Project 1

Axl H. Kleven
Live Ljungqvist Storborg
Henrik Haug

22. september 2025

https://github.uio.no/heh/FYS3150

Sammendrag

Introduction

Poisson equation

In this article we will be looking at numerical solutions of the one-dimensional Poisson equation, as shown in equation (eq:Poisson)

$$-\frac{d^2u}{dx^2} = f(x) \tag{1}$$

Where u(x) is an unknown function we wish to find, and f(x) is a known function.

We will use the Poisson equation for a function f(x) given by

$$f(x) = 100e^{-10x} \qquad x \in [0,1] \tag{2}$$

And boundary conditions for u(x) given by

$$u(0) = u(1) = 0 \tag{3}$$

The Poisson equation is a second-order differential equation that shows up in several areas of physics, e.g. electrostatics. This makes its scaled numerical solutions useful to study now, as they will be necessary to calculate in later projects.

Theory

Discretization

The definition of the double derivative is given in equation (5)

$$\frac{d^2f}{dx^2} = \lim_{h\to 0}\frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{4}$$

Thomas Algorithm

The Thomas Algorithm is used to solve tridiagonal matrix equations. It is a simplified form of the gaussian elimination and works as follows: Consider an equation on the form

$$A\vec{v} = \vec{g} \tag{5}$$

Where A is a n x n tridiagonal matrix on the shape of

$$A = \begin{pmatrix} b_1 & c_1 & 0 & 0 & \\ a_2 & b_2 & c_2 & 0 & \\ 0 & a_3 & b_3 & c_3 & ... \\ 0 & 0 & a_4 & b_4 & \\ & & ... & & \end{pmatrix}$$

And $\vec{v}, \vec{g}$ are vectors on the form

$$\vec{v} = \begin{pmatrix} v_1 \\ v_2 \\ ... \\ v_n \end{pmatrix}, \vec{g} = \begin{pmatrix} g_1 \\ g_2 \\ ... \\ g_n \end{pmatrix}$$

The first step is to forward substitute so that the matrix A is on the shape

$$A = \begin{pmatrix} \tilde{b}_1 & c_1 & 0 & 0 & \\ 0 & \tilde{b}_2 & c_2 & 0 & \\ 0 & 0 & \tilde{b}_3 & c_3 & ... \\ 0 & 0 & 0 & \tilde{b}_4 & \\ & & ... & & \end{pmatrix}$$

We do this by following a set of steps:

$$\tilde{b}_1 = b_1$$

$$\tilde{b}_i = b_i - \frac{a_i}{\tilde{b}_{i-1}} c_{i-1} \qquad \text{i = 2, 3, ..., n}$$

$$\tilde{g}_1 = g_1$$

$$\tilde{g}_i = g_i - \frac{a_i}{\tilde{b}_{i-1}} g_{i-1} \qquad \text{i = 2, 3, ..., n}$$

And the second step is to backwards substitute to find the values in $\vec{v}$, again following a set of steps:

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

$$v_i = \frac{\tilde{g}_i - c_i v_{i+1}}{\tilde{b}_i} \qquad \text{i = n-1, n-2, ..., 1}$$

Errors

The logarithm of the absolute error is shown in equation (6).

$$\log_{10}(\Delta_i) = \log_{10}(|u_i - v_i|) \tag{6}$$

The logarithm of the relative error is shown in equation (7).

$$\log_{10}(\epsilon_i) = \log_{10}\left(\frac{u_i - v_i}{u_i}\right) \tag{7}$$

Methods

1

Our first task is to analytically prove that

$$u(x) = 1 - \left(1 - e^{-10}\right) x - e^{-10x} \tag{8}$$

Is a viable solution for the one-dimensional Poisson equation (1). We start by checking the boundary conditions for u, as shown in equation (3).

$$u(0) = 1 - (1 - e^0) * 0 - e^0 = 1 - 0 - 1 = 0$$

$$u(1) = 1 - (1 - e^{-10}) - e^{-10} = 1 - 1 + e^{-10} - e^{-10} = 0$$

We now insert u(x) into the Poisson equation, and expect the left side to equal the right side (f(x)).

$$-\frac{d^2}{dx^2}(1-(1-e^{-10})x-e^{-10x}) = f(x)$$

$$\begin{aligned} f(x) &= -\frac{d}{dx}(e^{-10}-1+10e^{-10x}) \\ &= 100e^{-10x} \end{aligned}$$

Which is as expected, and u(x) is a solution to the one-dimensional Poisson equation.

