# Project 1 - FYS4150

Gustav Bredal Erikstad, Eirik Elias Prydz Fredborg
& Marte Cecilie Wegger

September 8, 2020

**Abstract**

An instance of the Poisson equation is solved using both a general and special case of the Thomas algorithm, and the LU decomposition algorithm from the Armadillo library. The CPU times of these are compared and found to be consistent with their number of FLOPs. The maximum relative error of the special case Thomas algorithm is evaluated in order to determine the optimal number of steps. This is found to be $n = 10^6$.

## I Introduction

In this project, we want to solve the one-dimensional Poisson's equation with Dirichlet boundary conditions by use of decomposition, finding a general and special case of the Thomas algorithm and creating a tridiagonal solver. The Poisson's equation from electromagnetism reads

$$\nabla^2 \Phi = -4\pi\rho(\mathbf{r}) \tag{1}$$

where $\Phi$ is the electrostatic potential, and $\rho(r)$ is the electric charge density as a function of radius. This equation can be rewritten, by assuming spherical symmetry. Using substitution, saying that $\Phi(r) = \phi/r$ and letting $\phi \to u$ and $r \to x$, we obtain

$$-u''(x) = f(x). \tag{2}$$

We want to solve this equation numerically, and look at the maximum relative error compared to the analytical solution, in order to later determine the optimal number of steps. The differential equation has a closed-form solution given by

$$u(x) = 1 - (1 - e{-}10)x - e^{-10x}. \tag{3}$$

In addition, we want to look at the number of floating point operations (FLOPs) for the general and special case and the LU decomposition by use of the Armadillo library, compare their CPU times and investigate the relation between their CPU times and number of FLOPs.

# II Method

The goal is to solve the equation

$$-u''(x) = f(x) \tag{4}$$

for $x \in (0,1)$ with boundaries $u(0) = u(1) = 0$. The source term used is

$$f(x) = 100e^{-10x}. \tag{5}$$

The differential equation (4) has a closed-form analytical solution given by

$$u(x) = 1 - (1 - e{-}10)x - e^{-10x}. \tag{6}$$

The first step is to discretize $u$ as $v_i$ and the variable $x_i = ih$ where $h$ is the step size defined as $h = 1/(n+1)$. In addition the boundary conditions for $v$ is that $v_0 = v_{n+1} = 0$.

The approximation of the second derivative looks like

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i, \tag{7}$$

for $i = 1, 2, ..., n$. We then rewrite this as

$$-v_{i+1} + 2v_i - v_{i-1} = h^2 f_i. \tag{8}$$

We can then write this set of equations as a matrix

$$A = \begin{bmatrix} 2v_1 & -1v_0 & 0 & ... & ... & 0 & h^2 f_1 \\ -1v_3 & 2v_2 & -1v_1 & 0 & ... & ... & h^2 f_2 \\ 0 & -1v_4 & 2v_3 & -1v_2 & 0 & ... & h^2 f_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & ... & 0 & -1v_n & 2v_{n-1} & -1v_{n-1} & h^2 f_{n-1} \\ 0 & ... & ... & 0 & -1v_{n+1} & 2v_n & h^2 f_n \end{bmatrix} \tag{9}$$

which can be written as a set of linear equations on the form

$$\mathbf{Av} = \tilde{\mathbf{b}}, \tag{10}$$

where

$$A = \begin{bmatrix} 2 & -1 & 0 & ... & ... & 0 \\ -1 & 2 & -1 & 0 & ... & ... \\ 0 & -1 & 2 & -1 & 0 & ... \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & ... & 0 & -1 & 2 & -1 \\ 0 & ... & ... & 0 & -1 & 2 \end{bmatrix}, \tag{11}$$

$$\mathbf{v} = [v_1, v_2, ..., v_{n-1}, v_n] \tag{12}$$

and

$$\tilde{\mathbf{b}} = [h^2 f_1,\ h^2 f_2,\ ...,\ h^2 f_{n-1},\ h^2 f_n]. \tag{13}$$

A is a tridiagonal matrix meaning all the elements, except for the diagonal and the ones immediately above and below the leading diagonal, is zero.

