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 general one dimensional Poisson equation is given by:

$$-u''(x) = f(x) \tag{1}$$

Where $u = v_i$ (discretized approximation) and $f(x)$ are functions. We have the interval $x \in (0,1)$ and the initial conditions $u(0) = u(1) = 0$. We have an approximation of the second derivative of u given by:

$$\frac{-v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i, \text{ where } i = 1, 2, 3......, n, \tag{2}$$

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 $\mathbf{Av} = \tilde{\mathbf{b}}$. In our case we have the source term $f(x) = 100e^{-10x}$ which gives us a closed form solution $u(x)$ given by:

$$u(x) = 1 - (1 - e^{-10})x - e^{-10x} \tag{3}$$

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

by multiplying **A** and **v** we get

$$\begin{bmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots & & \\ 0 & \vdots & \vdots & -1 & 2 & -1 \\ 0 & \vdots & \vdots & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_n \\ v_{n+1} \end{bmatrix} = \begin{bmatrix} 2v_0 - v_1 \\ -v_0 + 2v_1 - v_2 \\ -v_1 + 2v_2 - v_3 \\ \vdots \\ -v_{n-1} + 2v_n - v_{n+1} \\ -v_n + 2v_{n+1} \end{bmatrix}$$

$$= \begin{bmatrix} h^2 f_0 \\ h^2 f_1 \\ h^2 f_2 \\ \vdots \\ h^2 f_n \\ h^2 f_{n+1} \end{bmatrix} = \widetilde{\mathbf{b}}$$

2.1 General algorithm

Hello world! Hello world! again.. Bye, world

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 $\vec{e}_i$. In this case we can precalculate the new values for the updated matrix elements d_i without taking into account the values for $\vec{e}_i$:

$$d_i = 2 - \frac{1}{\tilde{d}_{i-1}} = \frac{i+1}{i} \tag{4}$$

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

$$\tilde{f}_i = f_i + \frac{(i-1)\tilde{f}_{i-1}}{i} \tag{5}$$

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:

$$u_{i-1} = \frac{i-1}{i}(\tilde{f}_{i-1} + \tilde{u}) \tag{6}$$

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:

$$\epsilon_i = \log_{10}\left(\left|\frac{v_i - u_i}{u_i}\right|\right) \tag{7}$$

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 x 10, 100 x 100, 1000 x 1000 and 10 000 x 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:

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

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