Project 1: Computational Physics - FYS3150

Fredrik Hoftun & Mikkel Metzsch Jensen

September 09, 2020

Contents

Abstract

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

1 Introduction

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

In this project we will investegate different approaches to solve the one-dimensional Poisson equation with Dirichlet boundary conditions given as follows:

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

We will rewrite this as a set of linear equations, and solve it by a number of different computational approaches on either gaussian elimination or LU decomposition. We will solve the equation above with the function:

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

Where the analytical solution then is given as:

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

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

2 Method

Show test and example og code somewhere in method. Show that your code works before showing results later.

2.1 defining the problem

2.2 Rewritting 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

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

$\iff$

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

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

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

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

$$\vdots$$

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

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

$$\mathbf{Av} = \mathbf{g}$$

With the following definitions:

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

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 equarion 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 Gausian elimination

Do Gaussial elimination... (or not)
We can solve our problem generally using Gaussian elimination on the matrix $\mathbf{Av} = \mathbf{g}$, 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

for $i = 2, \ldots, n$ **do** ▷ Forward substitution eliminating a_i
    $b_i = b_i - c_{i-1} \cdot a_i / b_{i-1}$ ▷ Update b_i
    $g_i = g_i - g_{i-1} \cdot a_i / b_{i-1}$ ▷ Update g_i
end for

$v_n = 0$ ▷ Backward substitution obtaining v_i
for $i = n - 1, \ldots, 1$ **do**
    $v_i = \frac{g_i - c_i \cdot v_{i+1}}{b_i}$
end for

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 $\mathbf{A}$:

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

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

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.

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

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

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

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.

2.5 LU decomposition

For the LU decomposition we ...
FLOPS $O(n^3)$ se 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 Pythons library Matplotlib to plot our results.

3 Results

GITHUB LINK HERE
Max error:

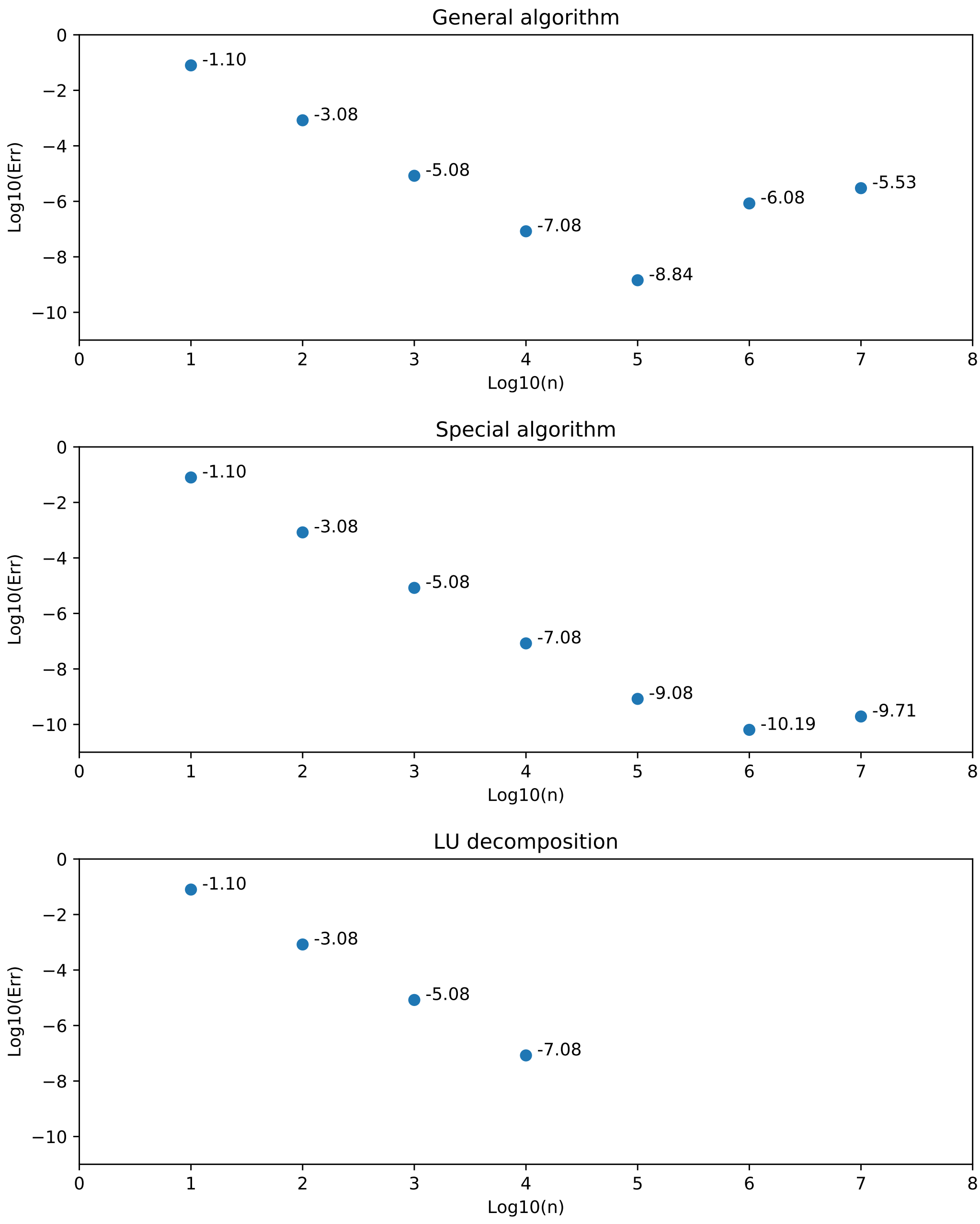

Figure 1: Log10 of the maximum error for each numerical solution compared to the analytical solution for different number of gridpoints n and different numerical methods.

Table 1: CPU Time

N	General algorithm [s]	Special algorithm	LU Decomposition
10^1	3×10^{-6}	3×10^{-6}	9.81×10^{-4}
10^2	4×10^{-6}	4×10^{-6}	1.96×10^{-4}
10^3	3.8×10^{-5}	3.7×10^{-5}	1.02×10^{-2}
10^4	3.41×10^{-4}	3.54×10^{-4}	2.45
10^5	3.79×10^{-3}	3.50×10^{-3}	nan
10^6	3.57×10^{-2}	3.24×10^{-2}	nan
10^7	3.16×10^{-1}	3.24×10^{-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 $\mathbf{Av} = \mathbf{g}$ and have seen that efficiency of the methods vary greatly. We have witnessed the importance of efficient implementation of algorithms ...

6 References

References

[1] Test