# FYS3150 - PROJECT 1 - AUTUMN 2015

Mari Dahl Eggen

October 10, 2015

**Abstract**

In this project we will develop an algorithm to solve a matrix equation where the matrix is tridiagonal. The goal is to develop an algorithm that has fewer floating point operations that the algorithms for Gaussian Elimination and LU-decomposition, so that the calculations are running faster, and so that we can do the calculations for bigger systems. Through different tests described in this report, we find that the customized algorithm was a success.

## Introduction

Some times it pays of to develop algorithms for special cases in the sciences. The reason for that is that the calculations on a computer can get very slow if the number of floating point operations is getting too big. In addition, the more floating point operations, the greater is the probability of round off errors in the calculations. If a customized algorithm is made, the floating point operations can be cut down to a minimum of what is needed, and then the algorithm can solve grater systems of the kind it is customized to solve. In this project an algorithm customized to solve matrix equations, where the matrix is tridiagonal, is developed. The algorithm is based on the method of Gaussian elimination, and we will through different tests show that the customized algorithm is both working as that method, and that it is faster. In addition we will find the optimal number of grid-point to execute the algorithm with, so that the relative error of the numerical solution is getting as small as possible.

# Theory

## Poisson equation

Poisson's equation is a classical equation from electromagnetism, and with spherical symmetrical properties it reads

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

where $\Phi$ is a electrostatic potential generated by the localized charge distribution $\rho(\mathbf{r})$. This equation can be expressed in a more simple way by use of the substitution $\Phi(r) = \frac{\phi(r)}{r}$ as

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

A standard linear second-order differential equation reads

$$\frac{d^2u}{dx^2} + k^2(x)u = f(x),$$

where $f(x)$ is the called inhomogenous term and $k^2(x)$ is a real function. If we let $\phi \rightarrow u$ and $r \rightarrow x$, sets $k^2(x) = 0$, and sets $f(x) = -4\pi\rho(x)$, the one-dimensional Poisson equation can be expressed as

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

One way to solve Equation (1) is to rewrite it as a set of linear equations, where we define the discretized approximation of $u$ as $v_i$. The grid points are set to $x_i = ih$ in the interval $[x_0, x_{n+1}]$. $u''(x)$ can then be approximated as

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i \quad \text{for } i = 1, \ldots, n, \tag{2}$$

where $f_i = f(x_i)$. In such a problem we need to have some boundary conditions. One set of those is called Dirichlet boundary conditions, and reads

$$x \in (0,1), \quad u(0) = u(1) = 0. \tag{3}$$

## Gaussian elimination

Gaussian elimination is a method of solving a linear set of equations, which can be represented as the matrix equation

$$\mathbf{A}\bar{\mathbf{v}} = \bar{\mathbf{b}} \tag{4}$$

$$\Rightarrow \quad \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{n,1} & \dots & \dots & a_{n,n} \end{pmatrix} \begin{pmatrix} v_1 \\ \vdots \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ \vdots \\ b_n \end{pmatrix}$$

$$\Rightarrow \quad \begin{array}{c} a_{1,1}v_1 + a_{1,2}v_2 + \dots + a_{1,n}v_n = b_1 \\ a_{2,1}v_1 + a_{2,2}v_2 + \dots + a_{2,n}v_n = b_2 \\ \vdots \\ a_{n,1}v_1 + a_{n,2}v_2 \dots + a_{n,n}v_n = b_n, \end{array}$$

in the case of a quadratic nxn-matrix. We will look at the simplest case where we assume that the matrix **A** is non-singular, and that the matrix elements $a_{i,i} \neq 0$. In this method the aim is to eliminate the unknowns from the set of equations, in a systematical way, so that the set of equations takes the form

$$\begin{array}{c} a_{1,1}v_1 + a_{1,2}v_2 + \dots + a_{1,n}v_n = b_1 \\ a_{2,2}v_2 + a_{2,3}v_3 + \dots + a_{2,n}v_n = b_2 \\ a_{3,3}v_3 + a_{3,4}v_4 + \dots + a_{3,n}v_n = b_3 \\ \vdots \\ a_{n,n}v_n = b_n. \end{array} \tag{5}$$