## A General case

The Thomas algorithm is a general algorithm for solving Eq.(7) numerically, by use of forward and backward substitution. When performing these calculations, we do not assume that we have a matrix with the same elements along the diagonal and the non-diagonal elements. We do not need to include the endpoints because we already know that $v_0 = v_{n+1} = 0$. We therefore rewrite matrix A in terms of one-dimensional vectors, with a length $n-1$, called $a$, $b$, $c$. We then get that $\mathbf{Av} = \tilde{\mathbf{b}}$ is

$$\begin{bmatrix} b_1 & c_1 & 0 & ... & ... & 0 \\ a_1 & b_2 & c_2 & 0 & ... & ... \\ 0 & a_2 & b_3 & c_3 & 0 & ... \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & ... & 0 & a_{n-2} & b_{n-1} & c_{n-1} \\ 0 & ... & ... & 0 & a_{n-1} & b_n \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} \tilde{b_1} \\ \tilde{b_2} \\ \vdots \\ \vdots \\ \tilde{b_n} \end{bmatrix} \tag{14}$$

Next, we need to perform a forward substitution. This can be done using Gaussian elimination, taking the element in the $i$'th row and subtracting the element directly above multiplied by $a_i/b'_i$, where $b'_i$ is defined in (16). We then get

$$\begin{bmatrix} b'_1 & c_1 & 0 & ... & ... & 0 \\ 0 & b'_2 & c_2 & 0 & ... & ... \\ 0 & 0 & b'_3 & c_3 & 0 & ... \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & ... & 0 & 0 & b'_{n-1} & c_{n-1} \\ 0 & ... & ... & 0 & 0 & b'_n \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} \tilde{b'_1} \\ \tilde{b'_2} \\ \vdots \\ \vdots \\ \tilde{b'_n} \end{bmatrix} \tag{15}$$

The primed variables are defined as

$$b'_i = b_i - \frac{c_{i-1}a_{i-1}}{b'_{i-1}} \quad \text{and} \quad \tilde{b'_i} = \tilde{b_i} - \frac{\tilde{b}_{i-1}a_{i-1}}{b'_{i-1}}, \tag{16}$$

where $b'_i$ is the modified diagonal element and $\tilde{b'_i}$ is the modified solution, with $b'_1 = b_1$ and $\tilde{b'_1} = \tilde{b_1}$.

We also need to apply backward substitution. Looking at the matrix (15) and working our way backwards, we get

$$v_i = \frac{\tilde{b}'_i}{b'_i} \quad \text{and} \quad v_{i-1} = \frac{\tilde{b}'_{i-1} - c_{i-1}v_i}{b'_{i-1}} \tag{17}$$

## B Special case

Now that we have a general algorithm for solving Eq.(7), we want to specialize it, and use the fact that the matrix has identical matrix elements along the diagonal set to 2, and the value -1 for the non-diagonal elements. Using the same approach as before, we find that the expression for the modified diagonal element $b'_i$ and solution $\tilde{b}'_i$ for the specialized case becomes

$$b'_i = b_i - \frac{1}{b'_{i-1}} \quad \text{and} \quad \tilde{b}'_i = \tilde{b}_i - \frac{\tilde{b}'_{i-1}}{b'_{i-1}}. \tag{18}$$

We can simplify these expressions even more. $b'_i$ has an analytic expression and by inserting this into the expression for $\tilde{b}'_i$, we get

$$b'_i = \frac{i-1}{i} \quad \text{and} \quad \tilde{b}'_i = \tilde{b}_i + \frac{\tilde{b'}_{i-1}(i-1)}{(i-2)}. \tag{19}$$

When looking at the expression for $v_{i-1}$ in (17), changing it to an expression for $v_i$, using that $c_i = -1$, and inserting the expression for $b'_i$ in (18), we obtain

$$v_i = \frac{(\tilde{b'}_i + v_{i+1})i}{i-1}. \tag{20}$$

## C Final equations

All the calculations has now been performed, and we have general algorithms and algorithms for our special case:

General algorithms: (21)

$$\text{Forward} — \quad b'_i = b_i - \frac{c_{i-1}a_{i-1}}{b'_{i-1}} \quad \text{and} \quad \tilde{b'_i} = \tilde{b_i} - \frac{\tilde{b}_{i-1}a_{i-1}}{b'_{i-1}}, \tag{22}$$

$$\text{Backward} — \quad v_{i-1} = \frac{\tilde{b}'_{i-1} - c_{i-1}v_i}{b'_{i-1}}, \tag{23}$$

Special algorithms: (24)

$$\text{Forward} — \quad \tilde{b'_i} = \tilde{b_i} + \frac{\tilde{b'}_{i-1}(i-1)}{(i-2)}, \tag{25}$$

$$\text{Backward} — \quad v_i = \frac{(\tilde{b'}_i + v_{i+1})i}{i-1}, \tag{26}$$

where the special algorithms has 4n floating point operations (FLOPs) in total, 2n FLOPs each, and the general algorithms has 9n FLOPs in total, 3n each.

