Project 1
FYS3150/4150

Lasse Braseth, Nicolai Haug, and Kristian Wold

September 10, 2018

All programs used in this project can be found at the GitHub repository referenced below. This project is based on the course material. In particular Section 6.4 and subsequent sections in [2] have been used as a basis to complement the project problem description in [1].

Abstract

In this project the solution of the one-dimensional Poisson equation, a second-order differential equation, is approximated numerically with three different algorithms based on recasting the Poisson equation as a tridiagonal matrix equation by discretization. The first algorithm is in the spirit of the Thomas algorithm, using a simple form of Gaussian elimination and backward substitution. The second algorithm optimizes the first algorithm by specializing it to the specific matrix spawned by the discretization of the Poisson equation. The third algorithm use the built-in LU-decomposition procedure in the C++ library Armadillo. Testing the algorithms on a case with a known closed form, it was found that all algorithms approximated the solution satisfactory for a relatively low number of grid-points, without much strain on the CPU and memory. Comparing the CPU times of each algorithm, it was found that the optimized algorithm was about twice as fast as the general algorithm, and about 3000 times as fast as the method using LU-decomposition. It was found that the LU-decomposition method was heavily memory-bound, only achieving of the order $n = 10^4$ grid-points before the computer crashed. The other two methods on the other hand easily handled $n = 10^7$ grid-points. The relative error of the optimized algorithm was found to scale as h^2, where h is the step-size, in accordance with the expected theoretical relative error. The relative error was found to be the smallest for $n = 10^5$ grid-points, also in accordance with the calculated analytical error.

1 Introduction

This project explores three different numerical algorithms for solving the Poisson equation, a second-order differential equation, recast as a tridiagonal matrix by discretization. The first algorithm, labeled "General Algorithm", use the Gaussian elimination procedure and backward substitution as described by the Thomas algorithm. The second algorithm, labeled "Optimized Algorithm", is an optimalization of the General Algorithm by specializing it to the specific matrix spawned by the discretization of the Poisson equation, known as a Toeplitz matrix. The third algorithm, labeled "LU-decomposition Algorithm" use the built-in LU-decomposition method in the C++ library Armadillo. The subject of particular interest in this project is the computational performance of the algorithms. This will be studied by comparing the difference in FLOPS (floating point operations per second) and computational time between the algorithms. A study of the numerical error produced by the approximations and machine error will also be carried out, as well as a visual comparison between a known closed form of the Poisson equation and the numerical solution.

This project is structured by first presenting a theoretical overview of the problem, followed by a derivation of the aforementioned algorithms. Next, the results from the implementation of the algorithms are presented, before lastly they and the approach are discussed and concluded upon.

2 Theory

2.1 The Poisson Equation

The Poisson equation is a classical equation which in electromagnetism describe the electrostatic potential Φ generated by a localized charge distribution $\rho(\boldsymbol{r})$. In three-dimensional space it reads

$$\nabla^2\Phi(\boldsymbol{r}) = -4\pi\rho(\boldsymbol{r}), \tag{2.1}$$

where ∇^2 is the Laplace operator. By assuming that Φ and ρ are both spherically symmetric, the potential is just dependent on the distance r from the source and can be written as

$$\nabla_r^2\Phi(r) = -4\pi\rho(r), \tag{2.2}$$

where the Laplace operator ∇_r^2 is given by

$$\nabla_r^2 = \frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d}{dr}\right)$$

The above equation can thus further be rewritten as

$$\frac{d^2\Phi(r)}{dr^2} = -4\pi r\rho(r) \tag{2.3}$$

The equation is now a one-dimensional problem, so by applying the substitutions $\Phi \to u$ and $r \to x$ the equation is represented as follows

$$-u''(x) = f(x), \tag{2.4}$$

where $f(x)$ is a source term.

In this project we will solve the one-dimensional Poisson equation Eq. (2.4) for $x \in (0,1)$ and Dirichlet boundary conditions $u(0) = u(1) = 0$.

