Fast Iterative Solvers. Project 1.

Summer Semester 2025

Report submitted by

Sebastian Bennet Linden

409559

Aachen, June 16, 2025

Rheinisch-Westfälische Technische Hochschule Aachen
Simulation Sciences M.Sc.

1 Introduction

This report documents two solving algorithms for linear systems in the form of $Ax = b$. It is generally assumed, that A is to large to be inverted directly, so instead iterative methods are applied to approximate x such that the residual $r = b - Ax$ is minimized. The linear systems, that arise from scientific applications like the finite element method are often very large and sparse. This sparsity allows for very efficient solvers like the Generalized minimal residual method (GMRES) and the Conjugate Gradient Method (CG). This report does not explain how these algorithms work, but instead focuses on their efficiency when approximating the solution x. In the case of the GMRES algorithm, three different preconditioners are applied and compared to the default case. These are:

- Jacobi
- Gauss-Seidel
- ILU(0)

Although the matrices change, for each system $Ax = b$ the true solution is $x = (1, 1, \ldots, 1)^T$ and the right hand side is determined as $b = Ax$. The initial guess set to be $x_0 = (0, 0, \ldots, 0)^T$. Convergence is reached when the relative residual drops below the tolerance of 10^{-8}:

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq 10^{-8}$$

The Implementation of both algorithms and the preconditioners was done in Python. To store the algorithms variables, the python library Numpy was used. Numpy provides many solving capabilities, however, for this exercise, only some basic operations like the 2-norm, dot product and matrix-vector multiplication were used.

2 GMRES

Task: For the full GMRES method, with and without preconditioning, plot the relative residual against iteration index on a semi-log scale. Compare and analyze the results obtained based on how many Krylov vectors were needed to converge in each case.

2.1 Setup

GMRES was implemented in its restarted version. This means, that as soon as m Krylov vectors are generated, the algorithm restarts with 0 Krylov vectors, but using the approximate solution x_k from the last iterations as the new initial guess. However, to prevent the algorithm from restarting, the restart parameter was set to $m = 600$, such that GMRES converged before 600 Krylov vectors were generated. In the implementation, preconditioning was applied optional within the GMRES algorithm in every Arnoldi iteration.

Numpy uses a highly optimized C backend to compute vector and matrix operations as efficiently as possible. This includes the parallelization of such operations. To exclude the effect of parallel computation on execution time, the python method time.process_time() was used to measure the single code CPU time or more specifically the cumulated time that the code ran on all CPU cores, that were used.

2.2 Results

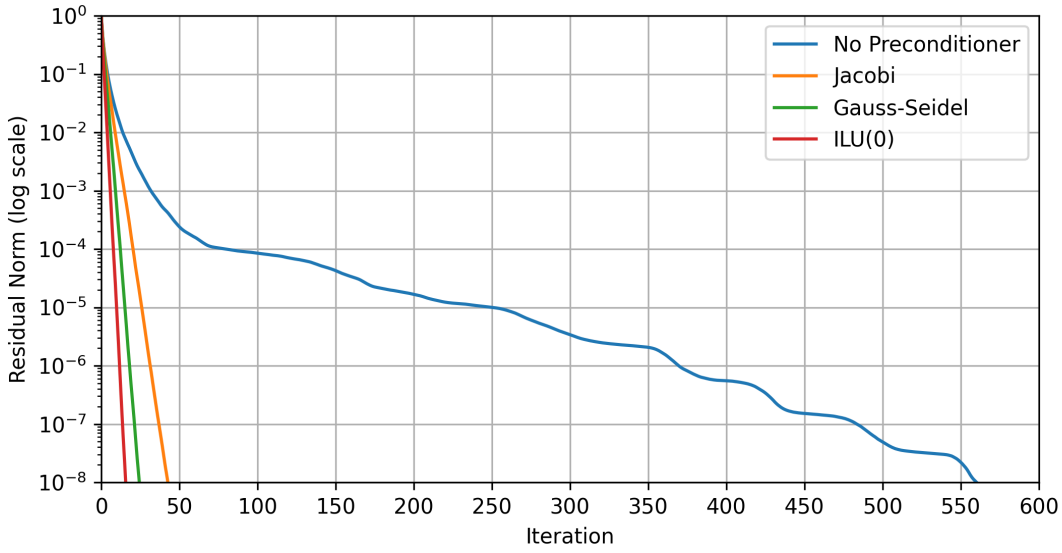

Figure 1: Convergence of full GMRES with and without preconditioning

In Figure 1, the GMRES algorithm is applied to a large, sparse matrix using first no preconditioning and then applying Jacobi, Gauss-Seidel and ILU(0) preconditioners to the linear system. The plot shows on its x-axis the number of Krylov vectors the GMRES generated to converge. In each iteration a new Krylov vector is generated. Each new basis

