Uppsala University

Information Technology

Assignments for Phd Position

Preconditioned Conjugate Gradient

Author:
Naser Shabani

December 10, 2024

Task: Preconditioned CG

Consider the solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ by the standard conjugate gradient method, where

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

The exact solution is $\hat{\mathbf{x}} = [1, 1, \ldots, 1]^\top$. Starting with $\mathbf{x}_0 = [0, 0, \ldots, 0]^\top$, run the un-preconditioned conjugate gradient method.

Closed-form solution Solution

First, we attempt to solve the problem using an analytic approach. As demonstrated in the proof below, we derive the exact solution to the problem.

Setup of the System

We have a linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$, where $\mathbf{A}$ is the $n \times n$ tridiagonal matrix:

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \cdots & 0 \\ 0 & -1 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

We want to solve for $\mathbf{x} = (x_1, x_2, \ldots, x_n)^\top$.

Equations

The system translates into the following set of equations:

- **For the first row:**

$$2x_1 - x_2 = 1.$$

- **For each intermediate row $i = 2, 3, \ldots, n-1$:**

$$-x_{i-1} + 2x_i - x_{i+1} = 0.$$

- **For the last row:**

$$-x_{n-1} + x_n = 0 \quad \Longrightarrow \quad x_n = x_{n-1}.$$

Deriving a General Formula

From the last equation, we have $x_n = x_{n-1}$. We don't know the values yet, but this is a clue that the solution might be constant or follow a simple pattern.

Next, consider the second-to-last equation (when $i = n-1$):

$$-x_{n-2} + 2x_{n-1} - x_n = 0.$$

Since $x_n = x_{n-1}$, substitute that in:

$$-x_{n-2}+2x_{n-1}-x_{n-1} = 0 \quad \Longrightarrow \quad -x_{n-2}+x_{n-1} = 0 \quad \Longrightarrow \quad x_{n-1} = x_{n-2}.$$

Now we have $x_{n-1} = x_{n-2} = x_n$. By backward induction, applying the same reasoning to each preceding equation:

$$x_{n-2} = x_{n-3}, \quad x_{n-3} = x_{n-4}, \quad \ldots$$

Eventually, this pattern forces:

$$x_2 = x_3 = x_4 = \cdots = x_{n-1} = x_n.$$

Let this common value be c. So far, we have:

$$x_1, \quad x_2 = c, \quad x_3 = c, \quad \ldots, \quad x_n = c.$$

Using the First Equation

From the first equation:

$$2x_1 - x_2 = 1.$$

We know $x_2 = c$, so:

$$2x_1 - c = 1.$$

From the second equation (for $i = 2$):

$$-x_1 + 2x_2 - x_3 = 0.$$

But $x_2 = c$ and $x_3 = c$, giving:

$$-x_1 + 2c - c = 0 \implies -x_1 + c = 0 \implies x_1 = c.$$

Thus $x_1 = c$. Substituting $x_1 = c$ into the first equation:

$$2c - c = 1 \implies c = 1.$$

Conclusion

We found that $c = 1$, and thus $x_1 = 1$, and all other $x_i = 1$. Hence, the unique solution is:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}_{n\times 1}.$$

The Conjugate Gradient Method

The Conjugate Gradient (CG) method is an iterative algorithm for solving systems of linear equations of the form:

$$\mathbf{A}\mathbf{x} = \mathbf{b},$$

where $\mathbf{A}$ is an $n \times n$ symmetric positive-definite (SPD) matrix and $\mathbf{b}$ is a given vector. Such systems commonly arise from the discretization of partial differential equations and other problems in scientific and engineering computations.

Key Properties and Requirements

Symmetry: The matrix $\mathbf{A}$ must be symmetric, i.e., $\mathbf{A} = \mathbf{A}^\top$. This symmetry ensures that the inner product-like structure

$$\langle \mathbf{u}, \mathbf{v} \rangle_{\mathbf{A}} = \mathbf{u}^\top \mathbf{A} \mathbf{v}$$

is well-defined.

Positive-Definiteness: The matrix $\mathbf{A}$ must be positive-definite. In other words, for any nonzero vector $\mathbf{z}$, we have:

$$\mathbf{z}^\top \mathbf{A}\mathbf{z} > 0.$$

Positive-definiteness guarantees that the search directions generated by the algorithm are meaningful and that the solution is unique.

