FYS4150 Project 1

Mikkel Killingmoe Christensen

September 13, 2017

Abstract

Insert Abstract here

GitHub repository:

`https://github.com/mikkello/FYS4150/`

1 Introduction

In this project, the goal is to find a numerical solution to a second degree differential equation. An example of such an equation is Poisson's equation in electromagnetism given by:

$$\frac{d^2\phi}{dr^2} = -4\pi r\rho(r) \tag{1}$$

This type of differential equation can be rewritten as the following with Dirichlet boundary conditions:

$$-u^{''}(x) = f(x), \quad x \in (0,1), \quad u(0) = u(1) = 0 \tag{2}$$

2 Theory

2.1 Poisson equation

To solve the one-dimensional Poisson equation with Dirchelet boundary conditions, an approximation of the second derivative is made using Taylor expansion:

$$\frac{d^2u}{dx^2} \approx \frac{u(x+h)+u(x-h)-2u(x)}{h^2} + O(h^2) \tag{3}$$

where $O(h^2)$ is the truncation error.

The equation is then discretized and a simpler notation is used to yield the following:

$$\frac{u_{i+1}+u_{i-1}-2u_i}{h^2} = -f_i \tag{4}$$

The step size is given by:

$$h = \frac{1}{n+1} \tag{5}$$

Furthermore, the equation is a system of linear equations and can thus be written as a linear equation using linear algebra. The equation reads as follows:

$$\mathbf{A}u = h^2\mathbf{f} \tag{6}$$

with

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{bmatrix} \tag{7}$$

being a tridiagonal $n \times n$ matrix. The problem is now possible to solve by finding the vector $\mathbf{u}$. For small matrices, this can easily be done by multiplying both sides with the inverse of $\mathbf{A}$. For larger matrices, however, this procedures is very slow and a better method is needed.

2.1.1 Analytical solution

Analytical solutions exists for the Poisson equation problem, and it will be useful to compare this to the numerical results. For the term:

$$f(x) = 100e^{-10x} \tag{8}$$

the exact solution is:

$$u(x) = 1 - (1 - e^{-10})x - e^{-10x} \tag{9}$$

2.2 Guassian elimination

Guassian elimination is an efficient method for solving these types of equations. The matrix is here transformed into the identity matrix, while a new vector $\tilde{d}$ is created with the solution.

The elimination is here shown with a generalized 4x4 matrix, but the method works for any $n \times n$ matrix.

$$\begin{bmatrix} a_1 & b_1 & 0 & 0 \\ c_1 & a_2 & b_2 & 0 \\ 0 & c_2 & a_3 & b_3 \\ 0 & 0 & c_3 & a_4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} \tag{10}$$

2.2.1 Forward substitution

To remove the elements below the diagonal in the matrix, the method of forward substitution is used. The first operation is to subtract $\frac{c_1}{a_1}$ times the first row. This will create an extended matrix looking like this:

$$\left[\begin{array}{cccc|c} a_1 & b_1 & 0 & 0 & d_1 \\ c_1 & a_2 & b_2 & 0 & d_2 \\ 0 & c_2 & a_3 & b_3 & d_3 \\ 0 & 0 & c_3 & a_4 & d_4 \end{array}\right] \sim \left[\begin{array}{cccc|c} a_1 & b_1 & 0 & 0 & d_1 \\ 0 & a_2 - \frac{c_1 b_1}{a_1} & b_2 & 0 & d_2 - \frac{d_1 c_1}{a_1} \\ 0 & c_2 & a_3 & b_3 & d_3 \\ 0 & 0 & c_3 & a_4 & d_4 \end{array}\right] \tag{11}$$

These operations can be generalized by introducing the new variables

$$\tilde{a}_i = a_i - \frac{b_{i-1} c_{i-1}}{\tilde{a}_{i-1}} \tag{12}$$

$$\tilde{d}_i = d_i - \frac{\tilde{d}_{i-1}c_{i-1}}{\tilde{a}_{i-1}} \tag{13}$$

Continuing the row reduction with the forward substitution starting with the lowest index of the c-diagonal will yield the following equation:

$$\begin{bmatrix} \tilde{a}_1 & b_1 & 0 & 0 \\ 0 & \tilde{a}_2 & b_2 & 0 \\ 0 & 0 & \tilde{a}_3 & b_3 \\ 0 & 0 & 0 & \tilde{a}_4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} \tilde{d}_1 \\ \tilde{d}_2 \\ \tilde{d}_3 \\ \tilde{d}_4 \end{bmatrix} \tag{14}$$

with the boundary conditions $\tilde{a}_1 = a_1$ and $\tilde{d}_1 = d_1$

The two formulas in the forwards substitution requires three floating point operations each, yielding a total of $6(n-1)$ FLOPS.

