Project 1 FYS3150

Anders P. Åsbø, Eivind Støland

CONTENTS

I. INTRODUCTION

One of the most versatile tools in modern science is numerical integration, thus it it simportant to understand its limits. In this paper we have performed numerical integration of a second order differential equation. This was done by discretizing the differential equation, and formulating it as a matrix-vector equation. The matrix-vector equation was then solved using both a general, and specialized Thomas algorithm, as well as LU-decomposition. Under the pretext of solving a 1D version of Poisson's equation, we have created and tested a numerical solver using the Thomas algorithm. Furthermore, we have compared our results with LU decomposition using the Armadillo library's solver. We measured the time spent by the various algorithms and their maximum relative error (to the analytic solution) in order to compare them quantitavely to see if they behave as expected.

II. FORMALISM

Poisson's equation is a well known equation from electromagnetism:

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

where $\boldsymbol{r}$ is the position, $\rho(\boldsymbol{r})$ is the charge density, and $\mathbf{\Phi}(\boldsymbol{r})$ is the electrostatic potential. If we assume spherical symmetry, this simplifies into a one-dimensional equation:

$$\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\mathbf{\Phi}}{dr}\right) = -4\pi\rho\,,$$

where $r = |\boldsymbol{r}|$. We can substitute $\mathbf{\Phi}(r) = \phi(r)/r$, which gives us:

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

This is a second order differential equation which can be written generally as:

$$-u''(x) = f(x)\,,$$

where we have changed $r \to x$, $\phi \to u$ and $4\pi r\rho \to f(x)$. In order to proceed we need to pick a specific sample problem to be used. We choose to solve this equation with Dirichlet boundary conditions, meaning that $x \in (0,1)$ and $u(0) = u(1) = 0$. As for the source term f we choose $f(x) = 100e^{-10x}$. This has the advantage of being possible to solve analytically:

$$\begin{aligned}
-u''(x) &= f(x)\\
-u''(x) &= 100e^{-10x}\\
-u'(x) &= -10e^{-10x} - C\\
-u(x) &= e^{-10x} - Cx - D\\
u(x) &= Cx + D - e^{-10x}\,,
\end{aligned}$$

where C and D are arbitrary constants. We use the boundary conditions in order to determine them:

$$\begin{aligned}
u(0) &= 0\\
\implies D - 1 &= 0\\
D &= 1\\
u(1) &= 0\\
\implies C + 1 - e^{-10} &= 0\\
C &= -(1 - e^{-10})
\end{aligned}$$

All in all this gives us that:

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

Now we have an analytical solution we can compare our numerical solutions to later.

In order to solve the problem numerically we first need some definitions in order. We discretize with the grid points given by $x_i = ih$, in the interval from $x_0 = 0$ to $x_{n+1} = 1$. The step length is then given by $h = 1/(n+1)$. We name the discretized approximation to the solution v_i. We can then approximate the second derivative of u as:

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

where $f_i = f(x_i)$ and $i = 1, ..., n$. We look at some terms separately in order to find the matrix-vector form of this problem. First we look at the expression at the boundaries, starting with $i = 1$:

$$\begin{aligned} -\frac{v_2 + v_0 - 2v_1}{h^2} &= f_1 \\ 2v_1 - v_2 &= h^2 f_1 \\ 2v_1 - v_2 &= \tilde{b}_1 \,, \end{aligned}$$

where we have defined a new variable $\tilde{b}_i = h^2 f_i$ and applied the relevant boundary condition. We then look at the expression when $i = n$:

$$\begin{aligned} -\frac{v_{n+1} + v_{n-1} - 2v_n}{h^2} &= f_n \\ -v_{n-1} + 2v_n &= \tilde{b}_n \,, \end{aligned}$$

where we also applied the relevant boundary condition. The general expression can also be rewritten as:

$$-v_{i-1} + 2v_i - v_{i+1} = \tilde{b}_i$$

This now clearly takes the shape of a matrix-vector problem. We define a vector $\mathbf{v}$ which contains all the v_i and similarly a vector $\tilde{\mathbf{b}}$ which contains all the b_i. The coefficients on the left-hand side of the equations determine a $n \times n$-matrix $\mathbf{A}$:

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

The matrix-vector problem we need to solve then is:

$$\mathbf{A}\mathbf{v} = \tilde{\mathbf{b}}$$

$\mathbf{A}$ is a tridiagonal matrix. This means we can apply methods specialized for this kind of linear algebra problem. A tridiagonal matrix can be decomposed into three vectors, one for the diagonal and one each for the bands above and below the diagonal. We choose the vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$. Their components are defined the following way:

$$\mathbf{A} = \begin{bmatrix} b_1 & c_1 & 0 & ... & ... & 0 \\ a_2 & b_2 & c_2 & 0 & ... & ... \\ 0 & a_3 & b_3 & c_3 & 0 & ... \\ ... & ... & ... & ... & ... & ... \\ 0 & ... & ... & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & ... & ... & 0 & a_n & b_n \end{bmatrix}$$

