matter of correctness: without it, a solver can report success while returning a meaningless answer. Practitioners who rely on residual-based stopping criteria—which is to say, nearly everyone—should treat an estimate of the condition number as an integral part of interpreting a "converged" result, and should regard preconditioning not merely as an accelerator but as a safeguard on accuracy [2, 22].

6 Threats to Validity and Limitations

Several caveats bound the scope of these conclusions. First, wall-clock times are single-run measurements on a shared machine and are sensitive to background load, memory bandwidth, and BLAS threading; they should be read as indicative order-of-magnitude comparisons rather than precise benchmarks, and the iteration counts—which are deterministic and machine-independent—are the more robust basis for comparison. Second, the condition numbers are Lanczos *estimates* of the extreme eigenvalues rather than exact values; for the SPD matrices here the shift-invert computation of $\lambda_{\min}$ is well established and the estimates are stable across runs, but they inherit the tolerance of the eigensolver. Third, the incomplete-Cholesky results depend on the drop tolerance, fill-factor bound, and fill-reducing ordering; a different parameterization would shift the balance between factorization cost, per-iteration cost, and iteration count, and in particular a natural ordering is known to perform poorly on the large finite-element matrix (Section 3.3) [2, 24]. Fourth, the study uses a single, structured right-hand side ($b = A\mathbf{1}$) chosen so that the exact solution is known; the residual–error decoupling we document is a property of the matrix conditioning and inequality (2), so it is not an artifact of this choice, but the exact error magnitudes would differ for other right-hand sides. Finally, we restrict attention to CG with two classical preconditioners; a broader study

would include modified and higher-fill incomplete factorizations, algebraic multigrid, and sparse approximate inverses, all of which are natural next steps and are surveyed in the references [2, 24, 26].

7 Conclusion

We have implemented a transparent conjugate gradient iteration and used it to compare unpreconditioned, Jacobi, and incomplete-Cholesky preconditioning on two SPD matrices from the SuiteSparse Matrix Collection spanning very different size and conditioning regimes. Across both matrices, incomplete Cholesky reduces the iteration count by roughly one-and-a-half orders of magnitude relative to the unpreconditioned solve—from 313 to 7 iterations on `bcsstk16` and from 1951 to 66 on `parabolic_fem`—and is the fastest variant in wall-clock terms on both, while Jacobi delivers a consistent but modest constant-factor gain. No solve reached the 20,000.00-iteration cap. Beyond confirming the expected acceleration, the study documents a sharper lesson on the ill-conditioned `bcsstk16`: unpreconditioned CG satisfied the residual tolerance while returning a solution with an $\mathcal{O}(1)$ error, a direct consequence of the condition number entering the bound relating error to residual. Preconditioning restored not only speed but accuracy, collapsing the solution error from order one to $\approx 10^{-8}$. The practical implication is that a residual-based stopping criterion must be read in the light of the matrix's conditioning, and that preconditioning is, for ill-conditioned systems, as much a guarantor of correctness as it is an accelerator of convergence. The custom solver, the per-iteration convergence histories for all six solves, the condition-number estimates, and the figures are provided as reproducible artifacts accompanying this report.

Reproducibility

The complete workflow is scripted. The benchmark driver (`run_cg_benchmark.py`) loads the two matrices and their right-hand sides, estimates the condition numbers, builds the Jacobi and incomplete-Cholesky preconditioners, runs the custom CG iteration of Algorithm 1 in all three variants, and writes a unified JSON summary together with one per-iteration residual-history CSV per solve. The plotting script (`plot_convergence.py`) regenerates all figures from those artifacts. The matrices are the public `bcsstk16` (group HB) and `parabolic_fem` (group Wissgott) from the SuiteSparse Matrix Collection [3, 14], and the computation relies on NumPy [11] and SciPy [29] (whose `spilu` exposes SuperLU [4] and whose `eigsh` exposes ARPACK [16]).

References

[1] Richard Barrett, Michael Berry, Tony F. Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1994. doi: 10.1137/1.9781611971538.

[2] Michele Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182(2):418–477, 2002. doi: 10.1006/jcph.2002.7176.