vector is orthogonalized against all previous vectors. Because of this, it is favorable to keep the Krylov space spanned by its basis vectors as small as possible. The application of the three preconditioners reduces the the number of Krylov vectors needed for convergence. This shows, that preconditioning the system can improve the GMRES algorithms convergence rate significantly.

Table 1: Convergence Results for Full GMRES with and without Preconditioning

Preconditioner	CPU Time [s]	Iterations	Converged
No Preconditioner	25.188	562	Yes
Jacobi	2.766	44	Yes
Gauss-Seidel	19.062	26	Yes
ILU(0)	5.453	17	Yes

Table 1 shows both the cumulative number of Krylov vectors (iterations) needed to reach convergence and the single core CPU time. However, looking at the CPU time, the improvement seems less significant. This is because the different preconditioners add different computational costs to each iterations. In this comparison, Jacobi, despite resulting in more iterations than ILU(0), is computationally inexpensive per iteration, making its total runtime the lowest.

Task: For GMRES(m), test and compare the following restart parameters m = 10, 50, 200 with the runtime required by the full GMRES.

Hints: Does the runtime increase or decrease compared to full GMRES? If the runtime changes, why does this happen? What factors, other than runtime, might motivate the use of restarts instead of full GMRES?

Optional: Recommend a restart parameter and justify your choice.

The following Figures 2, 3 and 4 show the convergence results of GMRES with and without preconditioning for different restart parameters m. Note, that for GMRES(10) the preconditioned systems diverged. The reason for this could not be determined.

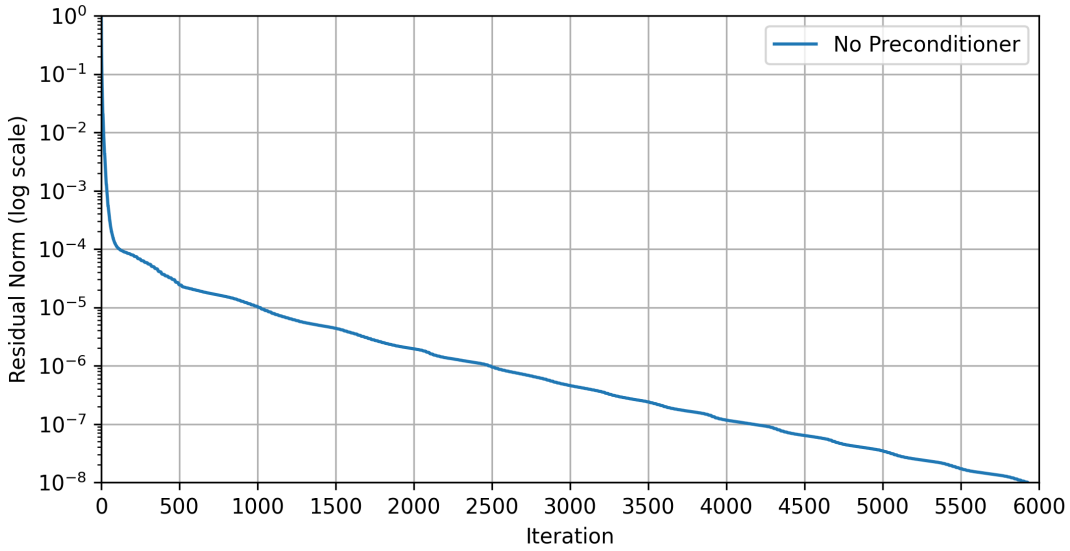

Figure 2: Convergence of GMRES(10) without preconditioning.

