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:

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

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 **a**, **b** and **c**. Their components are defined the following way:

$$\mathbf{A} = \begin{bmatrix} b_1 & c_1 & 0 & ... & ... & 0\\ a_1 & b_2 & c_2 & 0 & ... & ...\\ 0 & a_2 & b_3 & c_3 & 0 & ...\\ ... & ... & ... & ... & ... & ...\\ 0 & ... & ... & a_{n-2} & b_{n-1} & c_{n-1}\\ 0 & ... & ... & 0 & a_{n-1} & b_n \end{bmatrix}$$

As the bands above and below the diagonal have one element less than the diagonal itself, we have that **a** and **c** both have $n-1$ elements instead of n.

III. IMPLEMENTATION

IV. ANALYSIS

A. Plots of the general Thomas algorithm

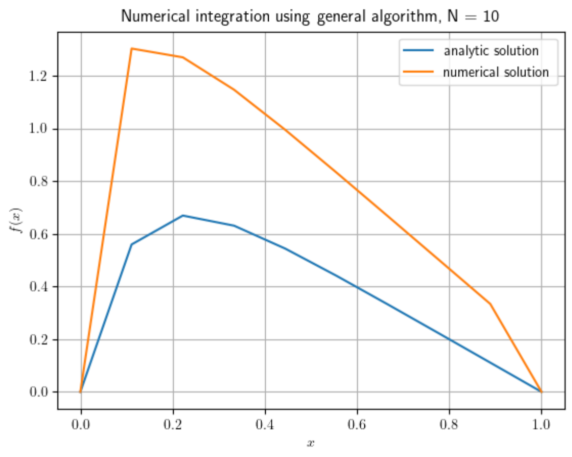

Figure 1. Plot of numerical and analytical solution, using the general Thomas algorithm with $N = 10^1$.

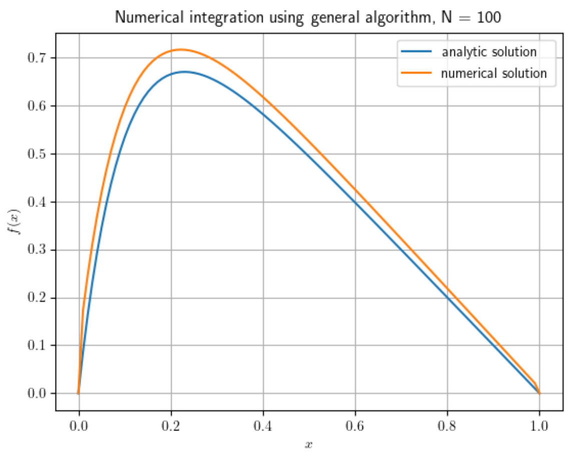

Figure 2. Plot of numerical and analytical solution, using the general Thomas algorithm with $N = 10^2$.

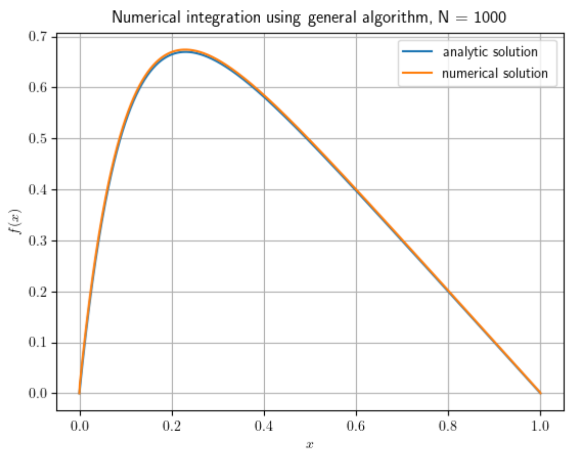

Figure 3. Plot of numerical and analytical solution, using the general Thomas algorithm with $N = 10^3$.

B. Relative error for Thomas' algorithms

$\log_{10}(h)$:	ϵ general algorithm	ϵ special algorithm	N
$-1.041\,393$	$3.026\,200 \times 10^{-1}$	$3.601\,314 \times 10^{-1}$	10^1
$-2.004\,321$	$3.426\,303 \times 10^{-2}$	$4.249\,885 \times 10^{-2}$	10^2
$-3.000\,434$	$3.474\,750 \times 10^{-3}$	$4.338\,587 \times 10^{-3}$	10^3
$-4.000\,043$	$3.479\,720 \times 10^{-4}$	$4.347\,831 \times 10^{-4}$	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

Table I. Table with $\log_{10}$ of relative error for general and special Thomas' algorithms, and $\log_{10}$ of step size h

V. CONCLUSION

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