2

For making a figure of the exact solution given in problem 1, we made a script implementing the function u(x) as seen in equation (8). The script calculates the u(x) values for the given x, and outputs u(x) and its respective x to a file we later read to create the figure. We use scientific notation with 8 decimals.

3

The Poisson equation (1) is defined for a continuous spectrum of x. For numerical use we need to discretize the equation. This will be done using the definition of the double derivative (4)

$$\frac{d^2f}{dx^2} = \lim_{h\to 0}\frac{f(x+h)-2f(x)+f(x-h)}{h^2}$$

Where, when discretizing, h will be the step size, meaning we can change our notation to

$$f(x+h) = f_{i+1}$$
$$f(x) = f_i$$
$$f_(x-h) = f_{i-1}$$

Where i is the step in the discrete range of x we are currently at.
Applying this to the Poisson equation:

$$-\frac{d^2u(x)}{dx^2} = f(x)$$

$$-\left(\frac{v_{i+1}-2v_i+v_{i-1}}{h^2}\right)=f_i$$

Where v(x) is our discretized approximation to u(x)

$$-v_{i+1}+2v_i-v_{i-1}=h^2f_i \tag{9}$$

4

We see that we can write our discretized approximation for u(x) (9) as a matrix equation for all i on the form

$$A\vec{v}=\vec{g}$$

We use N points in x, giving n = N-1 possible steps for i. We start by writing the equation out for increasing i, using $g_i \equiv h^2 f_i$:

$$g_1 \equiv h^2 f_1 = -v_0 + 2v_1 - v_2$$

$$g_2 \equiv h^2 f_2 = 0 - v_1 + 2v_2 - v_3$$

$$g_3 \equiv h^2 f_3 = 0 + 0 - v_2 + 2v_3 - v_4$$

$$...$$

$$g_{n-1} \equiv h^2 f_{n-1} = 0 + 0 + ... + 0 - v_{n-2} + 2v_{n-1} - v_n$$

Where we, because of the boundary conditions, know v_0 and v_n, meaning we can write

$$g_1 \equiv h^2 f_1 + v_0$$

$$g_{n-1} \equiv h^2 f_{n-1} + v_n$$

And thus can rewrite this system of equations to a matrix equation

$$\begin{pmatrix} 0 & 2 & -1 & 0 & 0 & ... & 0 \\ 0 & -1 & 2 & -1 & 0 & ... & 0 \\ 0 & 0 & -1 & 2 & -1 & ... & 0 \\ & & & ... & & & \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ ... \\ v_{n-1} \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ ... \\ g_{n-1} \end{pmatrix}$$

Where we can simply remove the first row of 0's in the matrix, and we have our matrix equation

$$A\vec{v}=\vec{g}$$

Where A will be of dimension (n-1) x (n-1)

5

a)
As we have seen, when there are N values in x, we will have n = N - 1 possible steps i, and A will be of dimensions (n-1) x (n-1).
If we rename the dimensions of A to a x a, we have the relation

$$a = n - 1 = N - 1 - 1 = N - 2 \tag{10}$$

b) From the last relation (10), we can tell that there will be missing two parts of the solution in $\vec{v}^*$. From our derivation in 4, we can see that these two parts will be the known initial values at the start and end of $\vec{v}^*$.

