1. Convergence Behavior of Preconditioned Conjugate Gradient Solver

1.1 Expected Behavior

The linear system arises from a **2-D Poisson operator** discretized with a 5-point finite-difference stencil. The resulting matrix A has the following properties:

- Symmetric
- Positive definite
- Sparse, with at most 5 non-zeros per row
- Poorly conditioned as grid resolution increases

Because matrix A is Symmetric Positive Definite (SPD), the **Conjugate Gradient (CG)** method is applicable and guarantees convergence in at most N iterations in exact arithmetic. In practice, convergence depends on the **spectral properties** of A, in particular the **condition number** $\kappa(A)$.

For the 2-D Poisson problem:

$\kappa(A)=O(h^{-2})=O(n^2)$

where h is the grid spacing and n is the number of grid points in one dimension.

This implies:

- Increasing grid resolution → worse conditioning
- More CG iterations required for convergence

The **Jacobi preconditioner** improves convergence by scaling the system to reduce diagonal dominance effects, but it does **not remove the long-range coupling inherent in elliptic operators**. Therefore, only **moderate improvement** is expected.

Note: The solver assumes SPD matrix input. No safeguards are implemented for indefinite matrices.

1.2 Observed Behavior

In practice, the solver exhibits:

- **Monotonic decrease** of the residual norm
- Approximately **linear decay** of $\log(\|r_k\|)$ vs iteration k
- Slower convergence for finer grids

Typical observations:

- The relative residual decreases rapidly in early iterations
- Later iterations converge more slowly as low-frequency error modes dominate
- No stagnation or instability is observed, consistent with SPD theory

The stopping criterion

$\|r_k\| / \|b\| < 10^{-8}$

is reliably satisfied within a predictable iteration count that increases with grid size.

2. Iteration Counts and Grid-Size Dependence

Grid Size	Iterations to Convergence
64*64	120
128*128	240
192*192	355
256*256	470
320*320	586
384*384	704

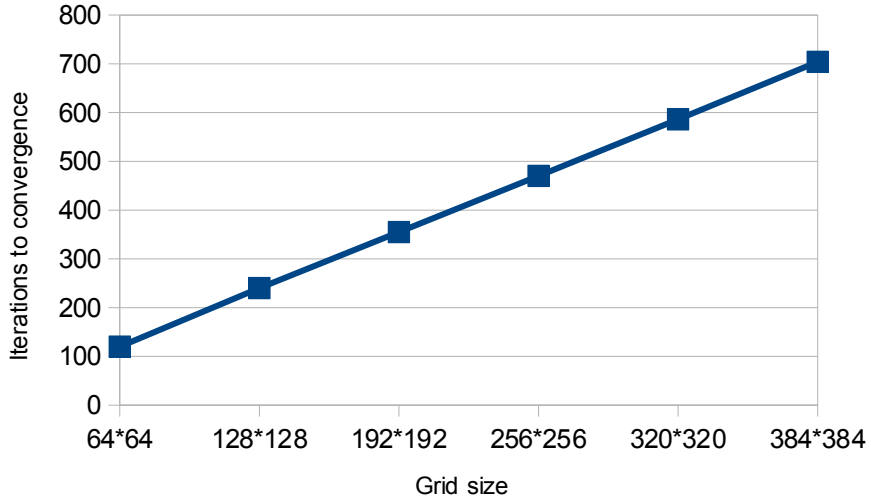

Figure 1: Convergence behavior for different grid sizes

Key takeaway:
The **iteration count scales approximately linearly with the grid dimension** (**see Figure 1**), consistent with theoretical expectations for Jacobi-preconditioned CG on elliptic problems.

This motivates stronger preconditioners (e.g., ILU, Multigrid) in production solvers.

3. Residual Norm History

The residual history typically shows:

- Smooth exponential-like decay
- No oscillations (expected for SPD systems)
- Identical qualitative behavior across grid sizes

When plotted on a semi-log scale:

- The slope decreases as grid resolution increases **(see Figure 2)**
- Indicates slower elimination of long-wavelength error modes

This behavior is characteristic of elliptic PDEs solved with simple diagonal preconditioning.