We study the source term

$$f(x) = 100e^{-10x} \tag{2.5}$$

with known closed form solution

$$u(x) = 1 - (1 - e^{-10})x - e^{-10x} \tag{2.6}$$

2.2 Approximation to the Second Derivative

In order to solve this equation numerically we have to discretize it. Let us start of by doing a Taylor expansion of u(x)

$$u(x \pm h) = u(x) \pm hu' + \frac{h^2}{2}u'' \pm \frac{h^3}{3!}u''' + \mathrm{O}(h^4)$$

Now we take the sum of the two

$$u(x+h) + u(x-h) = 2u(x) + h^2u'' + \mathrm{O}(h^4)$$

If we solve this equation for u'' we have the approximation for the second derivative

$$u'' = \frac{u(x+h) + u(x-h) - 2u(x)}{h^2} + \mathrm{O}(h^2)$$

Where $\mathrm{O}(h^2)$ is the truncation error, which we will come back to in our derivation of the error in Eq. (2.3). Using the formalism $u(x) \to v_i$, $u(x \pm h) \to v_{i\pm1}$ on Eq. (2.4) we see that our differential equation assumes the discrete form

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i \tag{2.7}$$

The boundary conditions are $v_0 = v_{n+1} = 0$. If we now write out the explicit terms we get

$$\begin{aligned}
-v_2 + 2v_1 &= h^2 f_1 \\
-v_3 - v_1 + 2v_2 &= h^2 f_2 \\
-v_4 - v_2 + 2v_3 &= h^2 f_3 \\
&\vdots \\
-v_n - v_{n-2} + 2v_{n-1} &= h^2 f_{n-1} \\
-v_{n-1} + 2v_n &= h^2 f_n
\end{aligned}$$

And we immediately see that this is exactly the same as writing

$$\begin{bmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \vdots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix} = h^2 \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}, \tag{2.8}$$

Thus we obtain the matrix equation

$$A\boldsymbol{v} = \tilde{\boldsymbol{f}}$$

2.3 Analytical Error

The function $u(x)$ is approximated with a Taylor expansion

$$u(x \pm h) = u \pm hu^{(1)} + \frac{h^2}{2}u^{(2)} \pm \frac{h^3}{3!}u^{(3)} + \frac{h^4}{12}u^{(4)} + \dots$$

By truncating the expansion the second derivative can be written as

$$u'' = \frac{u(x+h) + u(x-h) - 2u(x)}{h^2} - \frac{u^{(4)}}{12}h^2$$

where the truncation error is

$$\epsilon_{trunc} = \frac{u^{(4)}}{12}h^2 \tag{2.9}$$

By setting h as small as possible we could in theory get a error as small as possible. In a computer this is not achievable due to the finite amount of information stored, so we need an extra term due to loss of numerical precision. The error can then be written as a sum of the truncation error and a roundoff error.

$$\epsilon = \epsilon_{trunc} + \epsilon_{roundoff}$$

In order to find the limit for the roundoff error one can rewrite the approximation of the second derivative in the following way

$$u'' = \frac{[u(x+h) - u(x)] + [u(x-h) - u(x)]}{h^2}$$

the denominator will at some point end in a subtraction of two nearly equal numbers, and will thus lead to loss in numerical precision. By demanding that this roundoff has to be smaller than or equal to the machine error

$$|u''| = |\frac{[u(x+h) - u(x)] + [u(x-h) - u(x)]}{h^2}| \leq \frac{2\epsilon_M}{h^2}$$

The total error is then bounded by

$$|\epsilon| \leq \frac{2\epsilon_M}{h^2} + \frac{u^{(4)}}{12}h^2$$

In order to find the optimal h we can differentiate this expression with respect to h and set it equal to zero. The optimal h is then equal to

$$h = \left(\frac{24\epsilon_M}{u^{(4)}}\right)^{1/4} \tag{2.10}$$

With double precision, $\epsilon_M \leq 10^{-15}$ and Eq. (2.6) gives that, $u^{(4)}(0) = 10000$. By insertion the optimal value is found to be

$$h \approx 10^{-5} \tag{2.11}$$

3 Method

3.1 General Algorithm

We will solve the set of linear equations $A\boldsymbol{v} = \boldsymbol{f}$ by first applying a forward substitution followed by a backward substitution. The matrix A is called a tridiagonal matrix and we can thus define three vectors $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ which contains all the diagonal terms. In order to show the flow of the algorithm we use a $4x4$ matrix

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 \\ a_1 & b_2 & c_2 & 0 \\ 0 & a_2 & b_3 & c_3 \\ 0 & 0 & a_3 & b_4 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix}$$

We want to eliminate the lower diagonal to make the matrix upper-diagonal. To knock out a_1, we can multiply the first row by a_1/b_1 and subtract it from the second