6

a)
We can apply the Thomas Algorithm to solve our tridiagonal matrix equation. The Thomas algorithm is described in the theory section.

b)
To count the amount of FLOPs in the Thomas Algorithm, we must look at all the mathematical operations in the steps. First step:

$$\tilde{b}_1 = b_1$$

0 FLOPs

$$\tilde{b}_i = b_i - \frac{a_i}{b_{i-1}} c_{i-1} \qquad \text{i = 2, 3, ..., n}$$

3n - 3(Remembering the case where i = 1) FLOPs

$$\tilde{g}_1 = g_1$$

0 FLOPs

$$\tilde{g}_i = g_i - \frac{a_i}{b_{i-1}} g_{i-1} \qquad \text{i = 2, 3, ..., n}$$

3n - 3 FLOPs

A total of 6n - 6 FLOPs in the first step. Moving on to the second step:

$$v_n = \frac{\tilde{g}_n}{\tilde{v}_n}$$

1 FLOPs

$$v_i = \frac{\tilde{g}_i - c_i v_{i+1}}{\tilde{v}_i} \qquad \text{i = n-1, n-2, ..., 1}$$

3n - 3 FLOPs

A total of 3n - 2 FLOPs in the second step. Summing up to a total of

$$9n - 8\text{FLOPs}$$

If we're a bit clever, we can define

$$m_i = \frac{a_i}{b_{i-1}}$$

In the first step, and using this calculating $\tilde{b}_i$ and $\tilde{g}_i$, saving a FLOP per iteration! Meaning our total is now

$$8(n-1)\text{FLOPs}$$

7

a)
For solving the matrix equation as seen in equation (5) we implement the Thomas algorithm as described in the theory section, and outputs the $\vec{v}$ values and its respective x to a file.

b)
For making a figure of the comparison of numerical- and exact solution we read through output file as mentioned above.

8

a)
For looking at error computations we make a figure showing the absolute error using equation (6). We use the logarithm because we're dealing with an exponential function, and the logarithm could reveal information about the exponent.

We ignore the boundaries (u(0) = u(1) = 0), seeing as these have 0 error.

b)

Similarly, we make a figure showing the relative error using equation (7).

We choose to make this figure as well, seeing as understanding the errors of our approximation is important for understanding its potential use.

c)
We are writing a table with two columns showing the maximum relative error for each choice of n_{step} by iterating over a container where we store all the maximum values.

9

a)
Specializing our algorithm is done by using the script from task 7, and filling in the values for a, b, and c.

b)
We used the same methode as in problem 6 b), and got the same number of FLOPs for the special algorithm as for the general algorithm.

c)
We use and modify the script from problem 7.

10

Timing tests were done by repeatedly measuring the time necessary to complete the general-, and the special algorithm for different values of n_{step}. We made a table with three columns representing n_{steps}, time for computing general algorithm and time for computing special algorithm, respectively.

