Universitetet i Oslo

FYS4150

Project 1

Jessie Warraich
Christina Knudsen

September 7, 2019

Contents

1 Abstract

2 Introduction

In this project we will solve the Poisson equation usin two different algorithms. Poisson's equation is a partial differential equation of elliptic type with broad utility in mechanical engeneering and theoretical physics. In the latter, it is for instance used to describe an electrostatic potential generated by a localized charge distribution or a mass distribution. By assuming spherical symmetry, the equation is reduced to one dimention.
We are going to use two different algorithms to solve the Poisson equation. The first is the tridioganal matrix algorithm and the second is the LU decomposition method. Where the tridiagonal matrix algorithm can be formed to the spesific problem at hand, the LU-decomposition algorithm is a general algorithm which can be used on any matrix. After uding the different theoretical models on Poissons equation, we will aldo test the precision of methods at different levels.

3 Theory

3.1 The Poisson Equation

The Poisson Equation is a classical equation from electromagnetism. The electrostatic potential Φ is generated by a localized charge distribution $\rho(\mathbf{r})$. In three dimentions it reads

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

With a spherically symmetric Φ and $\rho(\mathbf{r})$ the equations simplifies to a one-dimensional equation in r, namely

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

which can be rewritten via a substitution $\Phi(r) = \phi(r)/r$ as

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

The inhomogeneous term f or source term is given by the charge distribution ρ multiplied by r and the constant -4π. We will rewrite this equation by letting $\phi \to u$ and $r \to x$. The general one-dimensional Poisson equation reads then

$$-u''(x) = f(x).$$

In this project the source term is given as $f(x) = 100e^{-10x}$, and we will compare it to the analytical solution $u(x) = 1 - (1 - e^{-10})x - e^{-10x}$. We will solve the equation with Dirichlet boundary conditions in the interval $x \in [0, 1]$.

3.2 Approximation of the second derivative

We will solve the one-dimentional Poisson equation with Dirichlet boundary conditions by rewriting it as a set of linear equations. The discretized approximation is defined as u

and v_i, with grid points $x_i = ih$ in the interval from $x_0 = 0$ to $x_{n+1} = 1$. The step length is given as $h = 1/(n+1)$. This gives the boundary conditions $v_0 = v_{n+1} = 0$. The second order derivative is approximated with the three point formula, and reads

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

After defining $\mathbf{f} = h^2 f$, one can rewrite the equation as $-v_{i+1} + v_{i-1} - 2v_i = h^2 f_i$. This can now be represented as a matrix equation, with $\mathbf{A}$ is a $n \times n$ triagonal matrix.

$$A\mathbf{v} = \mathbf{f} \tag{2}$$

$$\begin{bmatrix} 2 & -1 & 0 & 0 & \dots & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & \dots & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix} \tag{3}$$

3.3 The tridiagonal matrix

3.4 Algorithm for the tridiagonal matrix

As deducted in section 3.3, we can develop an algorithm to solve the tridiagonal matrix. As seen from the matrix equation EQUATION, $\mathbf{A}$ is a triagonal matrix with only three numbers along the diagonals, with the rest of the matrix being equal to zero. To make an efficient code(??), this can be computed as three vecors a_i, b_i and c_i.
This can even become more efficient by reducing the number of mathematical operation. Since a_i, b_i and c_i are vectors containing the same constant in the whole vector, one can just muliply with just the number. Reducing the number of mathematical operations even more, one can calculate which number is used on beforehand and use the pre-computed numbers direcly in the calculations.

3.5 LU-decomposition

In addition to solve Poissons equation by the tridiagonal matrix algorithm, one can also use another algorithm for solving the set of matrix equations. The LU-decomposition decomposes the initial matrix $\mathbf{A}$ to a lower triangular matrix $\mathbf{L}$ and an upper triagonal matrix $\mathbf{U}$,

$$\mathbf{A} = \mathbf{LU}.$$

Now taking a look back at equation 2, and substitute $\mathbf{A}$ with the newfound expression

$$\mathbf{LUv} = \mathbf{f}.$$

These can be solved as

$$\mathbf{Ly} = \mathbf{f} \tag{4}$$

$$\mathbf{Uv} = \mathbf{y} \tag{5}$$

3.6 Number of FLOPS

4 Method

The programs can be found in our GIT repository.

4.1 General algorithm

This algorithm is made of two different loops, the decomposition and forward subtituion followed by the backward substitution, as previously discussed (legg inn referanse).

```
f_tilde[0] = f[0];
b_tilde[0] = b[0];
for(int i=1; i<n+1; i++){
    b_tilde[i] = b[i] - a[i-1]*c[i-1]/b_tilde[i-1];
    f_tilde[i] = f[i] - a[i-1]*f_tilde[i-1]/b_tilde[i-1];
}
v[0] = v[n] = 0;
v[n-1] = f_tilde[n-1]/b_tilde[n-1];
for(int i=n-2; i>=1; i--){
    v[i] = (f_tilde[i]-c[i]*v[i+1])/b_tilde[i];
    error[i] = log10(abs((v[i]-u[i])/u[i]));
}
```

As seen from the psuedo code above, nothing is assumed of the tridiagonal matrix. Therefore it can be used as a general code to solve tridiagonal matrix multiplications.