Now that we have the equations and the number of floating point operations, we want to code these algorithms for $n \times n$ matrices of different sizes. We want to do this so that we later can create plots and compare the results. We also want to calculate the CPU time for both the general and specialized algorithms for matrices up to $n = 10^6$ points, so that we also can compare these results.

## D LU decomposition

We want to solve our algorithms by use of the armadillo function for the LU decomposition. This we want to do in order to compare the time usage between this decomposition and our tridiagonal solver. The number of FLOPs for an LU decomposition goes as $(2/3)n^3$.

## E Relative error

To evaluate the numerical accuracy of the specialised algorithm the relative error in the dataset is calculated using

$$\varepsilon_i = \log_{10}\left(\left|\frac{v_i - u_i}{u_i}\right|\right), \tag{27}$$

for $i = 1, 2, ..., n$, where $\varepsilon$ is the relative error, $v$ is the numerical solution and $u$ is the analytic exact solution. The maximum value is then extracted for different values for $n$ up to and including $n = 10^7$. When performing these calculations, we use the special case of the Thomas algorithm described in subsection B. These values are presented in the Results section.

# III Results

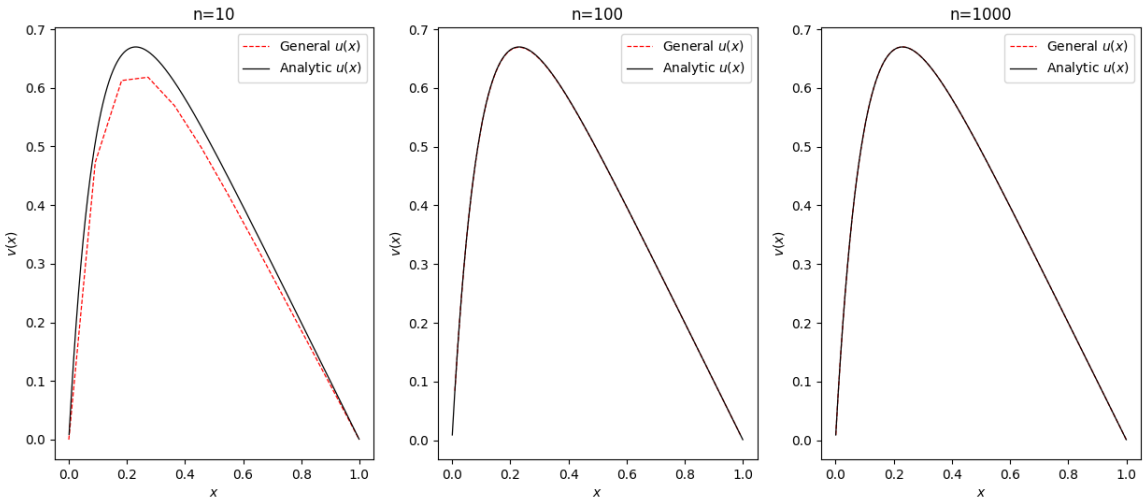


Figure 1: Comparison plot of general algorithm solution and the analytic solution for different number of steps, $n$.

In Figure 1, we have compared the solution from the general algorithm to the analytic solution using $n = 10$, $n = 100$ and $n = 1000$.

Table 1: CPU times in ms for the specialized and general algorithms using varying number of steps

| n | 10 | $10^2$ | $10^3$ | $10^4$ | $10^5$ | $10^6$ |
|---|---|---|---|---|---|---|
| General algo | 0.032 | 0.02 | 0.077 | 1.588 | 6.852 | 65.364 |
| Special algo | 0.003 | 0.016 | 0.061 | 1.435 | 6.16 | 65.114 |
| LU | 0.278 | 1.505 | 99.661 | 44316.5 | | |

