Project 1 FYS4150

Kjetil Karlsen and Vilde Mari Reinertsen

September 14, 2017

Abstract

We have studied the solution to a tridiagonal matrix by using a general, Gaussian elimination method, a specialized Gaussian elimination method and LU-decomposition, in order to do numerical derivation. By changing the step-size we have found that the relative error of the solutions genearly decreases until at a certain step-size, round-off errors increase the error. The computational cost was also analyzed for the three methods.

Contents

1 Introduction

In this project we got familiar with C++ and different aspects of numerical calculations. We examined round-off errors, floating point operations and the CPU time of different methods. We used both Gaussian elimination and LU decomposition to solve Poisson's equation for a certain function, $f(x)$, in our program. The source code can be found in this git repository.

The report starts with some theory on how to transform the analytical continuous equation to a discrete matrix equation. On this form, the problem can be solved numerically with different methods. After that, the different algorithms belonging to the different methods are described. Then, the results are presented. They include the solution of Poisson's equation compared to the analytical solution, with the error development, and the CPU time of the different methods. In the end a discussion about floating point operations, round-off errors and CPU time is given. At last the report is summarized in the conclusion.

All the theory and methods are based on chapters 2 and 6 from Jensen, [1]

2 Theory

In this project we will solve the general one-dimensional Poisson's equation (Eq. 1) for $f(x) = 100e^{-10x}$ and $x \in [0,1]$. The boundary conditions are Dirichlet bondary conditions. That means that $u(0) = u(1) = 0$.

2.1 Rewriting the problem to a matrix problem

$$-u''(x) = f(x) \tag{1}$$

We rewrite the second derivative by switching from the continuous $u(x)$ to n discrete number of points $v(x_i) = v_i$ and then using a Taylor expansion to obtain the new Poisson equation (Eq. 2).

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i \qquad \text{for } i = 1, \ldots, n, \tag{2}$$

where $f_i = f(x_i)$.

The next step is to rewrite Eq. 2 as a linear set of equations of the form of Eq. 3.

$$\frac{1}{h^2}\mathbf{A}\mathbf{v} = \mathbf{f}, \tag{3}$$

where $\mathbf{A}$ is an $n \times n$ tridiagonal matrix:

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

Writing out the equations, without including the constant $\frac{1}{h^2}$, gives us:

$$\begin{aligned} i = 1 &: -v_0 + 2v_1 - v_2 = f_1 \\ i = 2 &: -v_1 + 2v_2 - v_3 = f_2 \\ &\vdots \\ i = n &: -v_{n-1} + 2v_n - v_{n+1} = f_n \end{aligned}$$

where $v_0 = 0$ and $v_{n+1} = 0$.

Adding some zeros to the equations for illustrative purposes:

$$\begin{aligned} i = 1 &: \quad 2v_1 - v_2 + 0 + 0 + 0 + 0 + \cdots + 0 = f_1 \\ i = 2 &: \quad -v_1 + 2v_2 - v_3 + 0 + 0 + \cdots + 0 = f_2 \\ i = 3 &: \quad 0 - v_2 + 2v_3 - v_4 + 0 + \cdots + 0 = f_3 \\ &\vdots \qquad\qquad \vdots \\ i = n-1 &: \quad 0 + \cdots + 0 - v_{n-2} + 2v_{n-1} - v_n = f_{n-1} \\ i = n &: \quad 0 + \cdots + 0 + 0 + 0 - v_{n-1} + 2v_n = f_n \end{aligned}$$

We can now recognize the equations as a product between a matrix and a vector that equals another vector.

We define vectors that fits like this:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \qquad \mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}$$

The matrix is the matrix $\mathbf{A}$, and we include the constant $\frac{1}{h^2}$ again. Then we have:

$$\begin{bmatrix} 2 & -1 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 2 & -1 \\ 0 & 0 & \cdots & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix} \implies \frac{1}{h^2}\mathbf{A}\mathbf{v} = \mathbf{f}$$

2.2 An analytical solution

The Poisson equation (Eq. 1) with our function $f(x) = 100e^{-10x}$ has an analytical solution (Eq. 4).

