

FYS3150-Project 1

Marcus Berget, Sebastian Amundsen, Andreas Wetzel

August 2020

Abstract

1 Introduction

In this project we will use numerical methods to solve the one dimensional Poisson equation. We will be rewriting the Dirichlet boundary conditions as a set of linear equations. Our numerical methods will have varying degrees of accuracy. We are going to compare our algorithms and the CPU time for the different numerical methods.

2 Method

The one dimensional Poisson equation with Dirichlet boundary conditions is given by:

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

Let's define the discretized approximation to $u(x)$ as v_i , and to $f(x)$ as f_i . We can then make an approximation of the second derivative of u given by:

$$\frac{-v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i, \text{ where } i = 1, 2, 3, \dots, n, \quad (2)$$

Here the step length of spacing is given by $h = 1/(n+1)$ and the grid points is defined by $x_i = ih$. We can rewrite equation 2 as a linear set of

equations on the form $\mathbf{A}\mathbf{v} = \tilde{\mathbf{b}}$:

$$\begin{aligned} \mathbf{A}\mathbf{v} &= \begin{bmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots & & \\ 0 & \vdots & \vdots & -1 & 2 & -1 \\ 0 & \vdots & \vdots & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_n \\ v_{n+1} \end{bmatrix} = \begin{bmatrix} 2v_0 - v_1 \\ -v_0 + 2v_1 - v_2 \\ -v_1 + 2v_2 - v_3 \\ \vdots \\ -v_{n-1} + 2v_n - v_{n+1} \\ -v_n + 2v_{n+1} \end{bmatrix} \\ &= \begin{bmatrix} h^2 f_0 \\ h^2 f_1 \\ h^2 f_2 \\ \vdots \\ h^2 f_n \\ h^2 f_{n+1} \end{bmatrix} = \tilde{\mathbf{b}} \end{aligned} \quad (3)$$

In our case we will use the source term $f(x) = 100e^{-10x}$ which gives us a closed form solution $u(x)$ given by:

$$u(x) = 1 - (1 - e^{-10})x - e^{-10x} \quad (4)$$

We will use this solution as a point of reference to discuss the accuracy of our numerical methods.

2.1 General algorithm for tri-diagonal matrix

There exists an algorithm for solving generic sets of linear equations, but in the case for equation (3) where we have a tri-diagonal matrix we can use a different algorithm which decreases the amount of floating point operations needed. We denote the elements in the leading diagonal of \mathbf{A} as b_1, b_2, \dots, b_n , the elements above the leading diagonal as a_2, a_3, \dots, a_n , and the elements below the leading diagonal as c_1, c_2, \dots, c_{n-1} . The algorithm uses a forward substitution to replace the leading diagonal with elements denoted by \tilde{b}_i , and replaces the righthand side in with the elements \tilde{f}_i as shown below:

$$\begin{aligned} \tilde{b}_i &= b_i - \frac{a_i c_{i-1}}{\tilde{b}_{i-1}} \\ \tilde{f}_i &= f_i - \frac{a_i \tilde{f}_{i-1}}{\tilde{b}_{i-1}} \end{aligned}$$

where $\tilde{b}_1 = b_1$ and $\tilde{f}_i = f_i$. The algorithm then continues with a backward substitution which gives the solution:

$$u_{i-1} = \frac{\tilde{f}_{i-1} - c_{i-1}u_i}{\tilde{b}_{i-1}} \quad (5)$$

2.2 Algorithm for specific tri-diagonal matrix

In our special case we can implement a solver that is even simpler than what is described previously. We will exploit the fact that the matrix has identical matrix elements along the diagonal and identical values for the non diagonal elements \tilde{e}_i . In this case we can precalculate the new values for the updated matrix elements d_i without taking into account the values for \tilde{e}_i :

$$d_i = 2 - \frac{1}{\tilde{d}_{i-1}} = \frac{i+1}{i} \quad (6)$$

Here the initial value is $\tilde{d}_1 = 2$. The new righthand side solution \tilde{f}_i is given by:

$$\tilde{f}_i = f_i + \frac{(i-1)\tilde{f}_{i-1}}{i} \quad (7)$$

Here the initial value is $\tilde{f}_1 = f_1$. The last step is to make a backward substitution which gives the final solution u_i :

$$u_{i-1} = \frac{i-1}{i}(\tilde{f}_{i-1} + \tilde{u}) \quad (8)$$

This method requires that we know the last value u_n in the u_i array. This value is given by $u_n = \tilde{f}_n / \tilde{b}_n$.

2.3 Relative error

Algorithms have a varying degree of uncertainty. We will test how precise our algorithm is for the specific tri-diagonal matrix case. The numerical solution will be compared to the analytical solution given the relative error ϵ_i :

$$\epsilon_i = \log_{10} \left(\left| \frac{v_i - u_i}{u_i} \right| \right) \quad (9)$$

For the numerical v_i and analytical u_i function values. We wish to extract the max value of the relative error for varying numbers of grid points n . This will give us some information about how precise the numerical approximation is compared to the analytical solution.

2.4 LU decomposition

We are now going to LU decomposition the matrixes 10×10 , 100×100 , 1000×1000 and $10\,000 \times 10\,000$. What the LU decomposition does, is that we can rewrite a matrix as the product of two other matrices L and U, like this:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ l_{n1} & l_{n2} & \dots & 1 \end{bmatrix}$$

There are several reasons why we use LU decomposition instead of standard Gaussian elimination. First of all, it is straight forward to solve the determinant of a matrix. Second, if we still need to solve a set of linear equation with the same matrix, but with a different vector, the number of FLOPS is of order n^3 . Where FLOPS is floating point operations per second. Where a such operation is the inverse.

3 Implementation

All programs used is available at:

https://github.com/Sebamun/FYS3150_Projekter

We implement the algorithms numerically with varying values of grid points n .

4 Results

4.1 General algorithm

4.2 Algorithm for specific tri-diagonal matrix

4.3 Relative error

4.4 LU decomposition

5 Discussion

6 Concluding remarks

References