Underlying Idea

The CG method can be viewed as a procedure that iteratively refines the solution by moving through a sequence of carefully chosen search directions. Unlike simpler iterative methods that rely on arbitrary directions, CG ensures that each new direction is $\mathbf{A}$-conjugate to all previous directions. This conjugacy property prevents the algorithm from reusing the same information multiple times, thereby improving efficiency.

In exact arithmetic, CG will find the exact solution in at most n steps, where n is the size of the matrix. However, due to floating-point rounding errors, it often converges to a sufficiently accurate approximate solution in far fewer iterations, making it highly effective for large sparse problems.

Algorithmic Steps

Initialization:

- Choose an initial guess $\mathbf{x}_0$ (often $\mathbf{x}_0 = 0$).
- Compute the initial residual:

  $$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0.$$

  If $\mathbf{x}_0 = 0$, then $\mathbf{r}_0 = \mathbf{b}$.
- Set the initial search direction:

  $$\mathbf{p}_0 = \mathbf{r}_0.$$

Iterative Process (for $k = 0, 1, 2, \ldots$):

1. **Step Length Calculation:** Compute the step length α_k along the current search direction:
$$\alpha_k = \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{p}_k^\top \mathbf{A}\mathbf{p}_k}.$$
2. **Solution Update:** Update the approximate solution:
$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k.$$
3. **Residual Update:** Update the residual:
$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k.$$
If $\|\mathbf{r}_{k+1}\|$ is sufficiently small (below a given tolerance), terminate.
4. **Conjugacy Update:** Compute β_k to ensure the new direction is **A**-conjugate to all previous ones:
$$\beta_k = \frac{\mathbf{r}_{k+1}^\top \mathbf{r}_{k+1}}{\mathbf{r}_k^\top \mathbf{r}_k}.$$
5. **Direction Update:** Update the direction vector:
$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k.$$

Termination: The algorithm terminates when the norm of the residual $\|\mathbf{r}_{k+1}\|$ is less than a prescribed tolerance, indicating that $\mathbf{x}_{k+1}$ is a sufficiently accurate approximation to the true solution.

Practical Considerations

Preconditioning: While CG converges in at most n steps in theory, real-world problems with large and ill-conditioned matrices may slow down convergence. Preconditioning modifies the system into a more favorable form, reducing the number of iterations required.

Numerical Stability: Floating-point arithmetic introduces rounding errors, so exact convergence in n steps is rarely observed. Nonetheless, CG typically provides a good approximation well before reaching n iterations.

Experiment Setup and Results

In this experiment, we examined the influence of the matrix size n on the convergence behavior of the Conjugate Gradient (CG) method. The system $\mathbf{Ax} = \mathbf{b}$ (already defined previously) was solved using CG for various dimensions n (e.g., $10, 50, 100, 200, 500, 1000$) under the following conditions:

- **Initial Guess:** $\mathbf{x}_0 = 0$.
- **Tolerance:** $\|\mathbf{r}_k\| \leq 10^{-6}$.
- **Maximum Number of Iterations:** $\text{max\_iter} = n$.

For each chosen size n, we found that the CG method required exactly n iterations to achieve the specified tolerance. This direct correlation between the problem dimension and the iteration count indicates that as the matrix size grows, the difficulty of the problem (reflected in its conditioning) directly impacts the convergence rate.

When n is small, reaching the stopping criterion is straightforward, but as n increases, the complexity of the spectrum of $\mathbf{A}$ effectively forces CG to utilize the full n iterations allowed to approximate the solution sufficiently.