One can reach that form by transforming the the desired coefficients in the matrix **A**, one by one, to zero. The method for doing this is operations done on the set of equations, and is called forward substitution. The formula for the left hand side of the equations is

$$a_{j,k}^{(m+1)} = a_{j,k}^{(m)} - \frac{a_{j,m}^{(m)} \cdot a_{m,k}^{(m)}}{a_{m,m}^{(m)}}, \quad j,k = m+1,...,n \quad \text{and} \quad m = 1,...,n-1, \tag{6}$$

and the formula for the right hand side is

$$b_j^{(m+1)} = b_j^{(m)} - \frac{a_{j,m}^{(m)} \cdot b_m^{(m)}}{a_{m,m}^{(m)}}, \quad j = m+1, ..., n \quad \text{and} \quad m = 1, ..., n-1. \tag{7}$$

The set of equations (5) is easy to solve. The method of doing it is called backward substitution, and the formula for doing it is

$$v_m = \frac{1}{a_{m,m}} \left( b_m - \sum_{k=m+1}^{n} a_{m,k} \cdot v_k \right), \quad m = n-1, n-2, ..., 1. \tag{8}$$

## LU-decomposition

This method is a form of Gaussian elimination where the starting point is a factorization of the the matrix **A** in Equation (4). **A** is factorized as two matrices called $L$ and $U$, where the names comes from the fact that the matrices are lower and upper diagonal. Thus we have

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

$$\Rightarrow \quad \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{n,1} & \dots & \dots & a_{n,n} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ l_{2,1} & 1 & & \vdots \\ \vdots & \ddots & \ddots & 0 \\ l_{n,1} & \dots & l_{n,n-1} & 1 \end{pmatrix} \begin{pmatrix} u_{1,1} & u_{1,2} & \dots & u_{1,n} \\ 0 & u_{2,2} & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \dots & 0 & u_{n,n} \end{pmatrix}.$$

The quadratic matrix **A** has a LU factorization if its determinant is different from zero, and the factorization is unique if **A** is non-singular. The LU factorization can be found by use of the two equations

$$i \le j: \quad l_{i,1}u_{1,j} + l_{i,2}u_{2,j} + ... + l_{i,i}u_{i,j} = a_{i,j}$$

$$i > j: \quad l_{i,1}u_{1,j} + l_{i,2}u_{2,j} + ... + l_{i,j}u_{j,j} = a_{i,j}.$$

Now the problem of solving Equation (4) has evolved to be the problem of solving the equation

$$\mathbf{LU}\bar{\mathbf{v}} = \bar{\mathbf{b}}. \tag{9}$$

It can be shown that Equation (10) can be split into two equations on the form

$$\mathbf{L}\bar{\mathbf{y}} = \bar{\mathbf{b}} \qquad \mathbf{U}\bar{\mathbf{v}} = \bar{\mathbf{y}}. \tag{10}$$

$$\Rightarrow \qquad \begin{gathered} y_1 = b_1 \\ l_{2,1}y_1 + y_2 = b_2 \\ l_{3,1}y_1 + l_{3,2}y_2 + y_3 = b_3 \\ \vdots \\ l_{n,1}y_1 + l_{n,2}y_2 + \cdots + l_{n,n}y_n = b_n \end{gathered} \qquad \begin{gathered} u_{1,1}v_1 + u_{1,2}v_2 + \cdots + u_{1,n}v_n = y_1 \\ u_{2,2}v_2 + u_{2,3}v_3 + \cdots + u_{2,n}v_n = y_2 \\ u_{3,3}v_3 + u_{3,4}v_4 + \cdots + u_{3,n}v_n = y_3 \\ \vdots \\ u_{n,n}v_n = y_n. \end{gathered}$$

These equations can be solved by use of backward substitution, that was discussed in the previous section. First to find $\bar{\mathbf{y}}$, and then to find $\bar{\mathbf{v}}$.

## Relative error

The relative error of a computed data set $i = 1, ..., n$ can be calculated by use of the formula

$$\epsilon_i = log_{10}\left(\left|\frac{v_i - u_i}{u_i}\right|\right), \tag{11}$$

where $v_i$ is the elements in the data set for the numerical solution and $u_i$ is the elements in the data set for the analytically solution.