$$u(x) = 1 - \left(1 - e^{-10}\right)x - e^{-10x} \tag{4}$$

That can be shown by putting it into the Poisson equation (Eq. 1). Then we first have to find the second derivative:

$$\begin{aligned} u(x) &= 1 - \left(1 - e^{-10}\right)x - e^{-10x} \\ u'(x) &= \left(1 - e^{-10}\right) + 10e^{-10x} \\ u''(x) &= -100e^{-10x} \end{aligned}$$

Second, we put the second derivative into Eq. 1:

$$-u''(x) = -(-100)e^{-10x} = 100e^{-10x} = f(x)$$

3 Method

3.1 Algorithm

The first algorithms that we used are based on Gaussian elimination. This method starts by making the matrix upper triangular, thereafter one can solve the set of linear equations by starting at the bottom of the matrix where there are only one unknown.

Our matrix is a tridiagonal matrix and therefore, just a few matrix operations were necessary to make the matrix upper triangular. Those are called forward substitution. The process of solving the set of equation from the bottom is called backward substitution.

3.2 The general tridiagonal matrix

The algorithm is first written for an equation with a general tridiagonal matrix on the short form:

$$\mathbf{Av} = \mathbf{f}$$

The full form looks like this:

$$\frac{1}{h^2}\begin{bmatrix} b_1 & c_1 & 0 & \cdots & 0 & 0 \\ a_1 & b_2 & c_2 & 0 & \cdots & 0 \\ 0 & a_2 & b_3 & c_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n-2} & b_{n-1} & c_{n-1} \\ 0 & 0 & \cdots & 0 & a_{n-1} & b_n \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}$$

The method used to solve the equation numerically was to first to a forward substitution to obtain a upper triangular matrix and then a backwards substitution to obtain the solution.

The algorithm preformed on the matrix values was:

The forward substitution:
The values on the diagonal:

$$\tilde{b}_i = b_i - \left(\frac{a_i \cdot c_i}{\tilde{b}_{i-1}}\right)$$

The values on the right side of the equation:

$$\tilde{f}_i = f_i - \left(\frac{\tilde{f}_{i-1} \cdot a_i}{\tilde{b}_{i-1}}\right)$$

for $i = 2, \ldots, n$

The backward substitution:

$$v_i = \frac{\left(\tilde{f}_i - c_i \cdot v_{i+1}\right)}{\tilde{b}_i}$$

for $i = n-1, \ldots, 1$. The boundary values are known ($v_0 = v_{n+1} = 0$). v_n was calculated first and like this:

$$v_n = \frac{\tilde{f}_n}{\tilde{b}_n}$$

3.3 This specific tridiagonal matrix

The matrix in our case had the same values on the different diagonals:

$$a_i = -1 \qquad b_i = 2 \qquad c_i = -1$$

We can then pre-calculate some of the operations in the algorithm and if becomes then:

The forward substitution:
The values on the diagonal:

$$\tilde{b}_i = b_i - \left(\frac{a_i \cdot c_i}{\tilde{b}_{i-1}}\right) = 2 - \left(\frac{(-1)\cdot(-1)}{\tilde{b}_{i-1}}\right) =$$

$$2 - \frac{1}{\tilde{b}_{i-1}} = \frac{i+1}{i}$$

The last expression can be shown to be true, by calculating some of the first values:

$$\tilde{b}_2 = 2 - \frac{1}{2} = \frac{3}{2} = \frac{2+1}{2}$$

$$\tilde{b}_3 = 2 - \frac{1}{\frac{3}{2}} = 2 - \frac{2}{3} = \frac{12}{6} - \frac{4}{6} = \frac{8}{6} = \frac{4}{3} = \frac{3+1}{3}$$

The values on the right side of the equation:

$$\tilde{b}_i = b_i - \left(\frac{\tilde{f}_{i-1} \cdot c_i}{\tilde{b}_{i-1}}\right) = b_i - \left(\frac{\tilde{f}_{i-1} \cdot (-1)}{\tilde{b}_{i-1}}\right) = b_i + \left(\frac{\tilde{f}_{i-1}}{\tilde{b}_{i-1}}\right)$$

If we put in $\tilde{b}_{i-1} = \frac{i}{i-1}$:

$$\tilde{f}_i = f_i + \frac{(i-1)\tilde{f}_{i-1}}{i}$$

for $i = 2, \ldots n$.