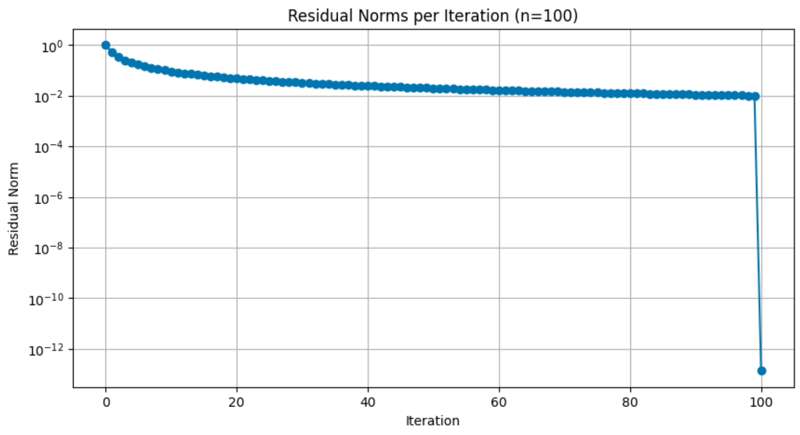

Figure 1: Residual Norms per Iteration

Deriving the Exact Form of the CG Iterates

The given formula states that after k iterations, the approximate solution vector $\mathbf{x}^{(k)}$ has a specific pattern:

$$\mathbf{x}^{(k)} = \begin{bmatrix} \frac{k}{k+1} & \frac{k-1}{k+1} & \cdots & \frac{1}{k+1} & 0 & \cdots & 0 \end{bmatrix}^\top,$$

for $1 \leq k \leq n-1$.

In other words, the first k entries of $\mathbf{x}^{(k)}$ form a linearly decreasing sequence from $\frac{k}{k+1}$ down to $\frac{1}{k+1}$, while all subsequent entries remain zero. Mathematically, the entries of $\mathbf{x}^{(k)}$ can be expressed as:

$$x_i^{(k)} = \begin{cases} \frac{k+1-i}{k+1}, & \text{for } i = 1, 2, \ldots, k, \\ 0, & \text{for } i > k. \end{cases}$$

This pattern continues for $k = 1, 2, \ldots, n-1$, with the last entry updated only on the nth iteration. With each iteration, the "front" of the solution vector moves one step further to the right. Initially, only the first component is corrected; in the next iteration, the first two components are refined; by the k-th iteration, the first k components have taken on a specific pattern, while the remaining entries remain at zero.

The reason behind this structured propagation is that the CG method, on this particular tridiagonal system, extracts information about the solution one component at a time, moving systematically down the line. Each new search direction introduced at iteration k effectively "reaches" one step further along the vector, allowing the solution approximation to incorporate and adjust another component.

The CG method constructs approximations $\mathbf{x}^{(k)}$ within a sequence of nested Krylov subspaces generated by $\mathbf{A}$ and the initial residual $\mathbf{r}^{(0)}$. Mathematically, the k-th Krylov subspace is defined as:

$$\mathcal{K}_k(\mathbf{A}, \mathbf{r}^{(0)}) = \operatorname{span}\{\mathbf{r}^{(0)}, \mathbf{A}\mathbf{r}^{(0)}, \mathbf{A}^2\mathbf{r}^{(0)}, \ldots, \mathbf{A}^{k-1}\mathbf{r}^{(0)}\}.$$

Since $\mathbf{r}^{(0)} = \mathbf{b}$ is nonzero only in the first component, the Krylov subspaces reflect the action of $\mathbf{A}$ repeatedly on a vector that starts concentrated at the first component. As a result, the influence of $\mathbf{A}$ on subsequent components appears incrementally, yielding the observed "one-step-at-a-time" update pattern.

This structure explains why the solution vector $\mathbf{x}^{(k)}$ updates progressively, with each iteration refining one additional component while the rest remain at zero until their turn is reached.

The formula for $\mathbf{x}^{(k)}$ gradually approaches the exact solution as k increases. By the time $k = n$, the entire vector is updated:

$$\mathbf{x}^{(n)} = \hat{\mathbf{x}} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}.$$

In exact arithmetic, the CG method guarantees that for an n-dimensional symmetric positive-definite (SPD) system, it will find the exact solution in at most n steps. The derived formula makes this theoretical property concrete by showing exactly how each iteration contributes to the final solution.

This property demonstrates the efficiency of the CG method, especially for large-scale problems, as it systematically incorporates one component of the solution at each step, eventually reaching the unique solution after n iterations.

Pen and Paper Experiment

Initial Conditions

- **Initial Guess:** $\mathbf{x}^{(0)} = \mathbf{0}$.
- **Initial Residual:** $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)} = \mathbf{b}$.
- **Initial Search Direction:** $\mathbf{p}^{(0)} = \mathbf{r}^{(0)} = \mathbf{b}$.

