

# Project 1: Computational Physics - FYS3150

Fredrik Hoftun & Mikkel Metzsch Jensen

September 09, 2020

## Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Introduction</b>   | <b>2</b> |
| <b>2</b> | <b>Method</b>   | <b>2</b> |
| 2.1      | Defining the problem . . . . .                                | 2        |
| 2.2      | Rewritting the problem as a set of linear equations . . . . . | 3        |
| 2.3      | General solution using Gaussian elimination . . . . .         | 4        |
| 2.4      | Simplified problem specific solution . . . . .                | 5        |
| 2.5      | LU decomposition . . . . .                                    | 6        |
| 2.6      | Comparing precision and error . . . . .                       | 6        |
| 2.7      | Implementation . . . . .                                      | 6        |
| <b>3</b> | <b>Results</b>  | <b>6</b> |
| 3.1      | General algorithm . . . . .                                   | 8        |
| 3.2      | Special algorithm . . . . .                                   | 8        |
| 3.3      | LU decomposition . . . . .                                    | 8        |
| <b>4</b> | <b>Discussion</b>   | <b>8</b> |
| <b>5</b> | <b>Conclusion</b>   | <b>8</b> |
| <b>6</b> | <b>References</b>   | <b>8</b> |

## Abstract

The goal of this project were to...  
What did we do?  
What did we find?

# 1 Introduction

In this project we will investigate different numerical approaches for the solving of the one-dimensional Poisson equation with Dirichlet boundary conditions. This is a problem that is used in many different scientific applications. It is used to describe electrostatic and magnetostatic phenomena in a quantitative manner. It also helps to understand diffusion and propagation problems. The solution is generally used in a wide range of fields such as engineering, physics, mathematics, biology, chemistry, etc. [1]. In this project we have used a second derivative approximation in order to rewrite the problem as a set of linear equations. We have written three different algorithms: A general and a specialized algorithm using gaussian elimination and then LU decomposition. These are described individually in the method sections along with a calculation of the number of floating point operations for each of them. Since we know the analytical solution for our problem we have compared relative error for different choices of stepsize  $h$  in the algorithm. We have also compared the CPU time of these computations. We found a significant difference in both precision (due to round off errors) and CPU time, which are presented in the results section.

1. Motivate the reader
2. What have I done
3. The structure of the report
4. conclusion?

## 2 Method

### 2.1 Defining the problem

We are going to solve the one-dimensional Poisson equation with Dirichlet boundary conditions given as the following:

$$u''(x) = f(x), \quad x \in (0, 1), \quad u(0) = u(1) = 0$$

In our case we will provide the function:

$$f(x) = 100e^{-10x}$$

Note that the algorithms for solving this problem will not be dependent on this specific choice of  $f(x)$ . The reason why we stick to this function is because we have the following analytical solution for  $f(x)$ :

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

We will use this analytical solution to evaluate the precision of the numerical solutions for different choices of steplength  $h$  between the discretized gridpoints  $x_i$ .

## 2.2 Rewriting the problem as a set of linear equations

In order to solve the Poisson equation numerically we discretize  $u$  as  $v_i$  with grid points  $x_i = ih$  in the interval  $x \in [x_0 = 0, x_1 = 1]$ . We then have the step length  $h = 1/(n + 1)$ . We use the following second derivative approximation

$$-u''(x_i) \approx -\frac{v_{i+1} + 2v_i - v_{i-1}}{h^2} = f(x_i)$$

$\Leftrightarrow$

$$-v_{i-1} + 2v_i - v_{i+1} = h^2 f(x_i)$$

We define the column vector  $\mathbf{v} = [v_1, v_2, \dots, v_{n+1}]$  and try to setup the equation for every step  $i$ . As we do this we see a pattern appearing

$$\begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} v_0 \\ \vdots \\ v_{n+1} \end{bmatrix} = h^2 f(x_0)$$

$$\begin{bmatrix} 2 & -1 & 0 & \cdots & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & \cdots \end{bmatrix} \begin{bmatrix} v_0 \\ \vdots \\ v_{n+1} \end{bmatrix} = h^2 \begin{bmatrix} f(x_0) \\ f(x_1) \end{bmatrix}$$

