# FYS 4150 - Project 1

Jørgen Glenndal & Ola Mårem

September 2022

## I Problem 1

Given the Poisson equation

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

we can, for a given $u(x)$ find a solution to $f(x)$.

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

We do this by differentiating the equation twice.

$$\frac{du}{dx} = -\left(1 - e^{-10}\right) + 10e^{-10x}$$

$$\frac{d^2u}{dx^2} = -100e^{-10x} \qquad \Longrightarrow \qquad f(x) = 100e^{-10x}$$

## II Problem 2

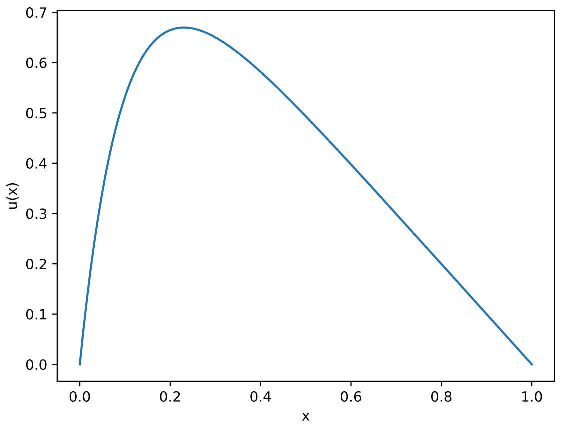

Figure 1: $u(x)$ for $x \in [0,1]$ with $n = 10^3$ points.

Using the numerical algorithm to be found in our GitHub repository.

## III Problem 3

We start by calculating the Taylor polynomial of degree 2 of $u$ at some point $a$. This will give us an approximation of $u$, which typically is close to the exact solution when $x$ is close to $a$. The approximation will be a function of $x$.

$$u(x) \approx T_2(u;a)(x) = \sum_0^2 \frac{u^n(a)}{n!}(x-a)^n = u(a)+u'(a)(x-a)+\frac{u''(a)}{2}(x-a)^2 \quad (2)$$

We can now evaluate the polynomial $T_2(u;a)(x)$ at $x = a+h$ and at $x = a-h$ where $h$ should be a small number. This results in two equations:

$$\begin{aligned} u(a+h) &\approx u(a) + u'(a)h + \frac{u''(a)}{2}h^2 \\ u(a-h) &\approx u(a) - u'(a)h + \frac{u''(a)}{2}h^2 \end{aligned} \quad (3)$$

Adding the two equations together gives

$$\begin{aligned} u(a+h) + u(a-h) &\approx 2u(a) + u''(a)h^2 \\ \Rightarrow u''(a) &\approx \frac{u(a+h) - 2u(a) + u(a-h)}{2} \end{aligned} \quad (4)$$

This is then an approximation to the double derivative of $u$ around the arbitrary function argument $a$. The Possion equation can now be written as

$$-\left(\frac{v(x+h) - 2v(x) + v(x-h)}{2}\right) = f(x) \quad (5)$$

where we have replaced $u$ with $v$ since we have approximated the equation such that we no longer solve for the exact solution $u$. We can now discretize $x$.

$$-\left(\frac{v(x_i+h) - 2v(x_i) + v(x_i-h)}{2}\right) = f(x_i) \quad (6)$$

Letting $v(x_i) \to v_i$ and $v(x_i + h) \to v_{i+1}$ we get

$$-\left(\frac{v_{i+1} - 2v_i + v_{i-1}}{2}\right) = f_i \quad (7)$$

which is a discretized version of the Possion equation.

## IV Problem 4

We have our discrete equation from exercise III, and use the first few values of $i$ to show that this can be written on the form $\mathbf{A}\vec{v} = \vec{g}$.

$$i = 0: \qquad -v_1 + 2v_0 - v_{-1} = h^2 f_0$$

$$i = 1: \qquad -v_2 + 2v_1 - v_0 = h^2 f_1$$

$$i = 2: \qquad -v_3 + 2v_2 - v_1 = h^2 f_2$$

$$i = 3: \qquad -v_4 + 2v_3 - v_2 = h^2 f_3$$

So we get

$$\begin{pmatrix} 2 & -1 & 0 & 0 & \cdots \\ -1 & 2 & -1 & 0 & \\ 0 & -1 & 2 & -1 & \\ 0 & 0 & -1 & 2 & \\ \vdots & & & & \ddots \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_i \end{pmatrix} = \begin{pmatrix} h^2 f_0 + v_{-1} \\ h^2 f_1 \\ h^2 f_2 \\ h^2 f_3 + v_4 \\ \vdots \\ g_i \end{pmatrix} \tag{8}$$

Which is indeed on the form $\mathbf{A}\vec{v} = \vec{g}$ where $\mathbf{A}$ is a tridiagonal matrix. $\vec{g}$ is linked to the original differential equation through $f(x)$, but with discrete step-length $h$ and built in boundary conditions on the first and last element.