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 \\ 0 & b_2 - \frac{a_1}{b_1}c_1 & c_2 & 0 \\ 0 & a_2 & b_3 & c_3 \\ 0 & 0 & a_3 & b_4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 - \frac{a_1}{b_1}f_1 \\ f_3 \\ f_4 \end{pmatrix}$$

To eliminate further, we treat $b_2 - \frac{a_1}{b_1}c_1$ as the new value for b_2 and we treat $f_2 - \frac{a_1}{b_1 f_1}$ as the new value for f_2. If we now carry out the same method for the next rows we eventually have a upper diagonal matrix of the form

$$\begin{pmatrix} b_1^* & c_1 & 0 & 0 \\ 0 & b_2^* & c_2 & 0 \\ 0 & 0 & b_3^* & c_3 \\ 0 & 0 & 0 & b_4^* \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = \begin{pmatrix} f_1^* \\ f_2^* \\ f_3^* \\ f_4^* \end{pmatrix}$$

Where we have the following equations

$$\begin{aligned} b_1^* &= b_1 \\ b_i^* &= b_i - \frac{a_{i-1}c_{i-1}}{b_{i-1}^*} \\ f_1^* &= f_1 \\ f_i^* &= f_i - \frac{a_{i-1}f_{i-1}^*}{b_{i-1}^*} \end{aligned}$$

We implement these equations in the following algorithm

```
for(int i=1; i<n; i++)
{
    temp = a[i]/b[i]
    b[i+1] = b[i+1] - temp*c[i]
    f[i+1] = f[i+1] - temp*f[i]
}
```

Note that we update b and f in the loop, so we do not need to make new vectors b^* and f^*. The c diagonal stays the same during the elimination, since the elements above is always zero. We need not carry out the calculations for changing the a-values either, since they will be zero after the elimination, and thus the forward substitution is finished.

To perform a backward substitution, we look at the upper diagonal matrix and multiply the last line with c_3/b_3 and subtract that from the third line which results in

$$\begin{pmatrix} b_1^* & c_1 & 0 & 0 \\ 0 & b_2^* & c_2 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = \begin{pmatrix} f_1^* \\ f_2^* \\ (f_3^* - c_3\tilde{f}_4)/b_3^* \\ \tilde{f}_4 \end{pmatrix}$$

We further eliminate c_2 by defining $(f_3^* - c_3\tilde{f}_4^*)/b_3^*$ as the new value for v_3 and do the same procedure as above. This results in the following equation

$$v_i = (f_i^* - v_{i+1}c_i)/b_i^*$$

v_i is implemented in the following algorithm

```
for(int i=n-2; i>0; i--)
{
    v[i] = (f[i]-c[i]*v[i+1])/b[i]
}
```

A important aspect in numerical computations is the efficiency of the algorithm, so it is therefore essential to keep track of the floating point operations. The forward substitution has 5 FLOPS per iteration with (n-1) iterations, which gives 5(n-1) FLOPS for the entire loop. The backward substitution has 3 FLOPS per iteration with (n-1) iterations, resulting 3(n-1) FLOPS. In total we then have 8(n-1) FLOPS for the General Algorithm.

3.2 Optimized Algorithm

If we now use the fact that all $\boldsymbol{a_i} = -1$ and all $\boldsymbol{c_i} = -1$, we can simplify the equations. The forward substitution can hen be written as

$$b_i^* = b_i - \frac{1}{b_{i-1}^*}$$
$$f_i^* = f_i - \frac{f_{i-1}^*}{b_{i-1}^*}$$

Which we implement in the following algorithm

```
for(int i=1; i<n; i++)
{

    b[i+1] = b[i+1] - 1/b[i]
    f[i+1] = f[i+1] + f[i]/b[i]
}
```

As we see this decreases number of FLOPS from 5 to 4 per cycle. For the backward substitution we get

$$v_i = (f_i^* + v_{i+1})/b_i^*$$

Which we implement as follows

```
for(int i=n-2; i>0; i--)
{
    v[i] = (f[i] +v[i+1])/b[i]
}
```

And this decreases the number of FLOPS from 3 to 2 per cycle, and thus the number of FLOPS has reduced down to 6(n-1) in total.