$\vdots$

$$\begin{bmatrix} 2 & -1 & 0 & \cdots & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & \cdots \\ 0 & -1 & 2 & -1 & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & & -1 & 2 & -1 \\ 0 & \cdots & & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_0 \\ \vdots \\ v_{n+1} \end{bmatrix} = h^2 \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_{n+1}) \end{bmatrix}$$

From this we see that we can write the problem as a linear set of equation:

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

With the following definitions:

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & \cdots & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & \cdots \\ 0 & -1 & 2 & -1 & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & & -1 & 2 & -1 \\ 0 & \cdots & & 0 & -1 & 2 \end{bmatrix}, \mathbf{v} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n+1} \end{bmatrix}, \tilde{\mathbf{g}} = h^2 \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f_{n+1} \end{bmatrix}$$

In this project we use  $f(x) = 100e^{-10x}$ . The solution for the Poisson equation in this case is given to be  $u(x) = 1 - (1 - e^{-10})x - e^{-10x}$ . We can ensure that this is true by inserting it and checking that the equation holds remains true. First find the double derivative of  $u(x)$ :

$$u'(x) = -(1 - e^{-10}) + 10e^{-10x}, \quad u''(x) = -100e^{-10x}$$

We now see that the solution satisfy the Poisson equation:

$$-u''(x) = 100e^{-10x} = f(x)$$

## 2.3 General solution using Gaussian elimination

Do Gaussian elimination... (or not)

We can solve our problem generally using Gaussian elimination on the matrix  $\mathbf{A}\mathbf{v} = \mathbf{g}$ , where  $a_i$  are the elements below the diagonal,  $b_i$  are the elements on the diagonal and  $c_i$  are the elements above the diagonal.

---

### Algorithm 1 General algorithm

---

|  |   |
|--|---|
| 1: <b>for</b> $i = 2, \dots, n$ <b>do</b><br>2: $b_i = b_i - c_{i-1} \cdot a_i / b_{i-1}$<br>3: $g_i = g_i - g_{i-1} \cdot a_i / b_{i-1}$<br>4: <b>end for</b> | $\triangleright$ Forward substitution eliminating $a_i$<br>$\triangleright$ Update $b_i$<br>$\triangleright$ Update $g_i$ |
| 5: $v_n = 0$<br>6: <b>for</b> $i = n - 1, \dots, 1$ <b>do</b><br>7: $v_i = \frac{g_i - c_i \cdot v_{i+1}}{b_i}$<br>8: <b>end for</b>                           | $\triangleright$ Backward substitution obtaining $v_i$  |

---

We can calculate the algorithms number of Floating Point Operations Per Second (FLOPS) easily. Each arithmetic operation is one FLO(PS). So in our forward substitution we have  $2 \cdot 3$  FLOPS and in the backward substitution we have  $1 + 3$  FLOPS. The forward substitution goes from 2 to  $n$ , totaling  $n - 2$  points. The backward substitution also has  $n - 2$  points. Then the total number of FLOPS are:

$$(6 + 3)(n - 2) + 1 = 9n - 17$$

## 2.4 Simplified problem specific solution

In this specific case we have  $a = -1$ ,  $b = 2$ ,  $c = -1$  which means we can further simplify our matrix  $\mathbf{A}$ :

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} 2 & -1 & 0 & \cdots & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & \cdots \\ 0 & -1 & 2 & -1 & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & & -1 & 2 & -1 \\ 0 & \cdots & & 0 & -1 & 2 \end{bmatrix} \sim \begin{bmatrix} 2 & -1 & 0 & \cdots & \cdots & 0 \\ 0 & 3/2 & -1 & 0 & \cdots & \cdots \\ 0 & -1 & 2 & -1 & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & & -1 & 2 & -1 \\ 0 & \cdots & & 0 & -1 & 2 \end{bmatrix} \\ &\sim \begin{bmatrix} 2 & -1 & 0 & \cdots & \cdots & 0 \\ 0 & 3/2 & -1 & 0 & \cdots & \cdots \\ 0 & 0 & 4/3 & -1 & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & & -1 & 2 & -1 \\ 0 & \cdots & & 0 & -1 & 2 \end{bmatrix} \sim \begin{bmatrix} 2 & -1 & 0 & \cdots & \cdots & 0 \\ 0 & 3/2 & -1 & 0 & \cdots & \cdots \\ 0 & 0 & 4/3 & -1 & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & & 0 & i_n/i_n - 1 & -1 \\ 0 & \cdots & & 0 & 0 & i_n + 1/i_n \end{bmatrix} \end{aligned}$$