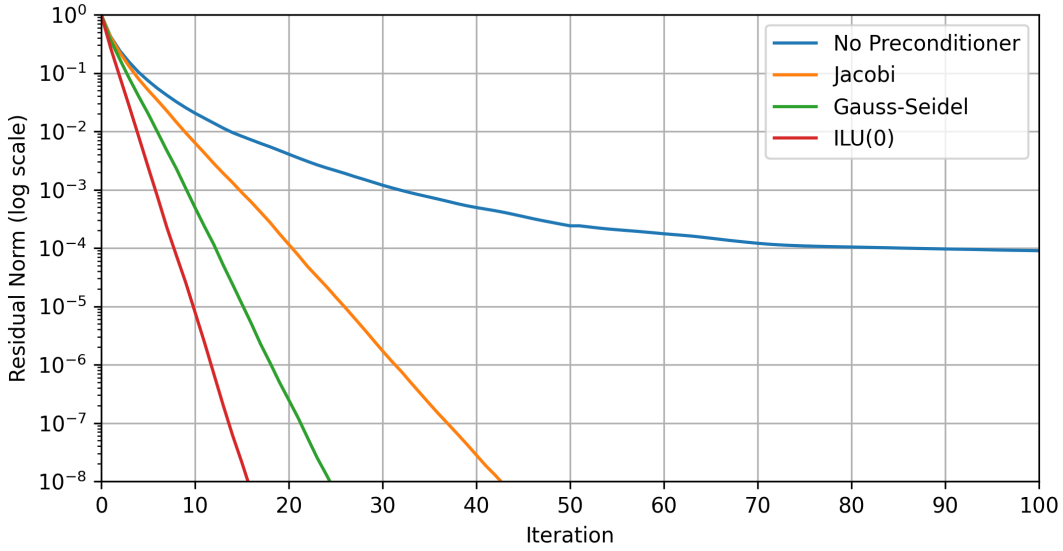

Figure 3: Convergence of GMRES(50) with and without preconditioning.

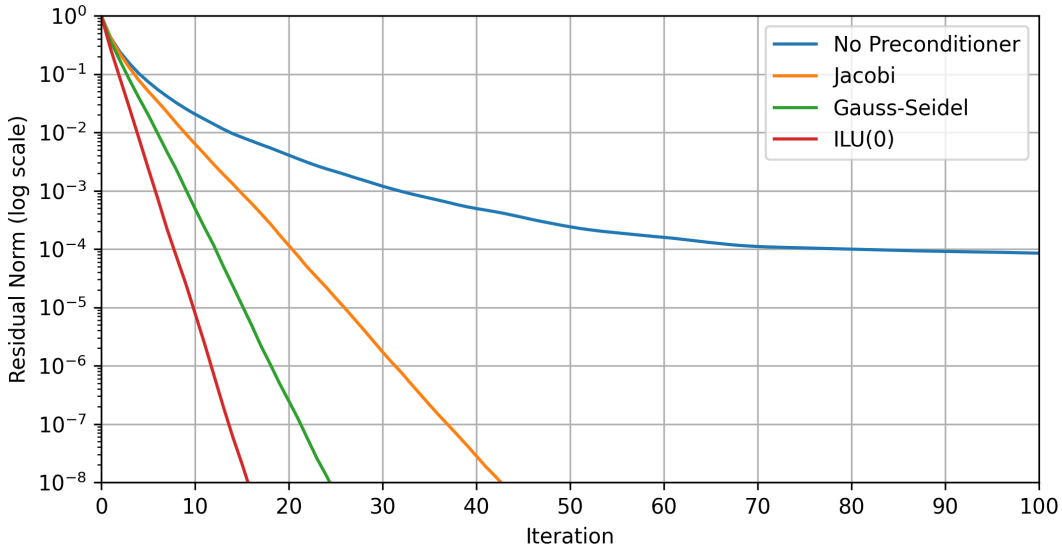

Figure 4: Convergence of GMRES(200) with and without preconditioning.

The first observation that can be made is the large increase in iterations for the unconditioned GMRES algorithm for smaller m. Choosing the restart parameter m too small can lead to slower convergence or even divergence (in the case of the preconditioned systems).

Table 2: GMRES Convergence with Varying Restart Parameter m. Preconditioned GMRES(10) failed to converge.