3.3 Relative Error

The relative error in a data set is a measure of the discrepancy between an exact value and some approximation to it. The relative error in our data set $i = 1, ..., n$ is given by

$$\eta_i = \left(\left| \frac{v_i - u_i}{u_i} \right| \right), \tag{3.1}$$

where u_i is the exact value, and v_i is the approximation to it. To easier interpret the relative error, we will use a log-log graph which is obtained by using logarithmic scales on both the horizontal and vertical axes. The relative error is thus computed by

$$\epsilon_i = \log_{10}(\eta_i) = \log_{10} \left(\left| \frac{v_i - u_i}{u_i} \right| \right) \tag{3.2}$$

The slope of the log-log graph will indicate the proportionality between the step-size h and the relative error η.

The graph will be obtained by computing the relative error for the function values u_i and v_i for different step-sizes, and extracting the maximum value of the relative error for each step-length h. The step-size is determined by the number of grid-points, and the number of grid-points for the different step-lengths will be $n = 10^1, 10^2, ..., 10^7$. For the numerical approximation, the Optimized Algorithm will be used.

3.4 LU-decomposition Algorithm

Another frequently used Gaussian elimination is the LU-decomposition. If a quadratic matrix is invertible, then the matrix can be written as a product of one lower and one upper diagonal matrix

$$A = LU \tag{3.3}$$

Thus we can write

$$\begin{aligned}
\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} &= \begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix} \\
&= \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ u_{11}l_{21} & u_{12}l_{21} + u_{22} & u_{13}l_{21} + u_{23} \\ u_{11}l_{31} & u_{12}l_{31} + u_{22}l_{32} & u_{13}l_{31} + u_{13}l_{31} + u_{23}l_{32} + u_{33} \end{pmatrix}
\end{aligned}$$

Now we see that we get the following relations for u

$$\begin{aligned}
u_{11} &= a_{11} \\
u_{12} &= a_{12} \\
u_{13} &= a_{13} \\
u_{22} &= a_{22} - u_{12}l_{21} \\
u_{23} &= a_{23} - u_{13}l_{21} \\
u_{33} &= a_{33} - (u_{13}l_{31} + u_{23}l_{32})
\end{aligned}$$

For l we get

$$\begin{aligned}
l_{21} &= \frac{1}{u_{11}} a_{21} \\
l_{31} &= \frac{1}{u_{11}} a_{31} \\
l_{32} &= \frac{1}{u_{22}} (a_{32} - u_{12}l_{31})
\end{aligned}$$

We can now see the pattern, and in general we write this as

$$\begin{aligned}
u_{ij} &= a_{ij} - \sum_{k=1}^{i-1} u_{kj}l_{ik} \\
l_{ij} &= \frac{1}{u_{jj}} (a_{ij} - \sum_{k}^{j-1} u_{kj}l_{ik})
\end{aligned}$$

LU-decomposition for general matrices scales as N^3. This will be important when we later compare times used by the different methods.

4 Results

4.1 Comparing the Analytical and Numerical Solution

In Fig. 1 is the visual comparison between the analytical solution, given by Eq. (2.6), and the numerical approximation generated by the General Algorithm, see Section 3.1, for $n = 10$, $n = 100$, and $n = 1000$ grid-points.