First Iteration ($k = 1$)

Step Length α_0:

$$\alpha_0 = \frac{(\mathbf{r}^{(0)}, \mathbf{r}^{(0)})}{(\mathbf{p}^{(0)}, \mathbf{A}\mathbf{p}^{(0)})}.$$

- Numerator: $(\mathbf{r}^{(0)}, \mathbf{r}^{(0)}) = (\mathbf{b}, \mathbf{b}) = 1$.

- Denominator: $(\mathbf{p}^{(0)}, \mathbf{A}\mathbf{p}^{(0)}) = (\mathbf{b}, \mathbf{A}\mathbf{b})$.

  Since $\mathbf{b}$ has only the first element non-zero:

$$\mathbf{A}\mathbf{b} = \mathbf{A}\mathbf{e}_1 = \text{first column of } \mathbf{A} = \begin{bmatrix} 2 \\ -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

  Thus:

$$(\mathbf{p}^{(0)}, \mathbf{A}\mathbf{p}^{(0)}) = (\mathbf{b}, \mathbf{A}\mathbf{b}) = 1 \cdot 2 = 2.$$

Therefore:

$$\alpha_0 = \frac{1}{2}.$$

Update $\mathbf{x}^{(1)}$ and $\mathbf{r}^{(1)}$:

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha_0 \mathbf{p}^{(0)} = \mathbf{0} + \frac{1}{2}\mathbf{b} = \begin{bmatrix} \frac{1}{2} \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

$$\mathbf{r}^{(1)} = \mathbf{r}^{(0)} - \alpha_0 \mathbf{A}\mathbf{p}^{(0)} = \mathbf{b} - \frac{1}{2}\begin{bmatrix} 2 \\ -1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{2} \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Second Iteration ($k = 2$)

Conjugacy Factor β_0:

$$\beta_0 = \frac{(\mathbf{r}^{(1)}, \mathbf{r}^{(1)})}{(\mathbf{r}^{(0)}, \mathbf{r}^{(0)})} = \frac{\frac{1}{2}^2}{1} = \frac{1}{4}.$$

Update Search Direction $\mathbf{p}^{(1)}$:

$$\mathbf{p}^{(1)} = \mathbf{r}^{(1)} + \beta_0 \mathbf{p}^{(0)} = \begin{bmatrix} 0 \\ \frac{1}{2} \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \frac{1}{4} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} \\ \frac{1}{2} \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Update $\mathbf{x}^{(2)}$ and $\mathbf{r}^{(2)}$: Skipping intermediate steps for brevity, the updated forms reveal the general pattern.

Illustrative Example ($n = 4$)

- Iteration 1: $\mathbf{x}^{(1)} = \left[\frac{1}{2}, 0, 0, 0\right]^\top$
- Iteration 2: $\mathbf{x}^{(2)} = \left[\frac{2}{3}, \frac{1}{3}, 0, 0\right]^\top$
- Iteration 3: $\mathbf{x}^{(3)} = \left[\frac{3}{4}, \frac{2}{4}, \frac{1}{4}, 0\right]^\top$
- Iteration 4: $\mathbf{x}^{(4)} = \hat{\mathbf{x}}$ (all components updated).

Analysis of Slow Convergence in the CG Method

In this particular problem, the Conjugate Gradient (CG) method appears to converge slowly because it essentially follows the "hardest possible path" to the solution, given the structure of the matrix $\mathbf{A}$ and the right-hand side vector $\mathbf{b}$. Several interrelated factors contribute to this behavior:

1. Ill-Conditioning and Eigenvalue Distribution: The convergence rate of CG is closely tied to the condition number of the matrix $\mathbf{A}$, which is the ratio $\kappa(\mathbf{A}) = \frac{\lambda_{\max}}{\lambda_{\min}}$ of its largest to smallest eigenvalue. For this tridiagonal system, as the dimension n grows, the smallest eigenvalue $\lambda_{\min}$ approaches zero, making the matrix increasingly ill-conditioned. This large condition number slows the rate at which CG can reduce the error, forcing it to take many iterations—ultimately n in exact arithmetic—to arrive at the exact solution.

2. Sequential "One-Component-at-a-Time" Propagation: The derived formula for the iterates shows that, at each iteration k, only the first k components of the approximate solution vector $\mathbf{x}^{(k)}$ are corrected away from zero, while the remaining entries remain unchanged. In effect, the "information" about the solution propagates from the first component of $\mathbf{x}$ to the last component in a chain-like fashion. Only after n steps does the influence reach the final component, thus completing the solution vector. This inherently limits any faster convergence that might occur if all components could be improved simultaneously.