4.2 Special algorithm

The algoritm with our specific case do require less of our computer, as discussed in section 3.3. Since the a lot of the values used in the loop can be pre-computed, it is easier to implement. As in section 4.1, we still have two loops over the decomposition, forward and backward substitution, these are now much simpler.

```
f_tilde[0] = f[0];
b_tilde[0] = 2;
for(int i=1; i<n+1; i++){
    b_tilde[i] = 2 - 1/b_tilde[i-1];
    f_tilde[i] = f[i] + f_tilde[i-1]/b_tilde[i-1];
}
v[0] = v[n] = 0;
v[n-1] = f_tilde[n-1]/b_tilde[n-1];
for(int i=n-2; i>=1; i--){
    v[i] = (f_tilde[i]+v[i+1])/b_tilde[i]
    error[i] = log10(abs((v[i]-u[i])/u[i]));
}
```

4.3 LU-decomposition algorithm

The LU-decomposition algorithm is implemented by using the Armadillo Linear Algebra library for *C++*. After initializing the tridiagonal matrix $\mathbf{A}$, we can solve equation 4 and 5 by using Armadillo's *solve* function. In addition, we initialize the upper left and bottom right part of the matrix, i.e. the start and end of the loop.(høres rart ut??). This is done in order to make the code more efficient, since we will not need *if*-statements.

```
A(0,0) = 2.;
A(n-1,n-1) = 2;
A(0,1) = -1;
A(1,0) = -1;
A(n-1,n-2) = -1;
A(n-2,n-1) = -1;
for (int i = 1; i <= n-2; i++){
    A(i,i) = 2;
    A(i, i+1) = -1;
    A(i, i-1) = -1;
}
```

4.4 Comparison to the analytical solution

The analytic solution was also implemented in the code. The algorithms were solved for different sizes of the matrices, and compared to the numerical solution by

$$\epsilon_i = \log_{10}\left(\left|\frac{v_i - u_i}{u_i}\right|\right). \tag{6}$$

Where u_i is the analytic solution and v_i is the numerical solution, both at a some point i. Equation 6 expresses the relative error between the two solutions. We will only look at the maximum error for different values of n.

5 Results

5.1 Numerical solution compared to the analytical

A plot of the analytical solution versus different values of n are shown in figure **??**.

5.2 Relative error

Here the relative error is shown in the log-log plot in figure **??**.

5.3 Computational speed

We also compared the computational speed of the different algorithms described in section **??**. This is shown in table **??** below, for different n-values.