2.2.2 Backward substitution

To remove the elements above the diagonal in the matrix, the method of backward substitution is used. As opposed to the forward substitution, we start with the highest index of the b-diagonal. The first operation is to remove b_3 by subtracting the third row with $\frac{b_3}{\tilde{a}_4}$ times the fourth row.

$$\left[\begin{array}{cccc|c} \tilde{a}_1 & b_1 & 0 & 0 & \tilde{d}_1 \\ 0 & \tilde{a}_2 & b_2 & 0 & \tilde{d}_2 \\ 0 & 0 & \tilde{a}_3 & b_3 & \tilde{d}_3 \\ 0 & 0 & 0 & \tilde{a}_4 & \tilde{d}_4 \end{array}\right] \sim \left[\begin{array}{cccc|c} \tilde{a}_1 & b_1 & 0 & 0 & \tilde{d}_1 \\ 0 & \tilde{a}_2 & b_2 & 0 & \tilde{d}_2 \\ 0 & 0 & \tilde{a}_3 & 0 & \tilde{d}_3 - \frac{b_3\tilde{d}_4}{\tilde{a}_4} \\ 0 & 0 & 0 & \tilde{a}_4 & \tilde{d}_4 \end{array}\right] \tag{15}$$

Repeating the procedure on the other rows will remove b_2 and b_3 and yield the following matrix:

$$\left[\begin{array}{cccc|c} \tilde{a}_1 & 0 & 0 & 0 & \tilde{d}_1 - \frac{b_1}{\tilde{a}_3}(\tilde{d}_2 - \frac{b_2}{\tilde{a}_3}(\tilde{d}_3 - \frac{b_3\tilde{d}_4}{\tilde{a}_4})) \\ 0 & \tilde{a}_2 & 0 & 0 & \tilde{d}_2 - \frac{b_2}{\tilde{a}_3}(\tilde{d}_3 - \frac{b_3\tilde{d}_4}{\tilde{a}_4}) \\ 0 & 0 & \tilde{a}_3 & 0 & \tilde{d}_3 - \frac{b_3\tilde{d}_4}{\tilde{a}_4} \\ 0 & 0 & 0 & \tilde{a}_4 & \tilde{d}_4 \end{array}\right] \tag{16}$$

Putting this new matrix into the linear equation, the following relations can be found:

$$u_4 = \frac{\tilde{d}_4}{\tilde{a}_4} \tag{17}$$

and

$$u_3 = \frac{\tilde{d}_3 - b_3 u_4}{\tilde{a}_3} \tag{18}$$

Generalizing this to a matrix with i-rows, the following formula for u_i can be found:

$$u_i = \frac{\tilde{d}_i - b_i u_{i+1}}{\tilde{a}_i} \tag{19}$$

This formula requires three floating point operations, yielding a total of $3(n-1)$ FLOPS. This means that the full Guassian elimination requires $9(n-1)$ FLOPS.

2.2.3 Numerical implementation of the Gaussian elimination

Listing 1: Implementing the Gaussian elimination algorithms in Python

```
from brg.datastructures import Mesh

mesh = Mesh.from_obj('faces.obj')
mesh.draw()
```

2.2.4 Gaussian elimination of a tridiagonal matrix

The fact that the matrix (7) that will be used in this problem is a tridiagonal matrix with two of the diagonals being -1, can simplify our algorithms and save some FLOPS.

Updating the Guassian elimination formulas with this special case, yields the following new equations:

$$\tilde{a}_i = \tilde{a}_i + \frac{b_{i-1}}{\tilde{a}_{i-1}} \tag{20}$$

which can be further simplified to:

$$\tilde{a}_i = \frac{i+1}{i} \tag{21}$$

$$\tilde{d}_i = d_i + \frac{\tilde{d}_{i-1}}{\tilde{a}_{i-1}} \tag{22}$$

$$u_i = \frac{\tilde{d}_i + u_{i+1}}{\tilde{a}_i} \tag{23}$$

The new set of equations will require $4(n-1)$ FLOPS, which is a great reduction in the required computation power. Equation (21) can be pre-calculated outside the algorithm and will therefore not add more FLOPS.