Preconditioner	Restart m	CPU Time [s]	Iterations	Converged
No Preconditioner	10	202.781	5929	Yes
Jacobi	10	-	-	No
Gauss-Seidel	10	-	-	No
ILU(0)	10	-	-	No
No Preconditioner	50	62.766	1813	Yes
Jacobi	50	2.781	44	Yes
Gauss-Seidel	50	19.141	26	Yes
ILU(0)	50	6.312	17	Yes
No Preconditioner	200	28.797	835	Yes
Jacobi	200	2.828	44	Yes
Gauss-Seidel	200	21.469	26	Yes
ILU(0)	200	4.531	17	Yes

Table 2 shows this trend more clearly. A higher restart parameter m reduces the number of Krylov vectors. Related to this, it can be observed, the CPU time decreases in the same way. Since GMRES(10) diverged for the preconditioned systems, no further analysis was conducted on possible improvements for these systems.

The optimal m to choose for the given system is therefore $m \geq 562$, as this is the number of Krylov vectors that the full GMRES needs to converge.

Task: For full GMRES (without preconditioning): check the orthogonality of the Krylov vectors.

Hints: Plot the computed values of v1 · vk against k on a semi-log scale.

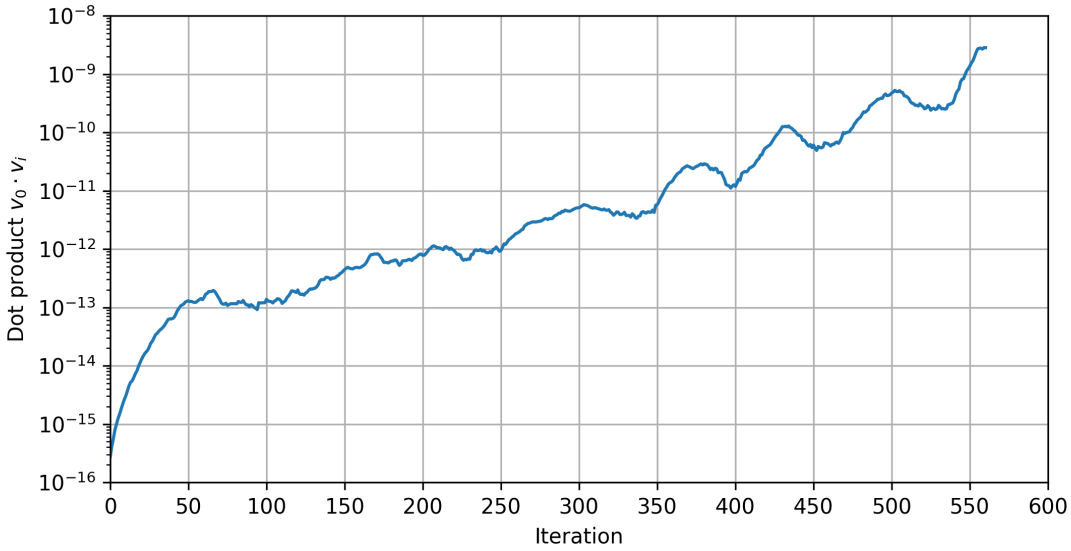

Figure 5: Orthogonality check for full GMRES Krylov vectors.

The dot product starts at 10^{-16} and rises quickly to 10^{-13}. From there, it increases further to 10^{-9}. This indicates that the Krylov vectors are not perfectly orthogonal. This loss of orthogonality may be due to the accumulation of round-off errors caused by finite-precision arithmetic. In particular, when multiplying floating-point numbers of different magnitudes, numerical inaccuracies can occur. Moreover, newly generated Krylov vectors tend to become shorter as the Krylov subspace expands, since the additional directions they introduce contribute less to the solution. The shorter a new vector v_k is, the more

susceptible it is to numerical noise, which can lead to a larger dot product with v_0.

Another contributing factor is the orthogonalization procedure itself. The (modified) Gram-Schmidt process, commonly used in GMRES, is sensitive to round-off errors, especially when many vectors are involved. As a result, the orthogonality between vectors can degrade as the iteration progresses.

3 Conjugate Gradient Method

Task: Plot the error in the A-norm, i.e., $\|e\|_A = \sqrt{\langle Ae, e\rangle}$, as well as the residual in the standard 2-norm, i.e., $\|r\|_2 = \sqrt{\langle r, r\rangle}$, against the iteration index on a semi-logarithmic scale for both cases. Compare qualitatively the difference in convergence between $\|e\|_A$ and $\|r\|_2$. Comment on your observations.