The backward substitution:

$$v_i = \frac{\left(\tilde{f}_i + v_{i+1}\right)}{\tilde{b}_i}$$

for $i = n-1, \ldots, 1$. The boundary values are known ($v_0 = v_{n+1} = 0$). v_n was calculated first and like this:

$$v_n = \frac{\tilde{f}_n}{\tilde{b}_n}$$

If we put in $\tilde{b}_{i-1} = \frac{i}{i-1}$:

$$v_i = \frac{(i-1)(\tilde{f}_i + v_{i+1})}{i}$$

$$v_n = \frac{(i-1)\tilde{f}_n}{i}$$

3.4 LU decomposition

The last method that we tried out is called LU decomposition. When using this method we factorized our matrix into two other matrices, one a lower triangular matrix and the other a upper triangular matrix. Here is an example for a 4×4-matrix:

$$A = LU = \begin{bmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix}$$

LU decomposition exists it the determinant of the matrix is non-zero.

The method goes like this:

$$\mathbf{Av} = \mathbf{f} \implies \mathbf{LUv} = \mathbf{f}$$
$$\mathbf{Uv} = \mathbf{L}^{-1}\mathbf{f} = \mathbf{w}$$

We first solve:

$$\mathbf{L}^{-1}\mathbf{f} = \mathbf{w}$$

Then we can easily get $\mathbf{x}$, because of the shape of $\mathbf{U}$, from:

$$\mathbf{Ux} = \mathbf{w} \implies \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

The bottom equation has only one unknown, so we start there.

3.5 Relative error

Relative error, ϵ is calculated as

$$\epsilon_i = \log_{10}\left(\frac{v_i - u_i}{u_i}\right) \tag{5}$$

, where v is the list of numerical values and u refers to the analytical result.

4 Result

4.1 The computer time

Table 4.1: This is a table listing the computing time for each algorithm. The LU algorithm was only possible to do up till $n = 1e4$ and the entry of 0 s means that this algorithm did not compute anything for the given value of n

n	general(s)	special(s)	LU(s)
1e1	1.5e-05	2.0e-06	1.8e-04
1e2	1.9e-05	5.0e-06	2.6e-03
1e3	4.8e-05	4.1e-05	2.7e-01
1e4	5.9e-04	3.3e-04	1.4e+02
1e5	5.2e-03	3.2e-03	0.0e+00
1e6	6.6e-02	4.0e-02	0.0e+00
1e7	5.8e-01	3.9e-01	0.0e+00

In the discussion section we will discuss the number of Floating Point Operations (FLOPS). There is however a limitation in the physical memory of the computer, which only allows us to create a certain size of matrix. Turns out we were not able to use the LU decomposition on the system for $n > 1e4$ matrixes on the form $n \times n$. Therefore there is no CPU-time for this algorithm.

It is also clear to see that the special algorithm is significantly quicker compared to the other algorithms, see section 5.1.

4.2 Nummerical precision

The relative error for the different values of h is represented in table 4.2. As these numbers are the maximum relative error of the entire matrix and does not represent a specific value of x. In order to generate the table we had to choose between the general and the special algorithm. As the special algorithm does not need to refer to a stored double in the script, but rather depends directly on h and an integer, this yields a smaller relative error for the smaller values of h compared to the general algorithm.

We can see that the smallest error, $log_{10}(RelativeError) = -10.15$, is accomplished for $h = 1 \cdot 10^{-6}$. As h decreases further, numerical round-off-errors become significant and the relative error increases.

Table 4.2: Maximum relative error (ϵ) for a given value of h. Both the special algorithm and the general was used to generate this table.

$\log_{10}$(h)	ϵ_{Max} , general	ϵ_{Max}, special
-1	-1.18	-1.18
-2	-3.09	-3.09
-3	-5.08	-5.08
-4	-7.08	-7.08
-5	-8.84	-9.08
-6	-6.08	-10.16
-7	-5.53	-9.09

For high values of h, numerical errors is not important and both the special and general algorithm gives similar errors.Therefore, the data plotted in figures 4.1 and 4.2 comes from the special algorithm. For high values of h, as in figure 4.1, the absolute error is significantly greater than for lower values of h, see figure 4.2.