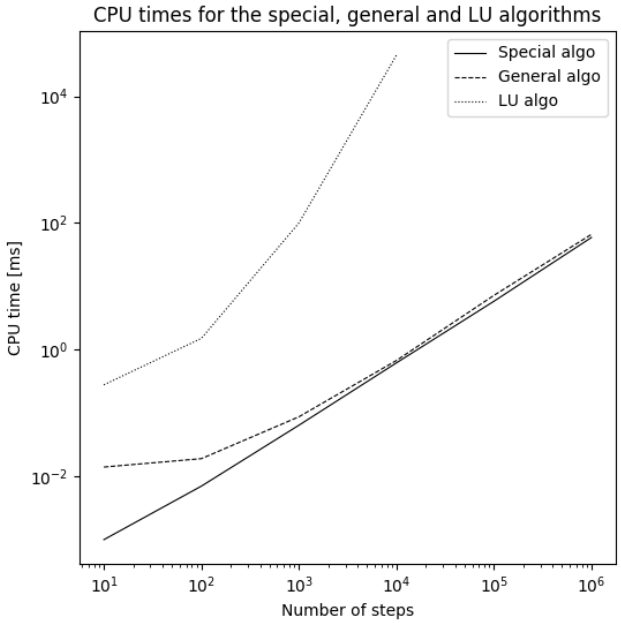


Figure 2: CPU times for the general and special algorithms for different number of steps. Both the $x$ and $y$ axes are logarithmic.

Table 1 lists the CPU times for the general and special algorithms in ms for varying number of steps. Figure 2 illustrates the evolution of CPU times listed in Table 1.

Table 2: Max relative errors ($\varepsilon$) for the special solution as a function of step length, $h$

| n | 10 | $10^2$ | $10^3$ | $10^4$ | $10^5$ | $10^6$ | $10^7$ |
|---|---|---|---|---|---|---|---|
| $\log_{10}(h)$ | -1.04139 | -2.00432 | -3.00043 | -4.00004 | -5 | -6 | -7 |
| $\log_{10}(\varepsilon)$ | -1.1797 | -3.08804 | -5.08005 | -7.007927 | -9.07909 | -10.1636 | -9.08998 |

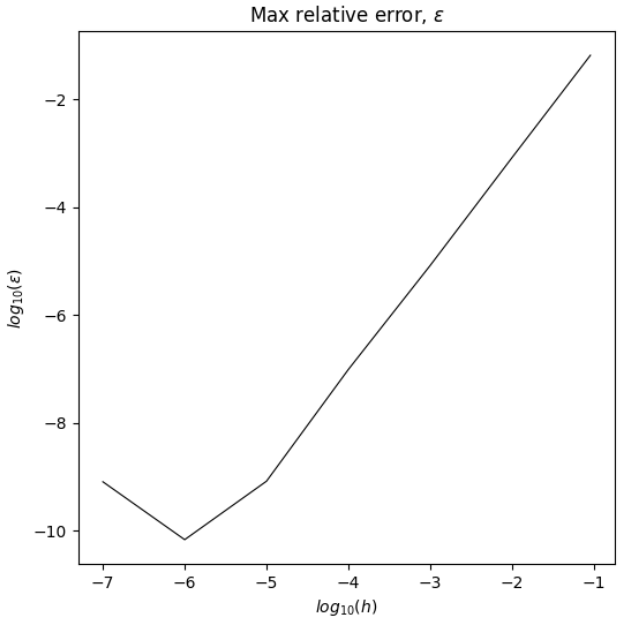


Figure 3: Plot of the logarithm of the max relative errors from the special solution as a function of step size.

The max relative errors from the special solution are calculated using (27), and listed in Table 2. The relative errors are plotted as a function of step length in Figure 3.

# IV Discussion and conclusion

When comparing numerical accuracy for different step lengths, one would expect accuracy to increase with decreasing step length. The results presented in Figure 1 are consistent with this. The gain in accuracy is most visible when comparing the plots for $n = 10$ and $n = 100$, whereas the difference is small to non-visible between $n = 100$ and $n = 1000$.

For our general and special algorithms, the number of FLOPs goes as $n$. The CPU times of these algorithms are illustrated in Figure 2, and show the expected linear relationship with the special algorithm being marginally more effective. The polynomial growth of the LU decomposition CPU time, is consistent with the number if FLOPs going as $n^3$. Because of this growth rate, the algorithm in the Armadillo library is unable to handle matrices of sizes $10^5 \times 10^5$ and larger.

In numerical analysis one would at some point expect numerical inaccuracies to arise. This loss of precision is due to the step length decreasing below a certain threshold. The evolution of the relative error presented in Table 2 and Figure 3 show that our special algorithm reaches this threshold around $\log_{10}(h) = -6$. Based on this we can conclude that the optimal number of steps for this algorithm is $n = 10^6$.