3. Structure of the Krylov Subspace: CG works by building iterates in nested Krylov subspaces:

$$\mathcal{K}_k(\mathbf{A}, \mathbf{r}^{(0)}) = \operatorname{span}\{\mathbf{r}^{(0)}, \mathbf{A}\mathbf{r}^{(0)}, \mathbf{A}^2\mathbf{r}^{(0)}, \ldots\}.$$

Since the initial residual $\mathbf{r}^{(0)} = \mathbf{b}$ is nonzero only in its first entry, the vectors in this sequence predominantly capture the influence of $\mathbf{A}$ applied repeatedly to a vector concentrated at the first node. The subspace grows incrementally, and each new direction "reaches" one step further along the vector. This inherently incremental process matches the structure of the problem, causing CG to exhaustively explore all n directions before obtaining the exact solution.

These factors—ill-conditioning, sequential propagation, and the structure of the Krylov subspace—combine to make the CG method converge slowly for this particular problem. However, this behavior is consistent with the theoretical guarantees of CG, which ensure convergence in at most n steps for an n-dimensional SPD system.

Effect of Initial Guess on Convergence

Choosing a more informed initial guess can lead to faster convergence. The Conjugate Gradient method constructs its search directions based on the initial residual $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$. If the initial guess $\mathbf{x}^{(0)}$ is far from the true solution, then the initial residual will be large, and CG must expend more iterations "pushing" the solution through the chain-like structure of the problem to reduce that residual.

Here are some key points on how the initial guess affects convergence:

Smaller Initial Residual: If the initial guess $\mathbf{x}^{(0)}$ is closer to the true solution $\mathbf{x}$, then $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$ will be smaller. A smaller residual typically allows CG to reach a given tolerance in fewer steps because less "work" is needed to reduce the error.

Influence on Krylov Subspace: The iterates of CG lie in a sequence of nested Krylov subspaces generated from the initial residual. A better initial guess changes the initial residual, potentially giving CG a more direct "route" to the solution. In other words, the subspace that CG explores may allow faster progress toward high-importance directions in the solution space.

Immediate Convergence If Perfect: If you somehow pick the exact solution as the initial guess, CG would converge immediately without any iterations, demonstrating that the method's performance is highly sensitive to the choice of initial guess.

Practical Implication: In theory, the Conjugate Gradient (CG) method guarantees convergence to the exact solution in at most n iterations for an n-dimensional symmetric positive definite (SPD) system. However, in practice, a well-chosen initial guess can significantly reduce the number of iterations required. Leveraging prior knowledge of the problem, such as using a solution from a similar, previously solved problem, or starting from a known approximate solution (a warm start), can greatly enhance the convergence speed of the CG method. In our case, the results remain unchanged when using random initial starting points, as CG still converges to the correct solution. However, if the exact solution is provided as the initial guess, CG solves the equation in a single iteration, demonstrating the method's sensitivity to the quality of the starting point.

Preconditioned Conjugate Gradient (PCG) Method

The Conjugate Gradient (CG) method is a widely used approach for solving large, sparse, symmetric positive-definite linear systems of the form $\mathbf{A}\mathbf{x} = \mathbf{b}$. However, its performance is significantly influenced by the condition number $\kappa(\mathbf{A})$. As the system dimension increases or when the problem is ill-conditioned, CG often requires a large number of iterations to achieve the desired accuracy. To address this issue, preconditioning techniques are employed to transform the original problem into an equivalent form with a better-conditioned matrix, thereby improving the convergence rate.

In this work, we implemented two preconditioning methods: Incomplete Cholesky and Simple Jacobi. The experiments were initially performed on the matrices provided in the assignment and subsequently extended to matrices from the SuiteSparse Matrix Collection to assess the effectiveness of our implementations. Detailed explanations of the two preconditioners and their respective impacts on the convergence of the CG method are included in the report.