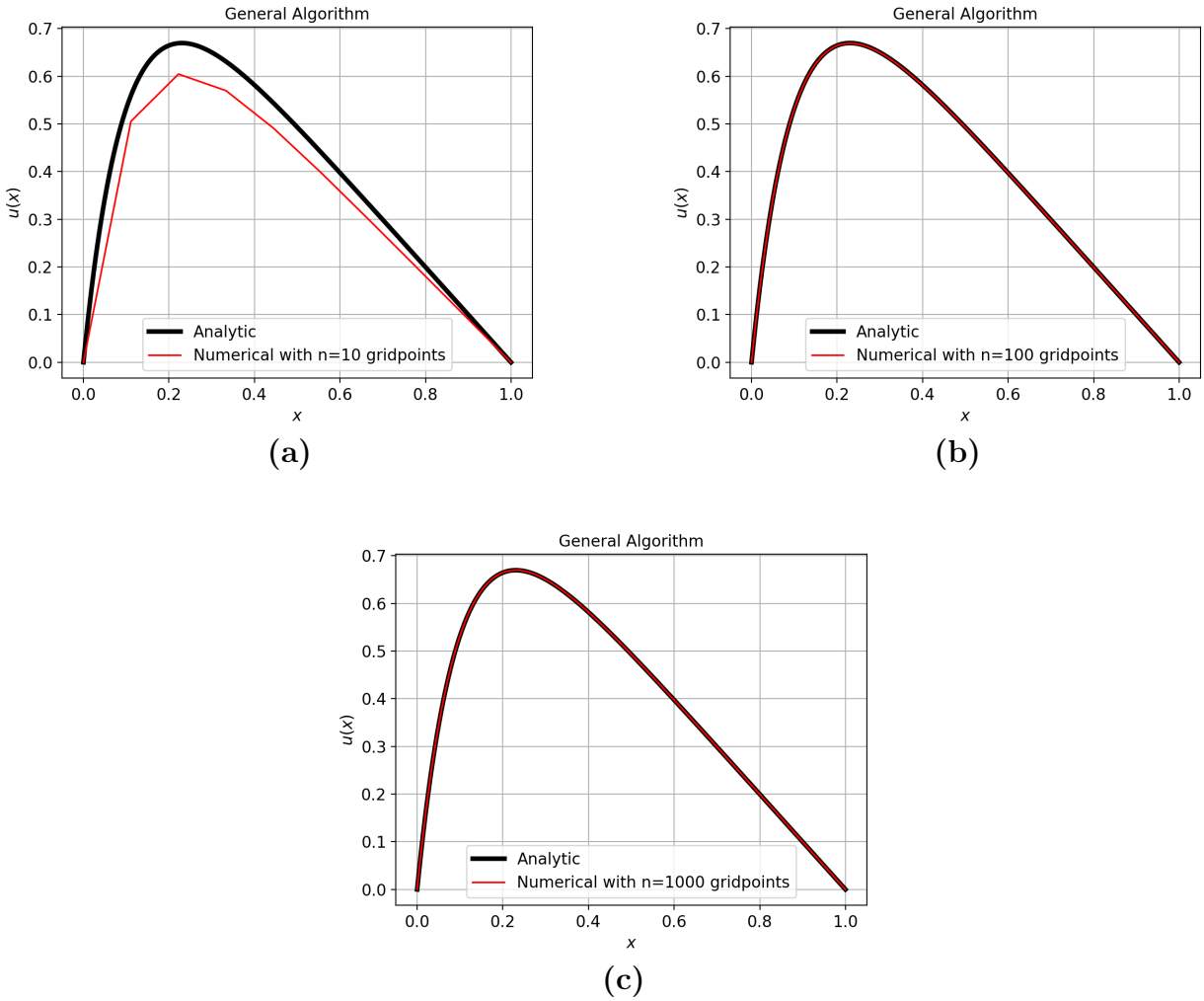

Figure 1: *This figure shows the numerical solution (red) to the discretized Poisson equation, given by Eq. 2.8, solved with the General Algorithm, see Section 3.1, plotted against the exact solution (black) with the closed-form given by Eq. 2.6. The number of grid-points in* ***(a)*** *is* $n = 10$*,* ***(b)*** *is* $n = 100$*, and* ***(c)*** *is* $n = 1000$*.*

4.2 Relative Error

In Fig. 2 is the relative error, given by Eq. (3.2), plotted as a log-log plot with a graph with a slope of 2 for reference. The plot is obtained by storing the maximum value of the relative error

in the data set $i = 1, ..., n$ for each step-length h, as described in Section 3.3.

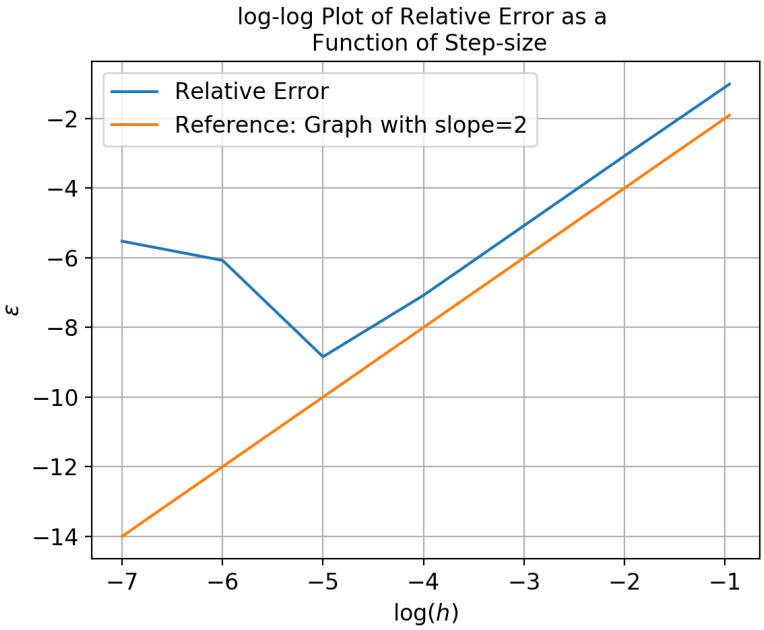

Figure 2: *Relative error of the numerical approximation generated by the Optimized Algorithm for* $n = 10^1, 10^2, ..., n = 10^7$ *grid-points.*

4.3 CPU Performance for the Different Algorithms

In Table 1 the CPU-time in seconds of all the algorithms is listed with respect to the number of grid points. The result is obtained by taking the mean value of five runs.