[3] Timothy A. Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1):1–25, 2011. doi: 10.1145/2049662.2049663.

[4] James W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 20(3):720–755, 1999. doi: 10.1137/S0895479895291765.

[5] Iain S. Duff, Roger G. Grimes, and John G. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, 15(1):1–14, 1989. doi: 10.1145/62038.62043.

[6] Aditi Ghai, Cao Lu, and Xiangmin Jiao. A comparison of preconditioned Krylov subspace methods for large-scale nonsymmetric linear systems. *Numerical Linear Algebra with Applications*, 26(1):e2215, 2019. doi: 10.1002/nla.2215.

[7] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 4th edition, 2013. ISBN 978-1-4214-0794-4.

[8] Anne Greenbaum. *Iterative Methods for Solving Linear Systems*, volume 17 of *Frontiers in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997. doi: 10.1137/1.9781611970937.

[9] Anne Greenbaum and Zdeněk Strakoš. Predicting the behavior of finite precision Lanczos and conjugate gradient computations. *SIAM Journal on Matrix Analysis and Applications*, 13(1): 121–137, 1992. doi: 10.1137/0613011.

[10] Ivar Gustafsson. A class of first order factorization methods. *BIT Numerical Mathematics*, 18 (2):142–156, 1978. doi: 10.1007/BF01931691.

[11] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020. doi: 10.1038/s41586-020-2649-2.

[12] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952. doi: 10.6028/jres.049.044.

[13] David S. Kershaw. The incomplete Cholesky–conjugate gradient method for the iterative solution of systems of linear equations. *Journal of Computational Physics*, 26(1):43–65, 1978. doi: 10.1016/0021-9991(78)90098-0.

[14] Scott P. Kolodziej, Mohsen Aznaveh, Matthew Bullock, Jarrett David, Timothy A. Davis, Matthew Henderson, Yifan Hu, and Read Sandstrom. The SuiteSparse matrix collection website interface. *Journal of Open Source Software*, 4(35):1244, 2019. doi: 10.21105/joss.01244.

[15] Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, 45 (4):255–282, 1950. doi: 10.6028/jres.045.026.

[16] Richard B. Lehoucq, Danny C. Sorensen, and Chao Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. Software, Environments, and Tools. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1998. doi: 10.1137/1.9780898719628.

[17] Thomas A. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Mathematics of Computation*, 34(150):473–497, 1980. doi: 10.1090/S0025-5718-1980-0559197-0.

[18] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of Computation*, 31(137): 148–162, 1977. doi: 10.1090/S0025-5718-1977-0438681-4.

[19] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, New York, NY, 2nd edition, 2006. doi: 10.1007/978-0-387-40065-5.

[20] Yvan Notay. Flexible conjugate gradients. *SIAM Journal on Scientific Computing*, 22(4): 1444–1460, 2000. doi: 10.1137/S1064827599362314.

[21] Christopher C. Paige and Michael A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629, 1975. doi: 10.1137/0712047.

[22] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2nd edition, 2003. doi: 10.1137/1.9780898718003.

[23] Yousef Saad and Henk A. van der Vorst. Iterative solution of linear systems in the 20th century. *Journal of Computational and Applied Mathematics*, 123(1–2):1–33, 2000. doi: 10.1016/S0377-0427(00)00412-X.

[24] Jennifer Scott and Miroslav Tůma. *Algorithms for Sparse Linear Systems*. Nečas Center Series. Birkhäuser Cham, 2023. doi: 10.1007/978-3-031-25820-6.

[25] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-94-125, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA, 1994.

[26] Valeria Simoncini and Daniel B. Szyld. Recent computational developments in Krylov subspace methods for linear systems. *Numerical Linear Algebra with Applications*, 14(1):1–59, 2007. doi: 10.1002/nla.499.

[27] Lloyd N. Trefethen and David Bau III. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997. doi: 10.1137/1.9780898719574.

[28] Henk A. van der Vorst. *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press, Cambridge, UK, 2003. doi: 10.1017/CBO9780511615115.

[29] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, and Paul van Mulbregt. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, 2020. doi: 10.1038/s41592-019-0686-2.