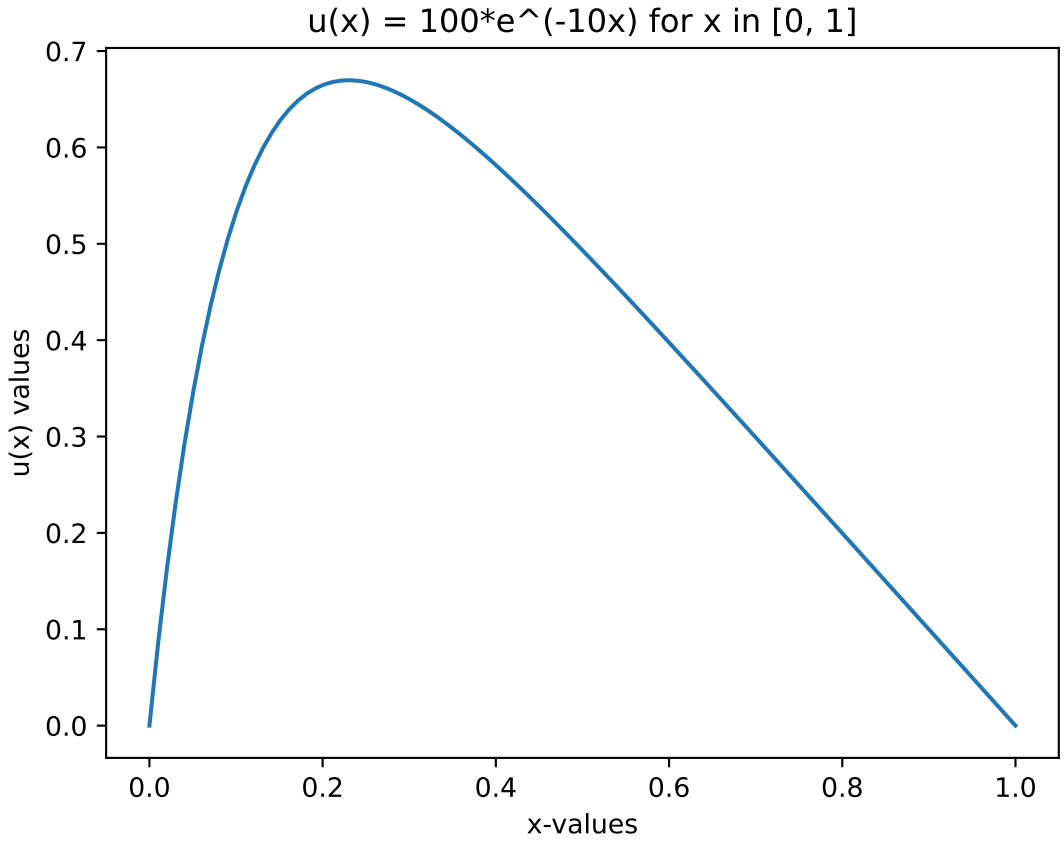
Results

1

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

Is a viable solution to the one-dimensional Poisson equation (1).

2



Figur 1: The exact solution of $u(x) = 100e^{-10x}$ for $x \in [0, 1]$.

3

Our discretized version of the Poission equation is given by

$$-v_{i+1} + 2v_i - v_{i-1} = h^2 f_i$$

Where v(x) is our discretized approximation to u(x). And the subscript i is the current step in the discrete range of x.

4

We can rewrite our discretized equation as a matrix equation on the form

$$A\vec{v} = \vec{g}$$

Where A is a tridiagonal matrix with the subdiagonal, main diagonal, and superdiagonal specified by (-1, 2, -1). For N points in x, we have a total of n = N-1 steps (i), such that A, $\vec{v}$, and $\vec{g}$ take the form

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 & ... & 0 \\ -1 & 2 & -1 & 0 & 0 & ... & 0 \\ 0 & -1 & 2 & -1 & 0 & ... & 0 \\ 0 & 0 & -1 & 2 & -1 & ... & 0 \\ ... & & & & & & \end{pmatrix}$$

Where A will be of dimension (n-1) x (n-1)

$$\vec{v} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ ... \\ v_{n-1} \end{pmatrix}, \vec{g} = \begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ ... \\ g_{n-1} \end{pmatrix}$$

Where

$$g_i \equiv h^2 f_i$$

And

$$g_1 \equiv h^2 f_1 + v_0$$
$$g_{n-1} \equiv h^2 f_{n-1} + v_n$$

5

a)
If $\vec{v}^*$ is a vector of length m representing a complete solution of the discrete Poisson equation, with corresponding x values given by a length m vector $\vec{x}$. And A is a matrix of dimension n x n, we have the relation between n and m

$$n = m - 2$$

b)
When solving the matrix equation we find v_i for i between 1 and m-1. v_0 and v_m is already known from initial values.