As the bands above and below the diagonal have one element less than the diagonal itself, we note that $\mathbf{a}$ and $\mathbf{c}$ both have $n-1$ elements defined this way instead of n elements (as $\mathbf{b}$). We choose, however, to let $\mathbf{a}$'s first component to be denoted a_2, as the indexing then matches those along the same row of the matrix $\mathbf{A}$.

In general this gives us equations on the form:

$$a_i v_{i-1} + b_i v_i + c_i v_{i+1} = \tilde{b}_i \,,$$

which we need to solve. As there is not element c_n or a_1, we simply define them to be 0 instead, and thus the equation is valid from $i = 1, ..., n$. Looking at the tridiagonal matrix $\mathbf{A}$ with general elements again, we can see that we can eliminate the band below the diagonal (the components of $\mathbf{a}$). First we take the first row multiplied by a_2/b_1 and subtract it from row 2. This leaves us with the following matrix:

$$\mathbf{A} = \begin{bmatrix} b_1 & c_1 & 0 & ... & ... & 0 \\ 0 & b_2 - \frac{a_2 c_1}{b_1} & c_2 & 0 & ... & ... \\ 0 & a_3 & b_3 & c_3 & 0 & ... \\ ... & ... & ... & ... & ... & ... \\ 0 & ... & ... & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & ... & ... & 0 & a_n & b_n \end{bmatrix}$$

We now define $\tilde{d}_2 = b_2 - \frac{a_2 c_1}{b_1}$, and $\tilde{d}_1$ in order to simplify further equations:

$$\mathbf{A} = \begin{bmatrix} \tilde{d}_1 & c_1 & 0 & ... & ... & 0 \\ 0 & \tilde{d}_2 & c_2 & 0 & ... & ... \\ 0 & a_3 & b_3 & c_3 & 0 & ... \\ ... & ... & ... & ... & ... & ... \\ 0 & ... & ... & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & ... & ... & 0 & a_n & b_n \end{bmatrix}$$

We can now similarly eliminate a_3 by multiplying row 2 with $\frac{a_3}{\tilde{d}_2}$ and subtract this from row 3. This gives us the following:

$$\mathbf{A} = \begin{bmatrix} \tilde{d}_1 & c_1 & 0 & ... & ... & 0 \\ 0 & \tilde{d}_2 & c_2 & 0 & ... & ... \\ 0 & 0 & b_3 - \frac{a_3 c_2}{\tilde{d}_2} & c_3 & 0 & ... \\ ... & ... & ... & ... & ... & ... \\ 0 & ... & ... & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & ... & ... & 0 & a_n & b_n \end{bmatrix}$$

We now define $\tilde{d}_3 = b_3 - \frac{a_3 c_2}{\tilde{d}_2}$. This process can be repeated until all the elements in the band below the diagonal are eliminated. The new diagonal elements ($\tilde{d}_i$) are given by the following formula:

$$\tilde{d}_i = b_i - \frac{a_i c_{i-1}}{\tilde{d}_{i-1}} \quad , \quad i = 2, ..., n \,,$$

with $\tilde{d}_1 = b_1$. This leaves us with a new matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} \tilde{d}_1 & c_1 & 0 & ... & ... & 0 \\ 0 & \tilde{d}_2 & c_2 & 0 & ... & ... \\ 0 & 0 & \tilde{d}_3 & c_3 & 0 & ... \\ ... & ... & ... & ... & ... & ... \\ 0 & ... & ... & 0 & \tilde{d}_{n-1} & c_{n-1} \\ 0 & ... & ... & 0 & 0 & \tilde{d}_n \end{bmatrix}$$

Now in order for us to use this to solve the problem, it also needs to be applied to $\tilde{\mathbf{b}}$ simultaneously, resulting in a new vector $\tilde{\mathbf{f}}$ with elements given as:

$$\tilde{f}_i = \tilde{b}_i - \frac{a_i \tilde{f}_{i-1}}{\tilde{d}_{i-1}} \quad , \quad i = 2, ..., n \,,$$

with $\tilde{f}_1 = \tilde{b}_1$. This gives us equations on the following form that we need to solve:

$$\tilde{d}_i v_i + c_i v_{i+1} = \tilde{f}_i \quad , \quad i = 1, ..., n$$

We note that $c_n = 0$, which means that when $i = n$, we have that:

$$\begin{aligned} \tilde{d}_n v_n &= \tilde{f}_n \\ v_n &= \frac{\tilde{f}_n}{\tilde{d}_n} \end{aligned}$$