## Method

To show that Equation (2) can be written as a linear set of equations on the form

$$\mathbf{A}\bar{\mathbf{v}} = \bar{\mathbf{b}}, \tag{12}$$

where $\mathbf{A}$ is the $n \times n$ tridiagonal matrix

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \vdots \\ 0 & -1 & 2 & -1 & 0 & \vdots \\ \vdots & \dots & \dots & \dots & \dots & \vdots \\ 0 & \dots & \dots & -1 & 2 & -1 \\ 0 & \dots & \dots & 0 & -1 & 2 \end{pmatrix} \tag{13}$$

and $b_i = h^2 f_i$, we will use Equation (12) as the starting point.

$$\begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \vdots \\ 0 & -1 & 2 & -1 & 0 & \vdots \\ \vdots & \dots & \dots & \dots & \dots & \vdots \\ 0 & \dots & \dots & -1 & 2 & -1 \\ 0 & \dots & \dots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} v1 \\ \vdots \\ v_{i-1} \\ v_i \\ v_{i+1} \\ \vdots \\ v_n \end{pmatrix} = h^2 \begin{pmatrix} f1 \\ \vdots \\ f_{i-1} \\ f_i \\ f_{i+1} \\ \vdots \\ f_n \end{pmatrix}$$

$$\Downarrow$$

$$2v_1 - v_2 = h^2 f_1$$

$$-v_1 + 2v_2 - v_3 = h^2 f_2$$

$$-v_2 + 2v_3 - v_4 = h^2 f_3$$

$$\vdots$$

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

$$\Downarrow$$

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

In the folloiwng we will assume that $f(x) = 100e^{-10x}$, and we will use the boundary conditions given in Equation (3). Then Equation (1) has the closed-form solution $u(x) = 1 - (1 - e^{-10})x - e^{-10x}$. To test if this is the right solution we take the derivative of $u''(x)$ twice.

$$u'(x) = -(1 - e^{-10})x - e^{-10x}$$
$$\Downarrow$$
$$-u''(x) = 100e^{-10x} = f(x)$$

The analytical solution $u(x)$ is correct, and it can be used as a comparison to a numerical solution.

To find the numerical solution of Equation (1) one can use the method Gaussian elimination or LU-decomposition, which are discussed in the theory section. Before we use one of the two methods we have to take a good look at the matrix equation that has to be solved. In this case we have shown that the matrix $\mathbf{A}$ is a tridiagonal matrix, and we then realize that the problem can be solved in a more efficient way. A general tridiagonal matrix can be expressed on the form

$$\mathbf{A} = \begin{pmatrix} b_1 & c_1 & 0 & \dots & \dots & 0 \\ a_2 & b_2 & c_2 & 0 & \dots & \vdots \\ 0 & a_3 & b_3 & c_3 & 0 & \vdots \\ \vdots & \dots & \dots & \dots & \dots & \vdots \\ 0 & \dots & \dots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & \dots & \dots & 0 & a_n & b_n \end{pmatrix}.$$

The set of equations derived from the matrix equation (4) is then

$$a_i v_{i-1} + b_i v_i + c_i v_{i+1} = b_i. \tag{14}$$

Now we can use the expression (14) to make an algorithm based on the formulas for forward substitution (6) and (7), and backward substitution (8). This will end up to be a more efficient variant of the Gaussian elimination method, because we will ignore all the matrix elements that we know is zero.

The update of the coefficients in the tridiagonal matrix then goes like

$$a_i = a_i - \frac{a_i \cdot b_i}{b_i} = 0, \quad b_i = b_i - \frac{a_{i-1} \cdot c_{i-1}}{b_{i-1}}, \quad c_i = c_i - \frac{a_{i-1} \cdot 0}{b_{i-1}} = c_i,$$

and the update of the right hand side of the equation goes like

$$f_i = f_i - \frac{a_{i-1} \cdot f_{i-1}}{b_{i-1}}.$$

Then we can find the unknowns by backward substitution, and the algorithm in this case goes as

$$v_i = \frac{f_i - a_i \cdot v_{i-1} - c_i \cdot v_{i+1}}{b_{i-1}}$$

We can take the simplifications further by use of the fact that $a_i = c_i = 1$ (when the minus sign is factorized out), and that the factor $\frac{a_{i-1}}{b_{i-1}} = \frac{1}{b_{i-1}}$ is found in the formula for both $b_i$ and $f_i$. In addition, all the elements $a_i = 0$ when we comes to the backward substitution, and so we get rid of the $a_i$-part in that algorithm. The code-snippet for forward substitution and backward substitution is listed in Listing 1. In the algorithm $v_i$ ($u[i]$) has been pushed one element ahead, the reason for that is that the vector $u$ in the code is two elements longer because of the boundary conditions.