## V Problem 5

### V.I a)

Because we have $m$ as a complete solution for $n$ amount of equations, then $n$ will be upper limited by $m$.

### V.II b)

If $h$ is constant, then the precision of each step would be equal. This would lead to better precision for the first part of the complete solution, but we would be missing a part of the solution. If we however need the complete set of solutions, then a variable $h$ would need to be implemented even though the precision at the first steps are worse than with $h =$ constant.

# VI Problem 6

## VI.I a)

The algorithm used to solve this problem is called the Thomas algorithm, also known as the tridiagonal matrix algorithm, and is done by the use of Gaussian elimination. the generalized form is as follows

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & \cdots \\ a_2 & b_2 & c_2 & 0 & \\ 0 & a_3 & b_3 & c_3 & \\ 0 & 0 & a_4 & b_4 & \\ \vdots & & & & \ddots \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_i \end{pmatrix} = \begin{pmatrix} h^2 f(x) + v_{-1} \\ h^2 f(x) \\ h^2 f(x) \\ h^2 f(x) + v_4 \\ \vdots \\ g_i \end{pmatrix} \tag{9}$$

We do this by following the forward algorithm, making sure that the upper triangle of our matrix become all zeros. The new variables $c_i'$ and $d_i'$ is designated as our first variables to do this.

$$c_i' = \begin{cases} \frac{c}{b} & i = 1 \\ \frac{c}{b - ac_{i-1}'} & i = 2, 3, ..., n-1 \end{cases}$$

$$g_i' = \begin{cases} \frac{g}{b} & i = 1 \\ \frac{cg - ag_{i-1}}{b - ag_{i-1}'} & i = 2, 3, ..., n \end{cases}$$

The solution can then be found by substituting the newly defined variables backwards such that the lower triangle of the matrix also become zeros. This results in the solution for $v_i$.

$$\begin{aligned} v_n &= g_n' \\ v_i &= g_i' - c_i' v_{i+1} \end{aligned} \tag{10}$$

## VI.II b)

We have three loops that run approximately $n$ times. There is therefore 3n,5n and 2n FLOPs in the three loops respectively, meaning that there are 10n FLOPs in the general algorithm.

## VII Problem 7

### VII.I a)

Take a look at problem_2_and_7.cpp in the GitHub repository

### VII.II b)

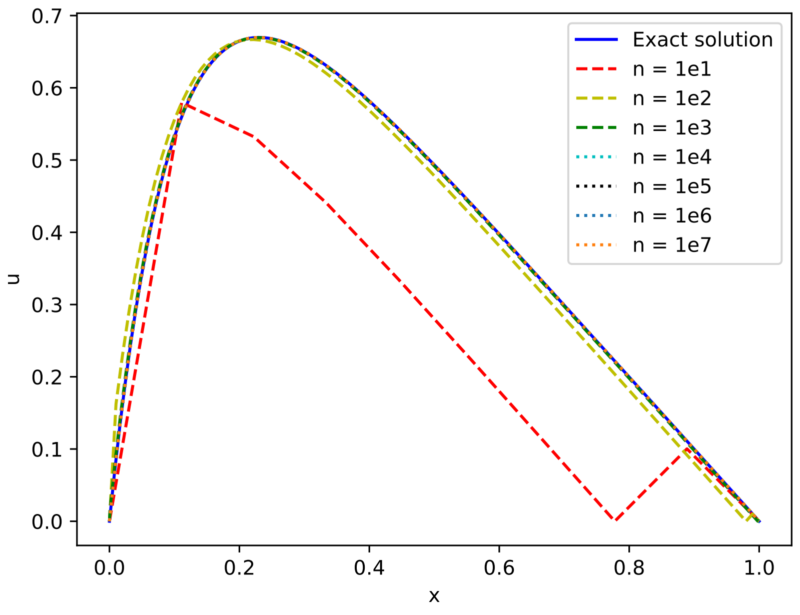

Figure 2: Comparison between the exact solution for $u(x)$ and the approximation for different n-steps $v_n(x)$

## VIII Problem 8

### VIII.I a) & b)

We avoided calculating the errors at the boundaries by letting the for loop run from the second element to the next to last element, since the first and last element represents the function value at the boundaries. In figure 3 we see that the absolute error, in general, decreases with higher n. The same is observed for the relative error except at $x$ close to 1.