6 Discussion

7 Conclusion

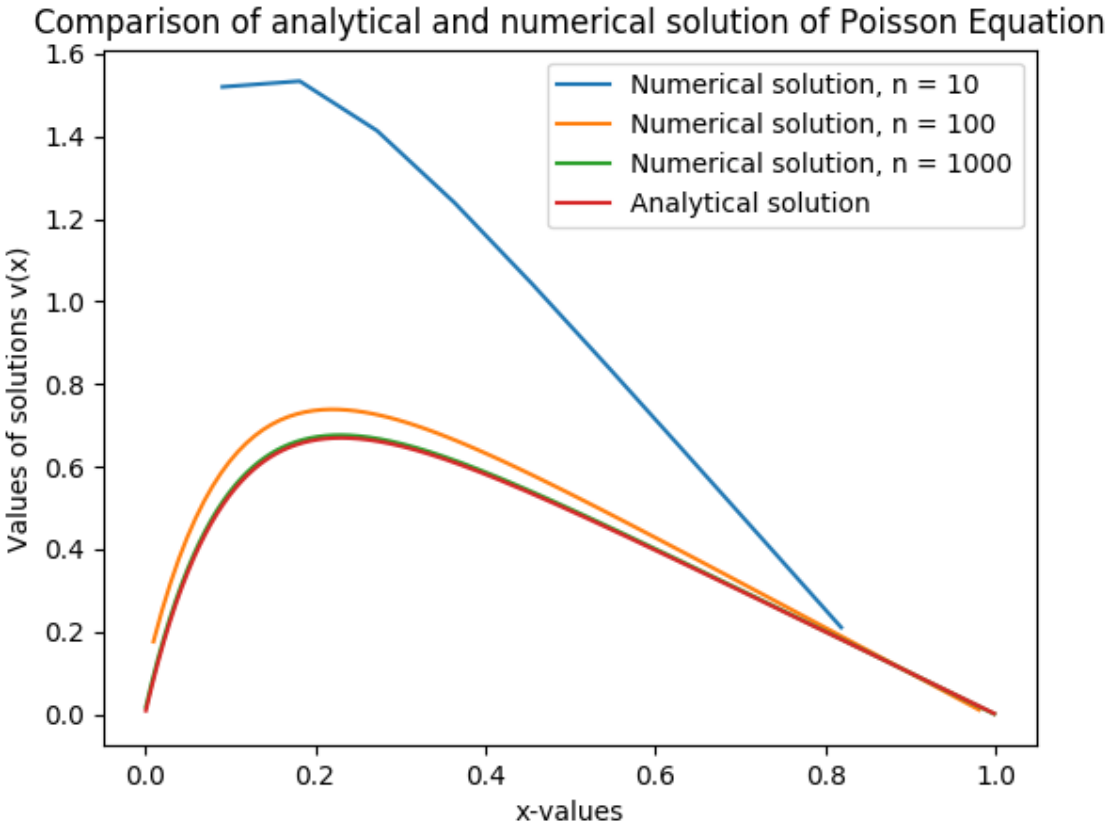

Figure 1: Plot of the analytical and numerical solution of Poisson's equation as described in section **??**. *Here the red line is the analytical solution, and the other lines are the numerical solutions with different step-sizes.*

N-values	General algorithm [s]	Special algorithm [s]	LU-decomposition [s]
10^1	$2.5 \cdot 10^{-5}$	$4.0 \cdot 10^{-6}$	$7.2 \cdot 10^{-3}$
10^2	$2.6 \cdot 10^{-5}$	$2.7 \cdot 10^{-5}$	$5.3 \cdot 10^{-2}$
10^3	$2.12 \cdot 10^{-4}$	$2.14 \cdot 10^{-4}$	$2.2 \cdot 10^{-1}$
10^4	$2.11 \cdot 10^{-3}$	$1.65 \cdot 10^{-3}$	16.42
10^5	$2.26 \cdot 10^{-2}$	$2.31 \cdot 10^{-2}$	N/A
10^6	$1.22 \cdot 10^{-1}$	$1.17 \cdot 10^{-1}$	N/A
10^7	1.83	1.59	N/A

Table 1: comparison of the time used to solve Poisson's equation with different algorithms and different values of n.

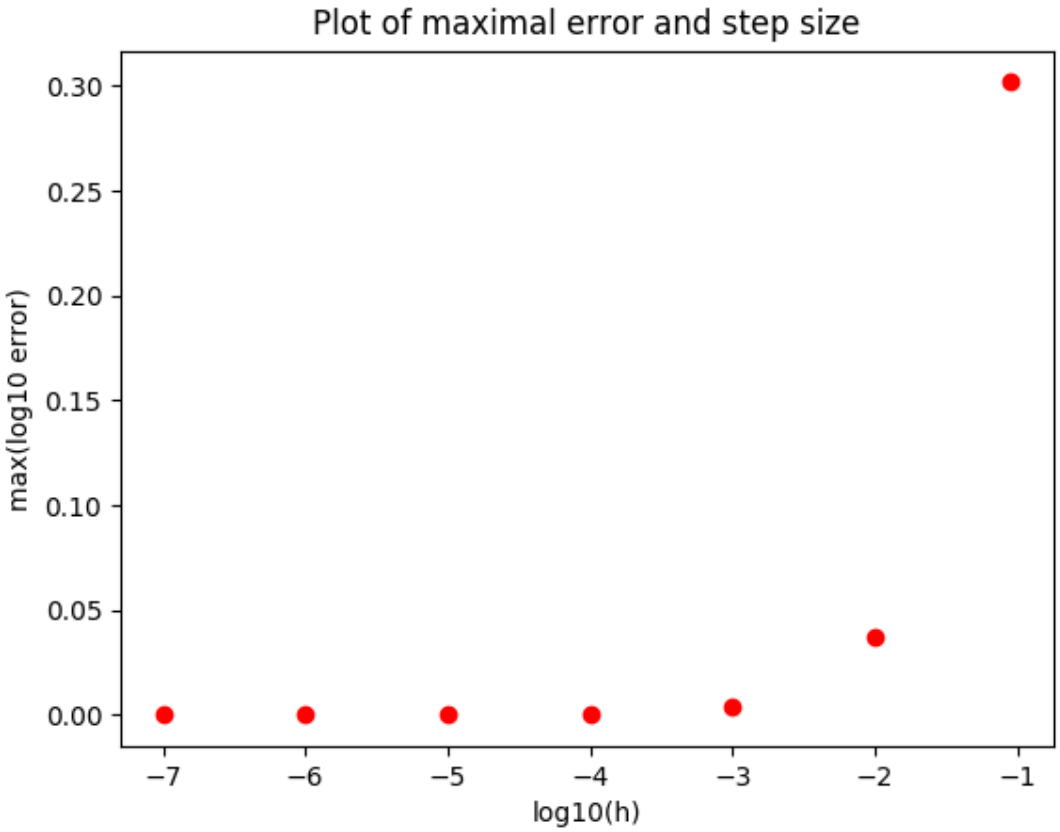

Figure 2: Plot of the relative error as described in section **??**. *Here the error is calculated by using equation 6.*