Resolution of 1D Poisson Equation using LAPACK

Karolina Ochman – 26109551

PSc – Calcul scientifique et modélisation – 2025 - 2026

1 Execution Guide

The project is provided with a modular structure and a compilation script. To build and execute the program, one should use the provided shell script `projet.sh`. This script handles the compilation of the C source code, ensuring that the LAPACK library is correctly linked using the `-llapack` flags. To run the project, open a terminal in the project directory and execute the following command: `projet.sh`.

The execution will generate files named `results.tex` and `results_mono.tex` containing the numerical results formatted for LaTeX. The source code itself is organized into two files: a monolith version and a modular version utilizing `struct` definitions for `Matrix` and `Vector` types. The modular version includes separate functions for the tridiagonal matrix initialization, the matrix-vector product Ax, vector subtraction, and the calculation of the ℓ_1 norm.

2 Results

n	$\|A\tilde{x} - b\|_1 / \|b\|_1$
5	4.55975510e-07
55	1.54693735e-05
105	5.65556948e-05
155	1.24863087e-04
205	2.02303243e-04
255	2.07519537e-04
305	4.37262264e-04
355	5.55116276e-04
405	7.31018139e-04
455	9.66746476e-04
505	1.04911323e-03
555	1.24038360e-03
605	1.63891935e-03
655	1.54670596e-03
705	2.26850552e-03

Figure 1: Relative residual $\frac{\|A\tilde{x}-b\|_1}{\|b\|_1}$ for different n

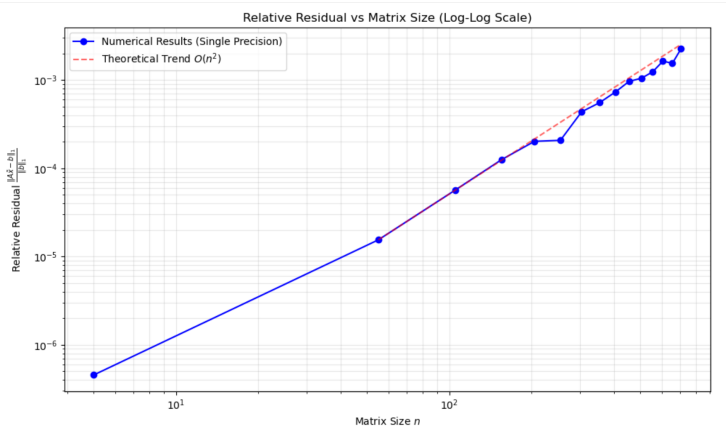

Figure 2: Relative residual vs matrix size n (log-log scale)

3 Discussion

Q1: Observation from numeric results

As indicated on the graph, as the size of the matrix (n) increases, the relative residual also increases. The growth of the residual is linked to the condition number of the matrix. The initial rapid increase in the residual for small n followed by a more linear trend suggests that while n is small, the system is well-conditioned, but as n grows, the $O(n^2)$ growth of the condition number $\kappa(A)$ becomes the primary driver of numerical instability. Furthermore, in single precision (float), we only have about 7 decimal digits of precision. As n increases, the "noise" from rounding errors starts to dominate and as a result the residual increases because of the accumulation of rounding errors.

Q2: Is SGESV optimal?

The use of `SGESV` is not optimal for this specific problem due to the properties of the matrix A. `SGESV` is a general-purpose solver that uses LU decomposition with partial pivoting, treating the matrix as a dense $n \times n$ array. It requires $O(n^2)$ storage, whereas for a tridiagonal matrix, we only need $O(n)$ space by storing just three vectors. For these cases, a better option would be a tridiagonal solver such as `SGTSV`. Specifically in our example, `SGESV` works well because the maximum value is $n = 705$. At this size, we store roughly $500,000$ floats, which is negligible for modern RAM. However, for a mesh where $n = 10^6$, `SGESV` would require approximately 4 TB of RAM, while a tridiagonal solver would require only 12 MB. Regarding time complexity, `SGESV` uses LU decomposition which is $O(n^3)$, while a tridiagonal solver like `SGTSV` (Thomas Algorithm) is $O(n)$, making it a lot more suitable for large-scale calculations.

Q3: Switching to double precision

For more precise calculations, we could consider switching the data type in our program from float to double. Double precision increases the significand from 24 bits to 53 bits, roughly doubling the number of decimal digits. It provides $\approx$ 15-17 significant digits compared to $\approx$ 7. Then, we also would need to use DGESV function in LAPACK, instead of SGESV. DGESV is an equivalent of SGESV, but it uses doubles instead of floats. Other required changes would include: fabsf to fabs where the norm is calculated, and values change 1.0f to 1.0. Summarizing, this change would reduce relative error for large values n.

4 Structural Implementation Comment

This version of the program is utilizing C structures for `Matrix` and `Vector` types. In LAPACK routines such as `SGESV` expect (Fortran-style) column-major storage. This implementation takes advantage of the fact that the discrete Laplacian matrix A is symmetric ($A = A^T$). Consequently, the row-major memory layout is equivalent to the column-major layout, allowing the solver function to run correctly without the need to transpose the matrix beforehand.