6

a)
For solving a matrix equation on the form

$$A\vec{v} = \vec{g}$$

Where A is a tridiagonal matrix, we apply the Thomas algorithm

b)
If we define

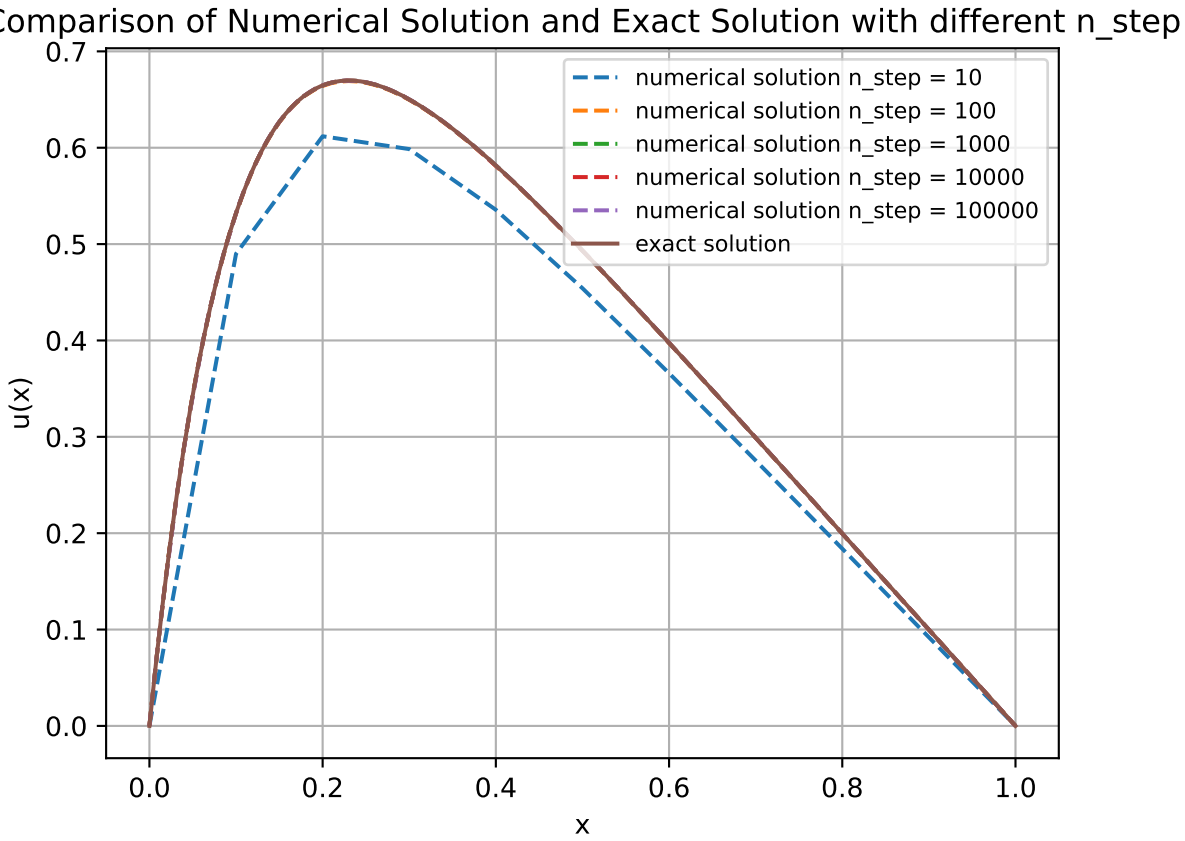
$$m_i = \frac{a_i}{b_{i-1}}$$

In the first step, and use this to calculate $\tilde{b}_i$ and $\tilde{g}_i$, there are a total of