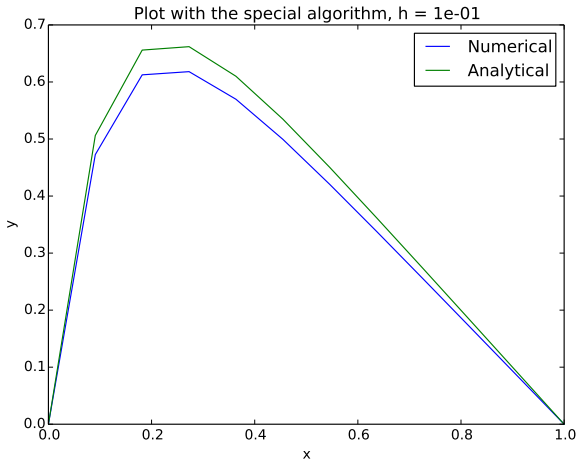

Figure 4.1: Plot of the results using the special algorithm comparing the numerical and analytical result for a 10×10 matrix. The nummerical solution is significantly different from the analytical one.

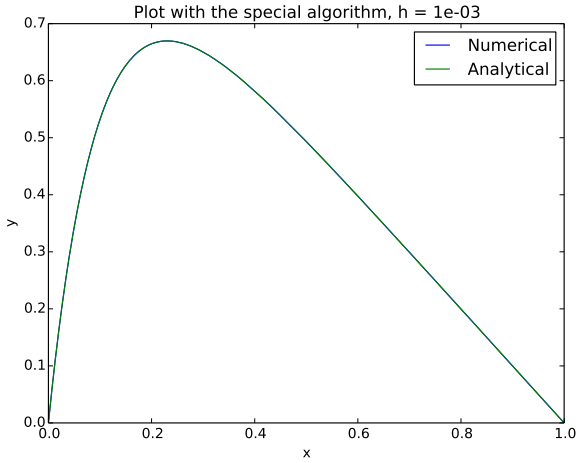

Figure 4.2: Plot of the results using the special algorithm comparing the numerical and analytical result for a $1e3 \times 1e3$ matrix. The errors are now impossible to see in this plot

5 Discussion

5.1 Floating point operations

A floating point operation is either addition, subtraction, multiplication or division. Comparing the number of floating point operations is a way to compare the cost of different algorithms. That way, one can find the most efficient algorithm to solve a problem.

5.1.1 Algorithm of the general tridiagonal matrix

The number of floating point operations in the Gaussian elimination method with the general tridiagonal can be found from the algorithm. The forward substitution looked like this:

$$\tilde{d}_i = d_i - \left(\frac{c_i \cdot e_i}{\tilde{d}_{i-1}} \right)$$

$$\tilde{b}_i = b_i - \left(\frac{\tilde{b}_{i-1} \cdot e_i}{\tilde{d}_{i-1}} \right)$$

for $i = 2, \ldots, n$

The first expression involves three floating point operations and these were preformed $n-1$ times.

The second expression also involves three floating point operations, so the sum of floating point operations from the forward substitution is:

$$3(n-1) + 3(n-1) = 6(n-1)$$

The algorithm for the backward substitution looked like this:

$$v_i = \frac{\left(\tilde{b}_i + e_i v_{i+1}\right)}{\tilde{d}_i}$$

for $i = 2, \ldots, n-1$. The last position, calculated first:

$$v_n = \frac{\tilde{b}_n}{\tilde{d}_n}$$

The first expression involves three floating point operations and is preformed $n-2$ times. The last involves one floating point operation and is preformed only once. The sum of floating point operations for the backward substitution is then:

$$3(n-2) + 1 = 3n - 5$$

The sum of floating point operation for the whole Gaussian elimination is then:

$$6n - 6 + 3n - 5 = 9n - 11 \approx 9n$$

5.1.2 Algorithm of the specific tridiagonal matrix

The number of floating point operations in the Gaussian elimination method with the specific tridiagonal can be found from the algorithm. The forward substitution looked like this:

$$\tilde{d}_i = \frac{i+1}{i}$$

$$\tilde{b}_i = b_i + \frac{\tilde{b}_i}{\tilde{d}_i}$$

for $i = 2, \ldots, n$

The first expression involves no floating point operations, it can be pre-calculated and does not count.