Where we have called the last diagonal element  $i_n$ . We see that  $b_i = \frac{i+1}{i}$ , reducing the computation time. With our new  $b_i$  we can create a new algorithm.

Similarly to the general algorithm we can calculate total FLOPS:

$$(2 * 2 + 2)(n - 2) + 1 = 6n - 11$$

Which for large  $n$  is considerably less. If we take into account that  $b_i = \frac{i+1}{i}$  can be precalculated for all  $i \in [1, \dots, n]$  before hand, we get a total number of FP:

$$(2 + 2)(n - 2) = 4n - 8$$

which for large  $n$  can be approximated to  $4n$

---

**Algorithm 2** Special algorithm, where  $a_i = -1$ ,  $b_i = 2$ ,  $c_i = -1$ 

---

```
1: for  $i = 2, \dots, n$  do                                ▷ Forward substitution eliminating  $a_i$ 
2:    $b_i = i + 1/i$                                          ▷ Update  $b_i$ 
3:    $g_i = g_i + g_{i-1}/b_{i-1}$                              ▷ Update  $g_i$ 
4: end for

5:  $v_n = 0$                                                 ▷ Backward substitution obtaining  $v_i$ 
6: for  $i = n - 1, \dots, 1$  do
7:    $v_i = \frac{g_i + v_{i+1}}{b_i}$ 
8: end for
```

---

## 2.5 LU decomposition

For the LU decomposition we ...

FLOPS  $O(n^3)$  se [https://en.wikipedia.org/wiki/LU\\_decomposition](https://en.wikipedia.org/wiki/LU_decomposition), søk etter "float"

## 2.6 Comparing precision and error

## 2.7 Implementation

We used C++ to implement our algorithms ... and we used Python's library Matplotlib to plot our results.

## 3 Results

GITHUB LINK HERE

Max error:

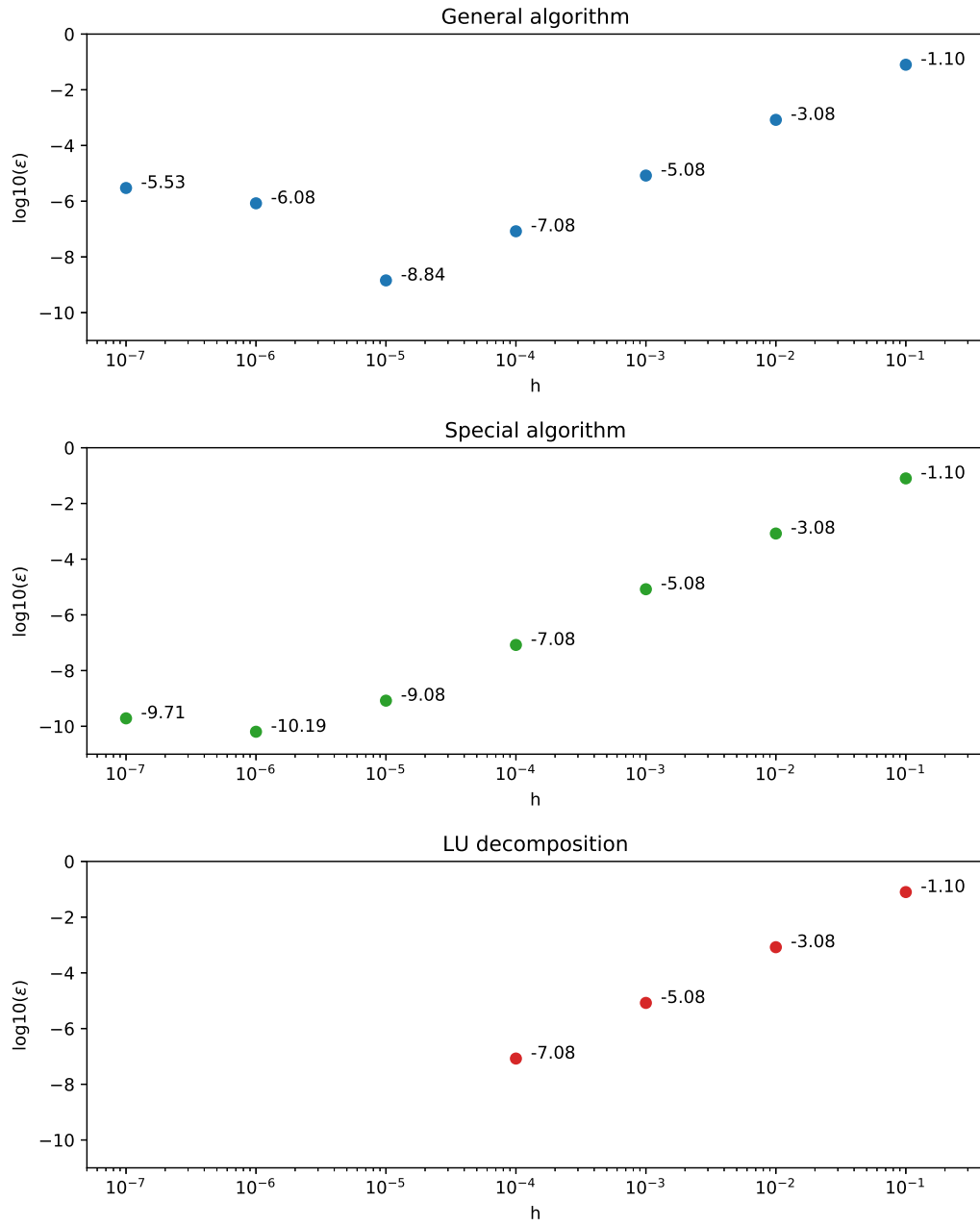


Figure 1: Log10 of the maximum error for each numerical solution compared to the analytical solution for different number of gridpoints  $n$  and different numerical methods.

Table 1: CPU Time

| <b>N</b> | <b>General algorithm [s]</b> | <b>Special algorithm</b> | <b>LU Decomposition</b> |
|----------|------------------------------|--------------------------|-------------------------|
| $10^1$   | $3 \times 10^{-6}$           | $3 \times 10^{-6}$       | $9.81 \times 10^{-4}$   |
| $10^2$   | $4 \times 10^{-6}$           | $4 \times 10^{-6}$       | $1.96 \times 10^{-4}$   |
| $10^3$   | $3.8 \times 10^{-5}$         | $3.7 \times 10^{-5}$     | $1.02 \times 10^{-2}$   |
| $10^4$   | $3.41 \times 10^{-4}$        | $3.54 \times 10^{-4}$    | 2.45                    |
| $10^5$   | $3.79 \times 10^{-3}$        | $3.50 \times 10^{-3}$    | nan                     |
| $10^6$   | $3.57 \times 10^{-2}$        | $3.24 \times 10^{-2}$    | nan                     |
| $10^7$   | $3.16 \times 10^{-1}$        | $3.24 \times 10^{-1}$    | nan                     |

### 3.1 General algorithm

### 3.2 Special algorithm

### 3.3 LU decomposition

## 4 Discussion

Error analysis

## 5 Conclusion

In this report we have used three different ways of computing  $\mathbf{A}\mathbf{v} = \mathbf{g}$  and have seen that efficiency of the methods vary greatly. We have witnessed the importance of efficient implementation of algorithms ...

## 6 References

### References

- [1] S. B. Gueye, K. Talla and C. Mbow, "Solution of 1d poisson equation with neumann-dirichlet and dirichlet-neumann boundary conditions using the finite difference method", Journal of Electromagnetic Analysis and Applications, Vol. 6, No. 10, pp. 309, 2014.