$$8(n-1)$$

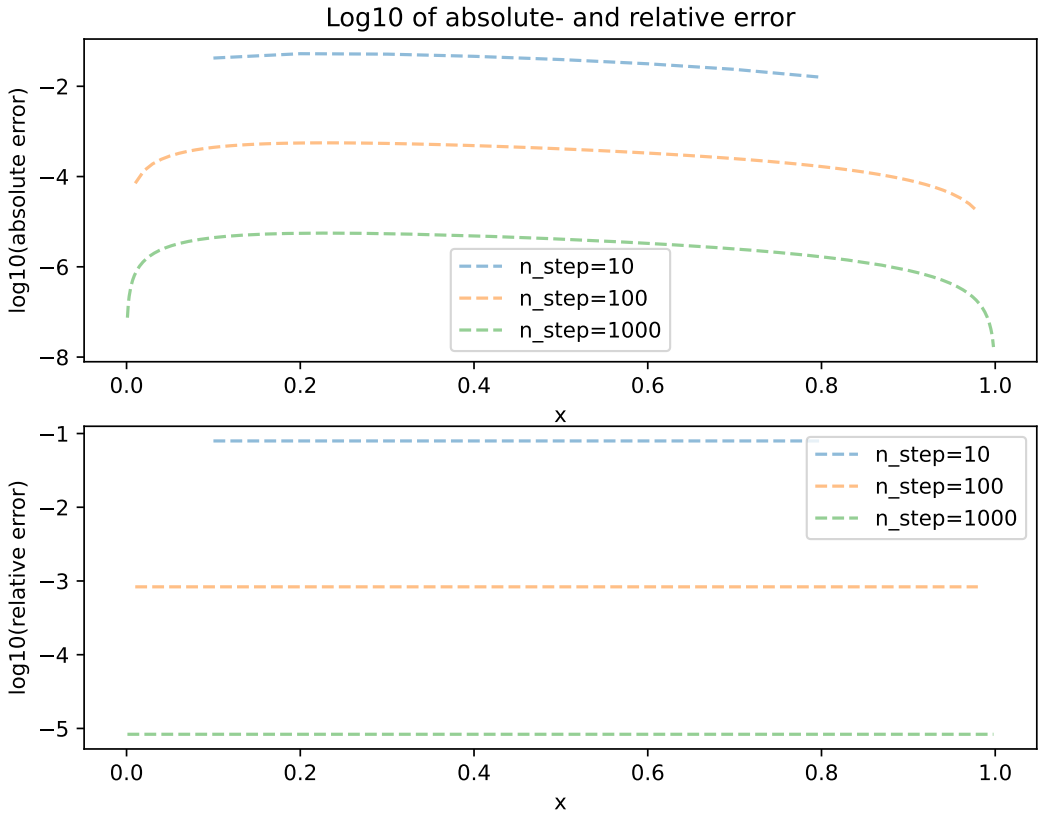
FLOPs in the Thomas Algorithm.

7



Figur 2: Comparing the exact solution of $u(x)$ represented by the red continuous line, and the numerical solution for $n_{step} = 10^1, ..., 10^5$ represented by dashed lines.

8



Figur 3: Simulating the logarithm of the absolute error in the figure above, and the relative error in the figure below, for different choices of n_{step}, against x. The boundary points $u(0) = u(1) = 0$ are not included.

c)

By looking at the results we can see that the relative error drastically decreases for n_{step} smaller than 10^5. We have the smallest relative error for $n_{step} = 10^6$. For larger n_{step}, the relative error increases.

9

b)

It is reasonable that we get the same number of FLOPs for both algorithms, because the computation is the same.

The difference in computation time depends on the number of loads and stores from memory to the processing unit. For the general algorithm, there are way more loads since the computation depends on values in the superdiagonal, diagonal and subdiagonal. For the special algorithm, the values of the diagonals are fixed, and there are no loads from memory associated with these vectors.

10

Tabell 1: Table containing time comparison between the general and special algorithm at n_{step}.

n_{step}	Time general (s)	Time special (s)
10	$5e-06$	$1e-06$
100	$9e-06$	$7e-06$
1000	$5.2e-05$	$4.3e-05$
10 000	0.000532	0.000417
100 000	0.004989	0.004623
1 000 000	0.058602	0.049769

Discussion

Conclusion

References