Now that we know one value $\mathbf{v}$, this will now uniquely determine the other elements. We can see this by rewriting:

$$\begin{aligned} \tilde{d}_i v_i + c_i v_{i+1} &= \tilde{f}_i \\ v_i &= \frac{\tilde{f}_i - c_i v_{i+1}}{\tilde{d}_i} \end{aligned}$$

As long as we know the last element in $\mathbf{v}$ (which we do at this point) this relation allows us to find v_i for $i = n-1, ..., 1$. As we run through the elements from the largest i to the smallest we call this part the backwards substitution part. For similar reasons we call calculating the $\tilde{d}_i$ and $\tilde{f}_i$ the forwards substitution part. This algorithm is called the Thomas algorithm [1] and is used for solving a general tridiagonal matrix. In our case however, we know what the elements of $\mathbf{A}$ are, allowing us to form a special algorithm. First we look at the $\tilde{d}_i$:

$$\tilde{d}_i = b_i - \frac{a_i c_{i-1}}{\tilde{d}_{i-1}}$$

We recognize that $b_i = 2$, and $a_i = c_{i-1} = i - 1$. This simplifies the equation:

$$\begin{aligned} \tilde{d}_i &= 2 - \frac{1}{\tilde{d}_{i-1}} \\ &= \frac{i+1}{i} \end{aligned}$$

We can simplify $\tilde{f}_i$ in a similar way:

$$\begin{aligned} \tilde{f}_i &= \tilde{b}_i - \frac{a_i \tilde{f}_{i-1}}{\tilde{d}_{i-1}} \\ \tilde{f}_i &= \tilde{b}_i + \frac{i-1}{i} \tilde{f}_{i-1} \end{aligned}$$

This also gives us a simplification for v_i:

$$\begin{aligned} v_i &= \frac{\tilde{f}_i - c_i v_{i+1}}{\tilde{d}_i} \\ &= \frac{i}{i+1} (\tilde{f}_i + v_{i+1}) \end{aligned}$$

This can be optimized somewhat, which will be discussed in section III. Further details on the topics discussed in this section and on LU decomposition can be found in [2].

III. IMPLEMENTATION

IV. ANALYSIS

A. Plots of the general Thomas algorithm

We ran the Thomas algorithm with $N = 10$, $N = 10^2$, and $N = 10^3$ steps and created plots of the resulting data, and compared this with the analytic solution. These plots can be found in figures 1, 2 and 3.

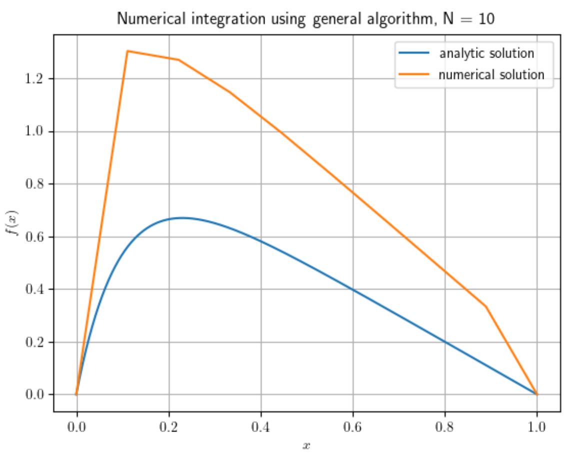

Figure 1. Plot of numerical and analytical solution, using the general Thomas algorithm with $N = 10^1$.

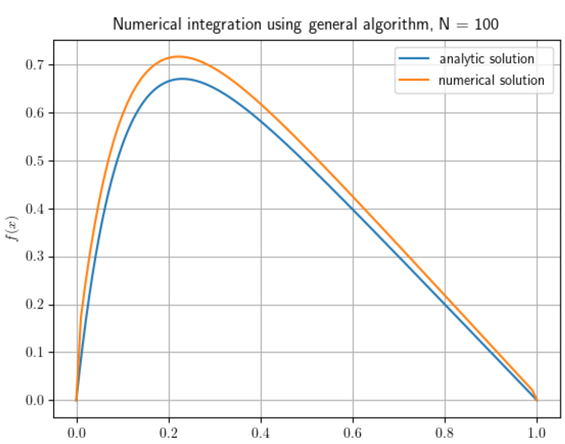

Figure 2. Plot of numerical and analytical solution, using the general Thomas algorithm with $N = 10^2$.