Compare the differences in convergence between the two CG cases. Comment what is happening, and why.

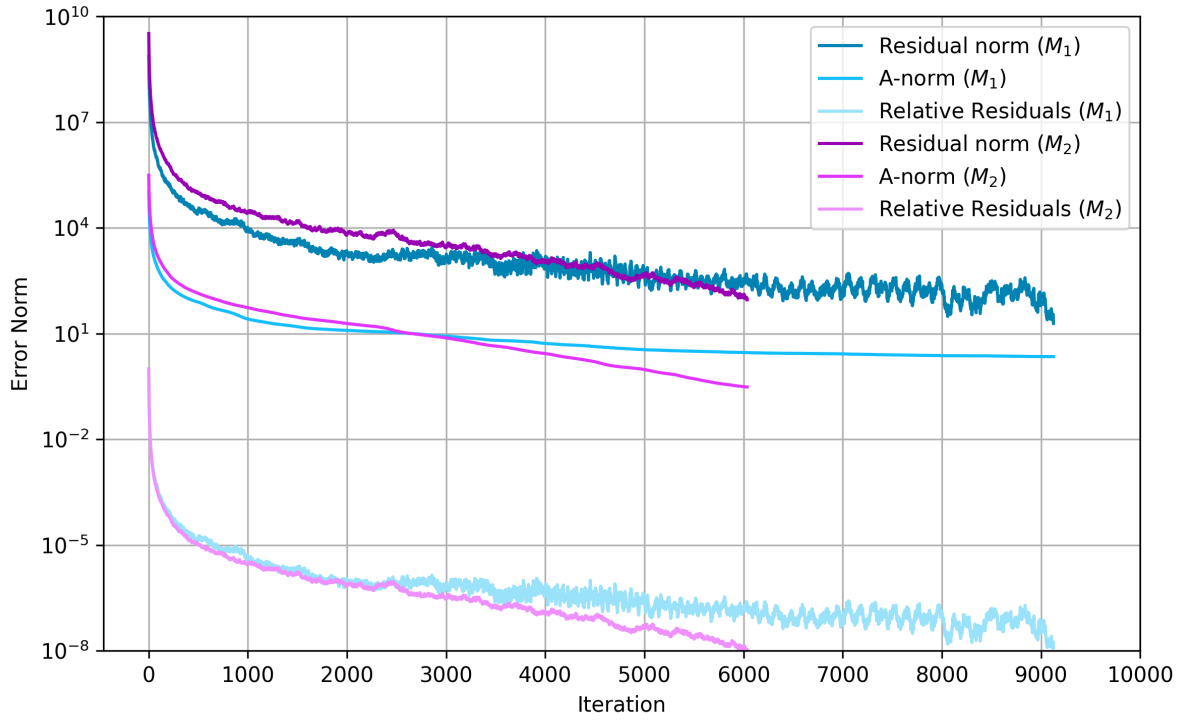

Figure 6: Convergence of Conjugate Gradient method for 2 matrices M_1 and M_2.

Table 3: Convergence Results for Conjugate Gradient Method

Matrix	CPU Time [s]	Iterations
M_1	100.453	9127
M_2	733.094	6037

The matrix $M_1 \in \mathbb{R}^{10^4 \times 10^4}$ leads to a shorter total runtime when solved with the Conjugate Gradient (CG) method compared to the larger matrix $M_2 \in \mathbb{R}^{10^5 \times 10^5}$. This is expected, as the computational cost per iteration increases with matrix size. However, fewer iterations were required to reach the convergence threshold (relative residual $< 10^{-8}$) for M_2 than for M_1.

It can be observed in Figure 6, that qualitatively, the residual norm $\|r_k\|_2$ is generally larger than the A-norm of the error $\|e_k\|_A$, where the A-norm is defined as

$$\|v\|_A = \sqrt{v^\top A v}, \quad \text{and the error } e_k = x_{\text{exact}} - x_k.$$

In both cases, all three quantities—the residual norm, the A-norm of the error, and the relative residual—decrease over iterations. What could explain the faster convergence of M_2? The convergence rate of the CG method depends primarily on the spectral properties of the matrix. A larger matrix size does not necessarily imply a worse condition number. In fact, the matrix M_2 might have a more favorable eigenvalue distribution, leading to better convergence.