2.2.5 Numerical implementation of the special case

Listing 2: Implementing the Gaussian elimination algorithms in Python for the special case

```
from brg.datastructures import Mesh

mesh = Mesh.from_obj('faces.obj')
mesh.draw()
```

2.3 LU-decomposition

Decomposing a matrix in one lower triagonal and one upper triagonal matrix is called LU-decomposition. This can be done to split a matrix equation into to new equations which makes the problem easier to solve. The matrix equation $A\mathbf{u} = \mathbf{v}$ will then read:

$$LU\mathbf{u} = \mathbf{v} \tag{24}$$

After introducing $\mathbf{y} = U\mathbf{u}$, the solution can be computed. First $\mathbf{y}$ is found from:

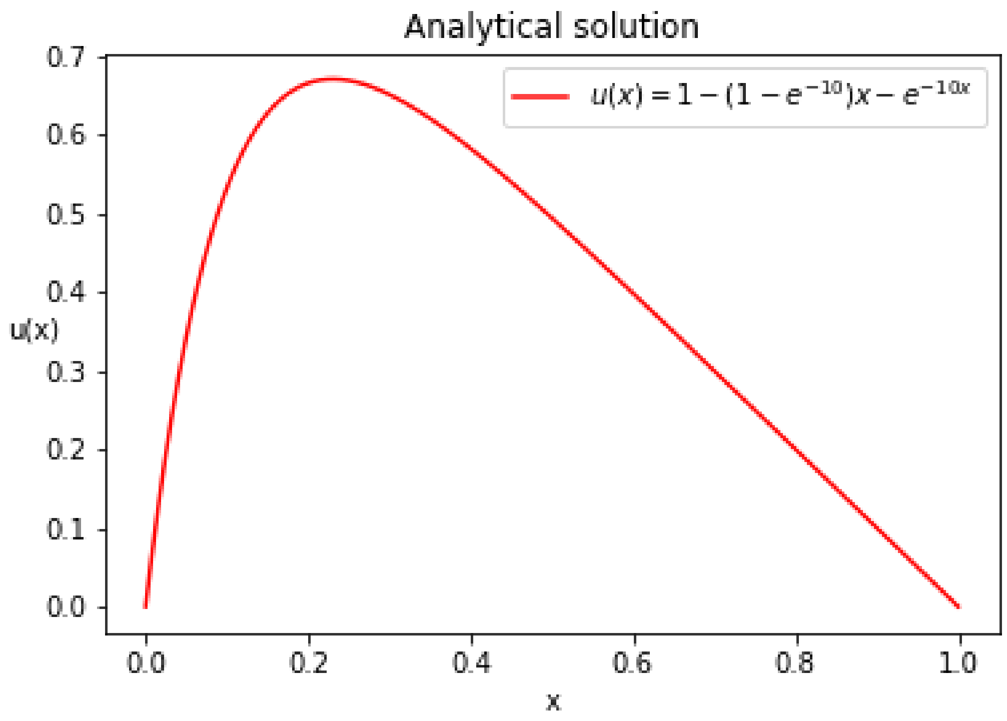
$$L\mathbf{y} = \mathbf{v} \tag{25}$$

Then,

$$U\mathbf{u} = y \tag{26}$$

can be solved to find $\mathbf{u}$.

Figure 1: Analytical solution



3 Method and results

3.1 Plot of the analytical solution

The analytical solution was plotted to get a feeling of what the numerical solution would look like. It can be seen in figure (1).

3.2 Implementation of general solver

A Python-program for the general solver was made, and the second derivative for $f(x) = 100e^{-10x}$ was calculated and compared with the analytical results for $n = 10, 100, 1000$. The calculation times were calculated and can be found in table (**??**).

Table 1: Computation time for the different algorithms and grid points.

n	General [s]	Special [s]	LU [s]
10^1	0.000033	0.000029	
10^2	0.000382	0.000210	
10^3	0.004647	0.001499	
10^4	0.036191	0.018247	
10^5	0.298454	0.148838	
10^6	2.787917	1.408053	

Table 2: The maximum errors for the different algorithm and grid points.

n	General	Special	LU
10^1	0.616569	0.258242	
10^2	0.060437	0.015616	
10^3	0.006005	0.001955	

4 Discussion and conclusion

References

[1] Morten Hjorth-Jensen, Computational Physics, August 2015