The second expression also involves two floating point operations, so the sum of floating point operations from the forward substitution is:

$$2(n-1) = 2n-2$$

The algorithm for the backward substitution looked like this:

$$v_i = \frac{\left(\tilde{b}_i + v_{i+1}\right)}{\tilde{d}_i}$$

for $i = 2, \ldots, n-1$. The last position, calculated first:

$$v_n = \frac{\tilde{b}_n}{\tilde{d}_n}$$

The first expression involves two floating point operations and is preformed $n-2$ times. The last involves one floating point operation and is preformed only once. The sum of floating point operations for the backward substitution is then:

$$2(n-2) + 1 = 4n - 3$$

The sum of floating point operation for the whole Gaussian elimination is then:

$$2n - 2 + 2n - 3 = 4n - 5 \approx 4n$$

5.1.3 Gaussian elimination in general

The Gaussian elimination method of solving a set of linear equations requires n^3 floating point operations for a regular $n \times n$-matrix, that is not tridiagonal or any other special type of matrix.

5.1.4 LU decomposition

The floating point operations for the LU decomposition method can be separated in two parts; the decomposing of the matrix into an upper triangular matrix multiplied with a lower triangular matrix and the calculation of the set of equation with different 'right hand side'-vectors.

The number of floating point operations required to LU decompose a matrix scales to n^3 for a n×-matrix. After the decomposition the floating points of the calculation of the set of equations scales to n^2 for a $n \times n$-matrix.

If one was to use the LU decomposition method for a $10^5 \times 10^5$-matrix it would require $\left(10^5\right)^3 + \left(10^5\right)^2 \approx 10^{15}$ floating point operations. If a computer calculates 10^9 floating points per second (flops) it would take $\frac{10^{15}\text{flop}}{10^9\text{flops}} = 10^6$ seconds, which is approximately 12 days. That is too long.

5.1.5 Comparison

It is easy to see that the Gaussian elimination method is more effective than the LU-decomposition, both from this discussion and table 4.2. It needs the least number of floating point operations. It is also possible to make the algorithm even more effective by specializing it to our tridiagonal matrix which has the same values on the diagonal and on the super- and sub-diagonal.

If however we had a regular random matrix and were to solve the equations for many different 'right-hand-side'-vectors, the best method of these would be the LU decomposition. Then we would only have to decompose the matrix once, which require n^3 floating point operations for an $n \times n$-matrix, and for every new 'right-hand-side'-vector it would only require n^2 floating point operations. Compared with the Gaussian elimination method that requires n^3 floating point operations to solve the set of equations every time.

The number of FLOPS is also reflected in the times the different algorithms use on the CPU, as shown in table 4.1.

5.2 Round-off error

To a given value, the relative error decreases as h decreases. From table 4.2 we see that the relative error also increases at a certain value of h and the nummerical round-off-errors come into play. The interesting part is that this happens for different values of h in the general algorithm and the special algorithm, with the special algorithm getting round-off errors much later than the general does. The general algorithm also has higher round off errors, which is possible to see for $log_{10}(h) = -8$.

This is likely due to the fact that the general algorithm refers to stored doubles. It does this to a much larger extent while running the algorithm, but also when it is calculating the relative error. On the other hand, the special algorithm refers much less to stored variables and does not refer to a list while other than the numerical one, while calculating the error.This means that the value of u from equation 5 might be more accurate as it is used both in the mentioning and the counter.

6 Conclusion

We found that the LU-decomposition was by far the most costly algorithm and also heavily limited the step-size h of the computations. It was also found that the specialized algorithm was quite a lot faster compared to the general. This specialized algorithm was also possible to run with a higher level of resolution (lower h) with the benefit of the lowest relative errors of this experiment.

The specialized algorithm requires more thinking in advance compared to the general, but is around twice as fast. However it is by far the more superior algorithm

References

[1] Morten Hjorth-Jensen. Computational physics: Lecture notes fall 2015. Department of Physics, University of Oslo, 8 2015. Chapter 2 and 6.