In this project we will count floating point operations in the way that addition, subtraction, multiplication and division, counts as one floating point operation each. In addition to count floating point operations, the time it takes to execute the customized algorithm for forward and backward substitution, is compared with the time it takes to execute the same job with LU-decomposition. We use the functions ludcmp() and lubksb() from the library lib.cpp to calculate the numerical solution by LU-decomposition, and uses the package time.h to take the time of the executions.

Listing 1: Forward and backward substitution for the tridiagonal matrix (13).

```
for(i=1;i<=(n-1);i++)
{
    //forward substitution
    double factor = (1.0/b[i-1]);
    b[i] = b[i] - factor;
    y[i] = y[i] - (factor*y[i-1]);
    a[i-1] = 0;
}
for(i=n;i>=1;i--)
{
    //fill u(x) by backward substitution
    u[i] = (y[i-1]-u[i+1])/b[i-1];
}
```

When the calculations by use of the algorithm in Listing 1 is executed, the relative error of the numerical solution is found by use of Equation (11). We will check the relative error for grid-points in the range $n = 10$ to $n = 10^6$, to find the $n$ that gives the least error. The error is of course getting smaller for bigger $n$, but at some point the step lengths are getting so small that round off errors in the calculations will appear.

## Results and discussion

### Floating point operations and execution time

For the method in Listing 1 there is $6n$ floating point operations, where $n$ is the size of the quadratic matrix, and so also the number of unknown parameters. In Gaussian elimination there is needed $\frac{2}{3}n^3$ floating point operations to find the answer, and by use of LU-decomposition there is needed floating point operations in magnitude of $n^2$. Then we know that the customized algorithm for the case we are looking at, is the most efficient of those three.

We have also measured the time that the execution of the algorithm in Listing 1 and the LU-decomposition takes. The results is listed in Table 1.

| **Log10(h)** | **Algorithm Listing (1) [sec]** | **LU-decomposition [sec]** |
|---|---|---|
| $-1$ | $3.0 \cdot 10^{-6}$ | $8.4 \cdot 10^{-5}$ |
| $-2$ | $4.0 \cdot 10^{-6}$ | $1.6 \cdot 10^{-3}$ |
| $-3$ | $2.7 \cdot 10^{-5}$ | $2.7$ |

Table 1: Execution time for the algorithm in Listing 1 and LU-decomposition.

Here we can see the result of the difference in floating point operations for the two methods. LU-decomposition is using much more time than the customized algorithm, and the time used is growing much faster for that method, as the grid-points in the calculation is increasing. Since the error for LU-decomposition is of the order $O(n^2)$ one can assume that the execution time is increasing as a parabola with $n$. Because of that the execution time for the LU-decomposition with $h = -4$ ($n = 10^4$) is going to be inconveniently long, and for $h = -5$ ($n = 10^5$) the algorithm is more or less useless.

## Numerical solution

The numerical solution from the customized algorithm is plotted in the figures 1, 2 and 3.