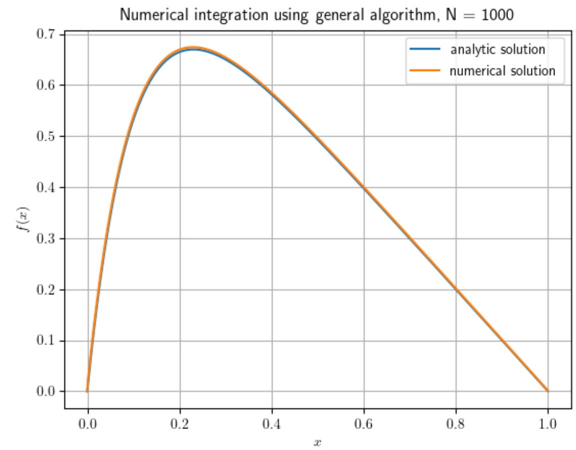

Figure 3. Plot of numerical and analytical solution, using the general Thomas algorithm with $N = 10^3$.

As expected we see that for larger N (shorter step length) the numerical approximation is closer to the analytic one, which also suggests that the algorithm is working as intended.

B. Relative errors

Table I. Table with $\log_{10}$ of relative error for general and special Thomas algorithms, LU decomposition, and $\log_{10}$ of step size h. To run the LU decomposition with $N \geq 10^5$ would have required more memory than we have available, and have thus not been performed.

$\log_{10}(h)$:	ϵ General	ϵ Special	ϵ LU	N
$-1.041\,393$	$3.026\,200 \times 10^{-1}$	$3.601\,314 \times 10^{-1}$	$1.052\,42$	10^1
$-2.004\,321$	$3.426\,303 \times 10^{-2}$	$4.249\,885 \times 10^{-2}$	$1.009\,57$	10^2
$-3.000\,434$	$3.474\,750 \times 10^{-3}$	$4.338\,587 \times 10^{-3}$	$3.474\,75 \times 10^{-3}$	10^3
$-4.000\,043$	$3.479\,720 \times 10^{-4}$	$4.347\,831 \times 10^{-4}$	$3.479\,72 \times 10^{-4}$	10^4
$-5.000\,004$	$3.480\,179 \times 10^{-5}$	$4.348\,760 \times 10^{-5}$	-	10^5
$-6.000\,000$	$4.210\,129 \times 10^{-6}$	$4.348\,746 \times 10^{-6}$	-	10^6
$-7.000\,000$	$1.005\,169 \times 10^{-6}$	$4.343\,971 \times 10^{-7}$	-	10^7
$-8.000\,000$	$-1.140\,500 \times 10^{-3}$	$3.765\,295 \times 10^{-8}$	-	10^8

C. Benchmarks

We measured execution time for the three algorithms (for various amounts of steps), in order to compare the time spent by each to see if everything behaves as expected. The results are shown in table II. We expected that the the slowest of the three would be the LU decomposisition, as that is a method used to solve general matrix-vector equations. The general Thomas algorithm should be quicker, as that is dependent on the matrix being tridiagonal, and the special Thomas algorithm should be the fastest of the three, as it is reliant on the elements also having specific values. We can see from the execution times listed in table II that these expectations were correct. The LU decomposition is the slowest, the general Thomas algorithm is quicker, and the special Thomas algorithm is the quickest one, for all amounts of steps used. Note that for $N = 10$ the execution time for the specialized and general algorithm is the same, which is also the same as the execution time for the specialized algorithm with $N = 10^2$ steps. This probably has something to do with precision in the measurement of the execution time, since the time measured is such a small number.

Table II. Table of execution time in seconds for general and special Thomas algorithms, and LU decomposition.

Algorithm	Execution time $[s]$	N
General Thomas	1×10^{-6}	10^1
General Thomas	3×10^{-6}	10^2
General Thomas	1.8×10^{-5}	10^3
General Thomas	0.000 119	10^4
Specialized Thomas	1×10^{-6}	10^1
Specialized Thomas	1×10^{-6}	10^2
Specialized Thomas	7×10^{-6}	10^3
Specialized Thomas	4.6×10^{-5}	10^4
LU decomposition	0.000 309	10^1
LU decomposition	0.000 755	10^2
LU decomposition	0.171 256	10^3
LU decomposition	158.701	10^4

V. CONCLUSION

[1] L. Thomas, *Elliptic Problems in Linear Difference Equations over a Network*, Tech. Rep. (Watson Sc. Comp. Lab. Rep., Columbia University, New York, 1949).

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

Appendix A: Source code

All code for this report was written in C++ and Python 3.8, and the complete set of files can be found at `https://github.com/FunkMarvel/FYS3150_Project_1.git`