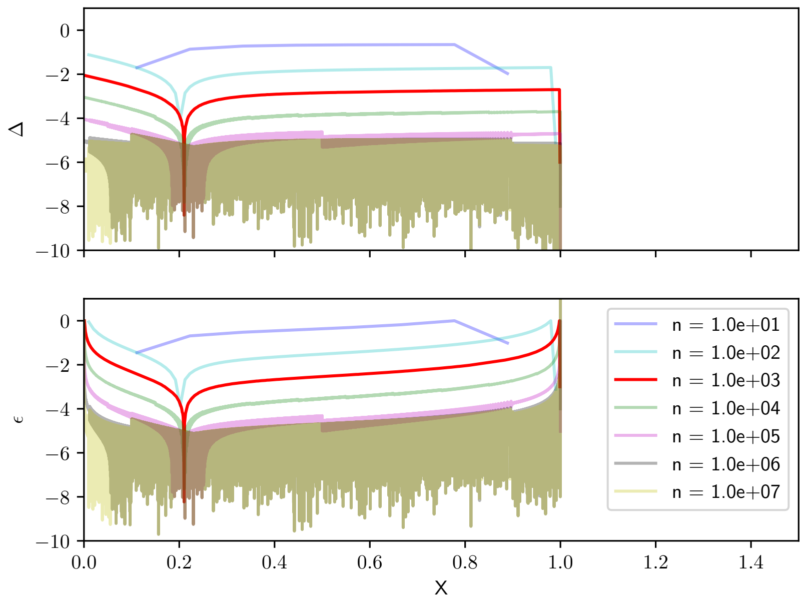

Figure 3: The top panel shows the absolute error and the lower panel shows the relative error of the solution along the $x$-axis.

| n | $\epsilon$ |
|---|---|
| $10^1$ | 1 |
| $10^2$ | 1 |
| $10^3$ | 1 |
| $10^4$ | 1 |
| $10^5$ | 1 |
| $10^6$ | $3.395 \cdot 10^{11}$ |
| $10^7$ | $7.977 \cdot 10^{11}$ |

Table 1: The maximum error along each of the n-values stated in figure 3.

## IX Problem 9

We specialized the algorithm by replacing the $\vec{a}$, $\vec{b}$ and $\vec{c}$ vectors with the integer numbers -1,2 and -1 respectively. The number of FLOPs was then reduced to 7n for the specialized algorithm. The code can be found in the GitHub repository.

## X Problem 10

We see that the specialized algorithm is faster, on average, than the generalized method. Both algorithms were ran for all n values ($n = 10, 10^2, ..., 10^7$) 100 times. We then averaged the time it took for each n-value. The result can be seen in the table below.

| n | General [s] | Specialized [s] |
|---|---|---|
| 10 | $3 \cdot 10^{-6}$ | $2 \cdot 10^{-6}$ |
| $10^2$ | $8 \cdot 10^{-6}$ | $6 \cdot 10^{-6}$ |
| $10^3$ | $4.1 \cdot 10^{-5}$ | $3.6 \cdot 10^{-5}$ |
| $10^4$ | $6.7 \cdot 10^{-4}$ | $5.5 \cdot 10^{-4}$ |
| $10^5$ | $6.3 \cdot 10^{-3}$ | $5.2 \cdot 10^{-3}$ |
| $10^6$ | $6.2 \cdot 10^{-2}$ | $5.1 \cdot 10^{-2}$ |
| $10^7$ | $6.4 \cdot 10^{-1}$ | $5.2 \cdot 10^{-1}$ |

Table 2: Overview of the average time taken for the generalized (10n-FLOPs) and the specialized (7n-FLOPs) Thomas algorithm.