Key Idea of Preconditioning

A preconditioner $\mathbf{M}$ is a matrix that approximates $\mathbf{A}^{-1}$ in some sense, making it easier to solve the transformed system. Instead of directly applying CG to $\mathbf{Ax} = \mathbf{b}$, we consider:

$$\mathbf{M}^{-1}\mathbf{Ax} = \mathbf{M}^{-1}\mathbf{b}.$$

If $\mathbf{M}$ is chosen such that $\mathbf{M}^{-1}\mathbf{A}$ (or $\mathbf{M}^{-1/2}\mathbf{A}\mathbf{M}^{-1/2}$ if a symmetric factorization is used) has a significantly lower condition number than $\mathbf{A}$, the CG iterations on this preconditioned system will converge faster.

What Makes a Good Preconditioner?

- **Effectiveness:** It should significantly reduce the condition number. Ideally, $\mathbf{M} \approx \mathbf{A}$ or $\mathbf{M} \approx \mathbf{A}^{-1}$, so that $\mathbf{M}^{-1}\mathbf{A}$ becomes closer to the identity matrix.
- **Efficiency:** Computing $\mathbf{M}$ and applying $\mathbf{M}^{-1}$ to a vector should be much cheaper than solving $\mathbf{Ax} = \mathbf{b}$ directly. A good preconditioner strikes a balance between being "close" to $\mathbf{A}$ and being computationally inexpensive to apply.

Common choices include incomplete factorizations (like Incomplete Cholesky), diagonal scaling, block diagonal approximations, or more sophisticated methods tailored to the problem's structure.

Preconditioned CG Algorithm Outline

The PCG algorithm modifies the standard CG steps by including the preconditioner $\mathbf{M}$. Let's denote:

$$\mathbf{y}_k = \mathbf{M}^{-1}\mathbf{r}_k,$$

where $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$ is the residual at iteration k. The preconditioner transforms the residual into a new vector $\mathbf{y}_k$ that acts as the "improved" search direction factor. The PCG iterations proceed as follows:

Initialization:

- Choose an initial guess $\mathbf{x}_0$.
- Compute the initial residual $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$.
- Apply the preconditioner: $\mathbf{y}_0 = \mathbf{M}^{-1}\mathbf{r}_0$.
- Set the initial search direction $\mathbf{p}_0 = \mathbf{y}_0$.

Iteration ($k = 0, 1, 2, \ldots$):

1. Compute $\mathbf{z} = \mathbf{A}\mathbf{p}_k$.
2. Compute the step length:
$$\nu_k = \frac{\mathbf{y}_k^\top \mathbf{r}_k}{\mathbf{p}_k^\top \mathbf{z}}.$$
3. Update the solution:
$$\mathbf{x}_{k+1} = \mathbf{x}_k + \nu_k \mathbf{p}_k.$$
4. Update the residual:
$$\mathbf{r}_{k+1} = \mathbf{r}_k - \nu_k \mathbf{z}.$$
5. Apply the preconditioner again:
$$\mathbf{y}_{k+1} = \mathbf{M}^{-1}\mathbf{r}_{k+1}.$$
6. Compute:
$$\mu_k = \frac{\mathbf{y}_{k+1}^\top \mathbf{r}_{k+1}}{\mathbf{y}_k^\top \mathbf{r}_k}.$$
7. Update the search direction:
$$\mathbf{p}_{k+1} = \mathbf{y}_{k+1} + \mu_k \mathbf{p}_k.$$

Termination: Repeat the steps until $\|\mathbf{r}_k\|$ (or another convergence criterion) is sufficiently small.

Effects of Preconditioning

- With a well-chosen preconditioner, the residual norm typically decreases much faster than in standard CG.
- The number of iterations required to achieve a given tolerance is often dramatically reduced.
- Even though each iteration might be slightly more expensive (due to applying $\mathbf{M}^{-1}$), the overall computational cost is generally lowered because the total number of iterations drops significantly.

Jacobi Preconditioner

The Jacobi preconditioner is one of the simplest preconditioning methods, leveraging only the diagonal elements of the matrix $\mathbf{A}$. It scales the system to reduce the disparity in magnitudes among equations, improving convergence.