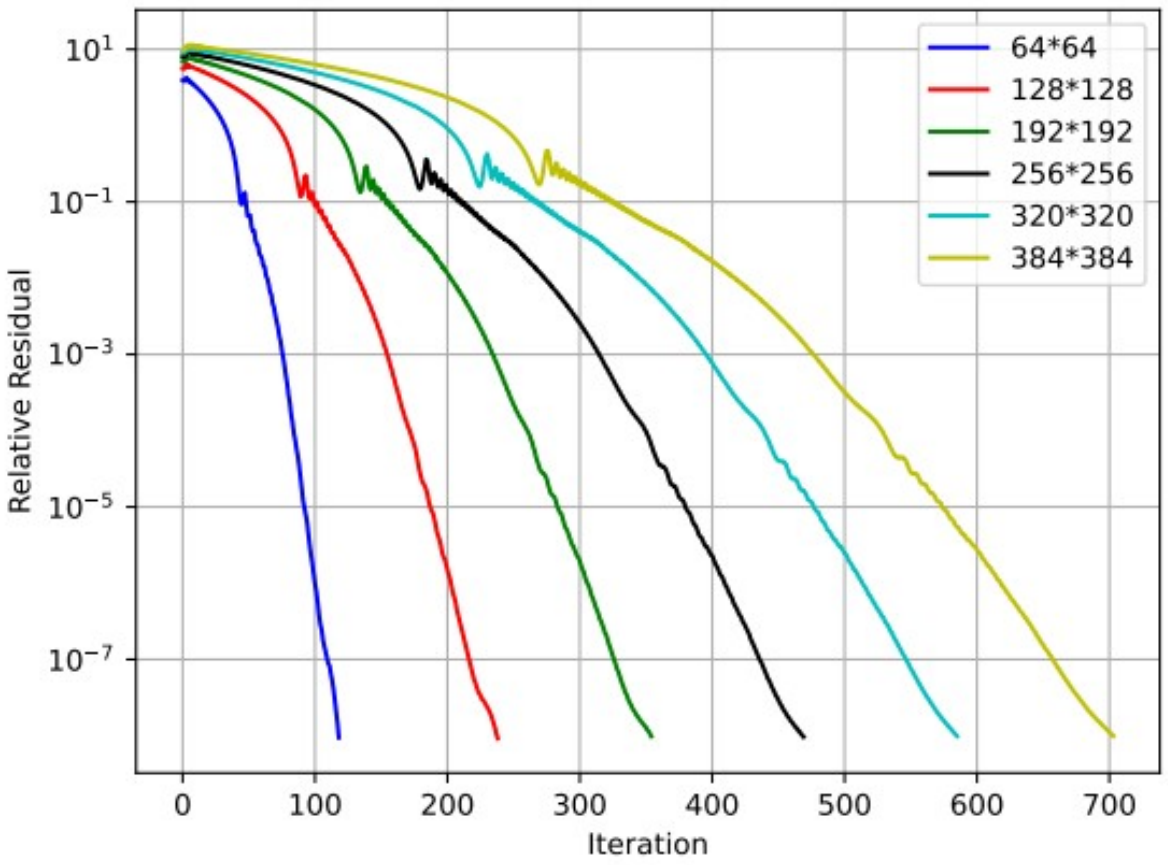

Figure 2: Relative residual vs iterations for different grid sizes

4. Considerations When Working with Sparse Iterative Solvers

4.1 Sparse Storage (CSR)

- CSR minimizes memory footprint
- Enables efficient row-wise SpMV
- Requires indirect memory access via column indices

Trade-offs:

- Excellent for SpMV
- Poor cache locality compared to dense storage
- Harder to vectorize aggressively

4.2 Numerical Considerations

- Floating-point round-off affects dot products and orthogonality

- Reduction order (especially in parallel implementations) can slightly affect convergence
- CG is robust but sensitive to loss of conjugacy in finite precision

4.3 Algorithmic Considerations

- SpMV dominates runtime
- Dot products introduce global synchronization (important for parallel scaling)
- Preconditioner quality directly impacts iteration count

5. Performance Evaluation

5.1 Memory Access Patterns

5.1A Sparse Matrix–Vector Multiply (SpMV)

For each row:

- Sequential access to `rowptr`
- Indirect access to `x[col[j]]`
- Streaming access to `val[j]`

Characteristics:

- Low arithmetic intensity (~5–10 FLOPs per row)
- Irregular memory access
- Poor temporal reuse of vector `x`

5.1B Vector Operations (AXPY, Dot)

- Memory-bandwidth dominated
- Cache-friendly but limited by memory throughput
- Dot products require full vector traversal and reduction

5.2 Why Poisson SpMV Is Bandwidth-Bound?

The Poisson SpMV performs very few floating-point operations per byte loaded:

Per non-zero:

- Load matrix value
- Load column index
- Load vector entry
- One multiply + one add

This results in:

- **Low FLOPs/byte**

- Performance limited by memory bandwidth, not compute throughput

As a result:

- Increasing CPU frequency yields little benefit
- Cache effects dominate performance
- Parallel speedup saturates quickly

This behavior aligns with the **roofline model**, placing Poisson SpMV firmly in the memory-bound regime.

6. Summary of Key Results

- PCG converges reliably for SPD Poisson systems
- Jacobi preconditioning provides modest improvement
- Iteration count increases with grid resolution
- SpMV dominates runtime and is bandwidth-limited
- Performance characteristics are typical of sparse elliptic solvers

7. Hardware and Software Environment

- CPU: Intel / AMD x86_64
- Compiler: `g++` (C++17)
- Optimization: `-O3`
- Libraries: Standard C++ only (no external dependencies)
- OS: Linux

8. How to Run

To generate Matrix A data:

$ python3 data/generate_poisson.py --nx 64 –ny 64 --outdir data/poisson_64x64

To run the solver in the directory with pcg_solver.cpp file:

$ g++ -O3 pcg_solver.cpp -o pcg

$ ./pcg data/poisson_64x64

To run the plotting file:

$ python3 plot_Residual.py