Table 1: *CPU-time for various methods and grid-points in seconds. The CPU-time is an average of 5 runs.*

Time Performance			
Number of Grid-points	**General Algorithm CPU Time in [s]**	**Optimized Algorithm CPU Time in [s]**	**LU-decomposition Algorithm CPU Time in [s]**
10	$1.15438 \cdot 10^{-5}$	$3.882 \cdot 10^{-7}$	$1.06497 \cdot 10^{-4}$
10^2	$2.36906 \cdot 10^{-5}$	$3.084 \cdot 10^{-6}$	$1.052824 \cdot 10^{-3}$
10^3	$6.79306 \cdot 10^{-5}$	$2.67376 \cdot 10^{-5}$	$9.896708 \cdot 10^{-2}$
10^4	$5.146074 \cdot 10^{-4}$	$2.747224 \cdot 10^{-4}$	2.323462
10^5	$5.125864 \cdot 10^{-3}$	$2.14747 \cdot 10^{-3}$	–
10^6	$4.7193 \cdot 10^{-2}$	$2.600468 \cdot 10^{-2}$	–

5 Discussion

5.1 Comparing the Analytical and Numerical Solution

From figure Fig. 1, we see the obvious trend that the numerical solution converges quickly towards the analytical solution Eq. (2.6) for increasing n. Already for $n = 100$ and $n = 1000$, the numerical solution is indistinguishable from the analytical as seen in the figures. This trend of fast convergence is expected, since the error caused by the discretization is expected to be proportional to the square of the step-size h according to Eq. (2.9).

5.2 Relative Error

The relation between the relative error Eq. (3.1) and step-size h is further verified by looking at the log-log plot Fig. 2 of the relative error as a function of h. For $log(h)$ between -1 and -5 (corresponding to n between 10 and 10^5), the relation is close to linear with slope 2, indicating that the relative error scales as h^2 as expected. This relation is however broken for $n > 10^5$ due to insufficient machine precision, causing roundoff-errors in the algorithm. This extremal point happens around $h = 10^{-5}$, which is in accordance with Eq. (2.11).

5.3 CPU Performance for the Different Algorithms

For $n = 10^5$ and higher, the solution using LU-decomposition from Armadillo caused the computer to freeze. This is not due to too long CPU-time, but rather too little memory available. Even though we have a sparse matrix, meaning most of the elements are zero, Armadillo need to explicitly store every element in memory with double precision. This results in $10^5 \cdot 10^5 * 8\text{B} \approx 80\text{GB}$, not even including the pointers that also need to be stored in memory. This far exceeds our available 8GB of RAM, making Armadillo's general LU-decomposition unviable for large numbers of grid-points. Our specialized method need only store the diagonal, source function and solution, totaling a mere $3 \cdot 10^5 \cdot 8\text{B} \approx 2.1\text{MB}$

6 Conclusion

We have experienced that simplifying algorithms have a significant advantage when solving problems numerically. We have also seen that numerical calculations poses problems due to memory restrictions. The optimized algorithm was able to solve the Poisson equation nearly twice as fast as the general algorithm, and nearly three thousand times faster than the LU-decomposition. This was in accordance to our expectations as the optimized algorithm reduces from $8n$ FLOPS to $6n$ FLOPS, and the optimized algorithm removes the heavy calculation of multiplication and division, such that CPU-time reduces by fifty percent. We also know that LU-decomposition scales as n^3, which also was in correspondence to our results.

References

This project is based on the course material, referenced below as [1] and [2]. In particular Section 6.4 and subsequent sections in [2] have been used as a basis to complement the project problem description in [1].

[1] FYS3150/4150 Computational Physics I: Project 1 problem description. Department of Physics, University of Oslo (August 6, 2018)

[2] Jensen, M. H. "Computational Physics - Lecture Notes Fall 2015". Department of Physics, University of Oslo (2015)