Definition and Construction

The Jacobi preconditioner is defined as:

$$\mathbf{M} = \operatorname{diag}(\mathbf{A}),$$

where $\mathbf{M}$ is a diagonal matrix containing the diagonal entries of $\mathbf{A}$. Its inverse is trivial to compute:

$$\mathbf{M}^{-1} = \operatorname{diag}\left(\frac{1}{a_{11}}, \frac{1}{a_{22}}, \dots, \frac{1}{a_{nn}}\right).$$

When applied, the preconditioned system becomes:

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}.$$

Key Properties

Advantages:

- Easy to implement and computationally inexpensive.
- Requires minimal memory as only the diagonal entries are stored.

Disadvantages:

- Provides limited improvement in conditioning, especially for highly ill-conditioned systems or matrices with significant off-diagonal contributions.

The Jacobi preconditioner is most suitable for well-conditioned matrices where modest improvements suffice or as a baseline comparison for more advanced preconditioners.

Incomplete Cholesky Factorization

The Incomplete Cholesky (IC) factorization is a method for creating a preconditioner from a sparse symmetric positive-definite matrix $\mathbf{A}$. The idea is rooted in the standard Cholesky decomposition, where a symmetric positive-definite matrix $\mathbf{A}$ can be factored exactly into:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^\top,$$

with $\mathbf{L}$ a lower-triangular matrix whose diagonal entries are positive.

However, when $\mathbf{A}$ is large and sparse (i.e., most entries are zero), computing and storing the full Cholesky factorization can be expensive. The exact factorization often introduces many new nonzero elements (known as fill-in) in $\mathbf{L}$. To address this, the Incomplete Cholesky factorization relaxes the requirement of a full factorization and allows the factorization to be "incomplete." This means certain nonzero fill-in entries that would occur in an exact factorization are intentionally dropped (set to zero) according to a prescribed pattern or tolerance. By doing so, the resulting $\mathbf{L}$ is sparser than the full Cholesky factor and thus cheaper to compute and apply.

Key Properties

- **Sparsity Control:** By dropping small or strategically chosen fill-in elements, we keep $\mathbf{L}$ and thus the preconditioner sparse.

- **Approximation Quality:** While incomplete factorization is not exact, it still approximates $\mathbf{A}$ reasonably well. As a result, using $\mathbf{M} = \mathbf{L}\mathbf{L}^\top$ as a preconditioner can significantly improve the conditioning of the system when applied inside iterative methods like CG.
- **Adjustability:** One can control the level of fill permitted in the incomplete factorization (for example, by allowing certain extra diagonals or by using a numeric dropping threshold) to balance between the cost of forming/applying the preconditioner and the rate of convergence it provides.

Incomplete LU Factorization

The Incomplete LU (ILU) factorization is a generalization of the idea behind Incomplete Cholesky factorization to non-symmetric (or non-positive-definite) matrices. A standard LU factorization decomposes a general square matrix $\mathbf{A}$ into:

$$\mathbf{A} = \mathbf{L}\mathbf{U},$$

where $\mathbf{L}$ is a lower-triangular matrix with ones on the diagonal, and $\mathbf{U}$ is an upper-triangular matrix. However, for large sparse matrices, computing the exact LU factorization can again lead to significant fill-in and computational expense.

The Incomplete LU factorization, or ILU, aims to create an approximation:

$$\mathbf{A} \approx \tilde{\mathbf{L}}\tilde{\mathbf{U}},$$

by dropping certain entries that would appear in $\mathbf{L}$ and $\mathbf{U}$ during factorization. This results in a sparser $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{U}}$, which can serve as a good preconditioner for iterative solvers like GMRES, BiCGSTAB, or even CG (when $\mathbf{A}$ is symmetric but one still uses ILU for historical or simplicity reasons).

Key Properties

- **Applicability to General Matrices:** Unlike the Incomplete Cholesky factorization, ILU does not require symmetry or positive-definiteness. It's more widely applicable to a broader class of problems.
- **Sparsity Patterns:** Similar to IC, ILU factorization can be guided by a chosen sparsity pattern or dropping strategy. Popular variants

include ILU(0) (no fill-in beyond the original pattern of $\mathbf{A}$) and ILU(k) (allowing fill-in up to k diagonals away from the main diagonal).

- **Preconditioning Quality:** A good ILU preconditioner can drastically reduce the iteration count needed by Krylov subspace methods for solving systems with nonsymmetric or indefinite matrices. More fill-in (i.e., allowing more nonzero entries in $\mathbf{L}$ and $\mathbf{U}$) tends to produce a better approximation and faster convergence but increases the factorization and application cost.