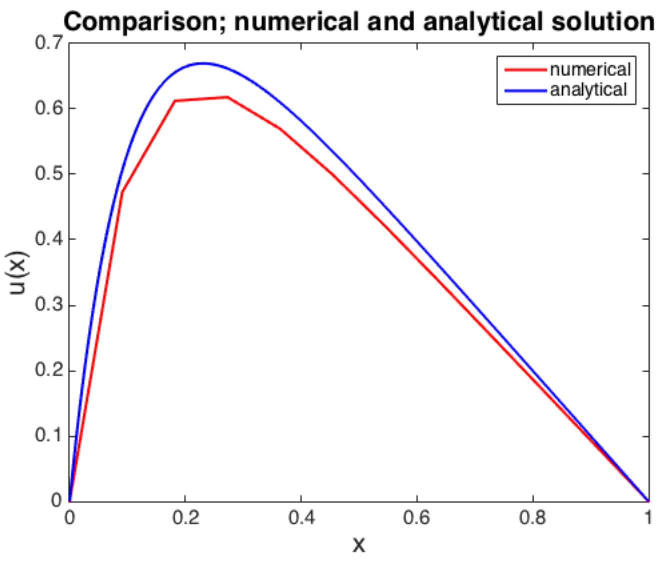
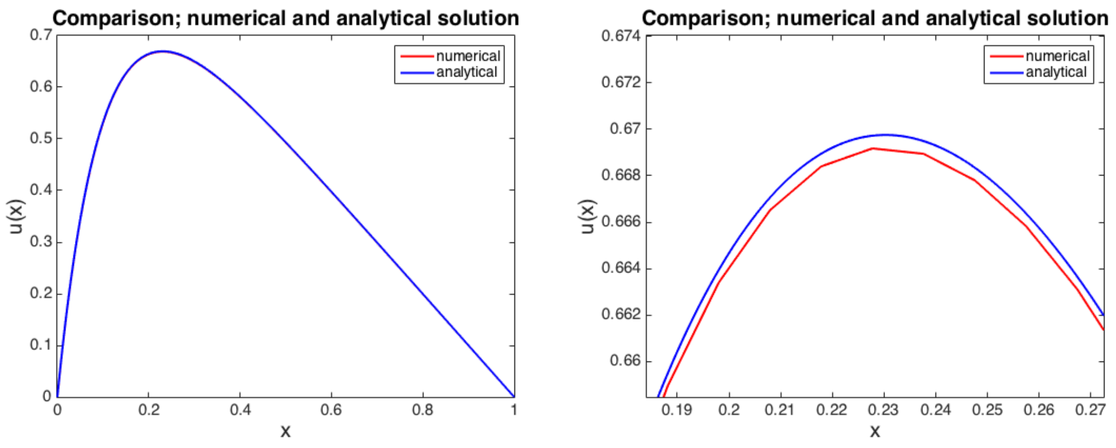
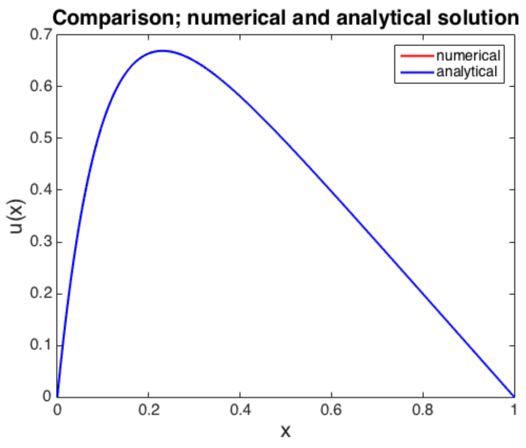


Figure 1: Plot of numerical solution together with the analytically solution for n = 10.

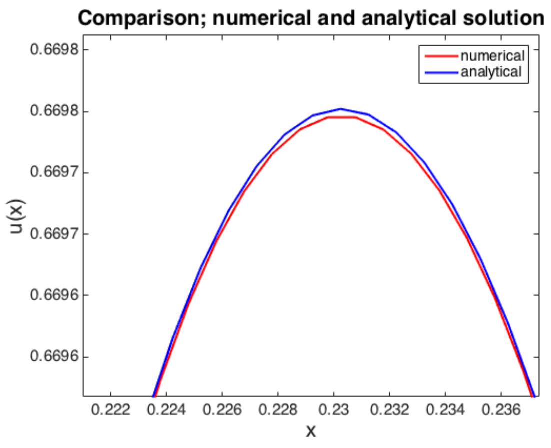


(a) Numerical and analytically solution (b) Numerical and analytically solution zoomed

Figure 2: Plot of numerical solution together with the analytically solution for n = 100.



(a) Numerical and analytically solution



(b) Numerical and analytically solution zoomed

Figure 3: Plot of numerical solution together with the analytically solution for n = 1000.

We can see that the numerical solution is getting drastically closer to the analytically solution when the grid points increases from $n = 10$ to $n = 100$. When the grid points is increased further, from $n = 100$ to $n = 1000$, the numerical solution is getting even closer to the analytically solution, but not as drastically as the previous change.

## Relative error

At last the relative error of the numerical calculations is found for different step lengths. You can see the results in Table 2. The plot of the relative error is in Figure 4. We can see that the relative error is dropping linearly with slope approximately equal to $-2$, as the step length is dropping. In the area where the step length is around $h = 10^{-5}$, we can see that the relative error is getting unstable, and it is here the round off errors are taking over. The result tells us that the step length $h = 10^{-5}$ is the most optimal step length if you want the least relative error, here the balance between the number of grid-points and the risk of round off errors is the best.

| **log10(h)** | **log10(Error)** |
|---|---|
| $-1$ | $-1.1796978$ |
| $-2$ | $-3.0880368$ |
| $-3$ | $-5.0800516$ |
| $-4$ | $-7.0793568$ |
| $-5$ | $-9.0048965$ |
| $-6$ | $-6.7713740$ |

Table 2: Relative error of numerically calculated solution.

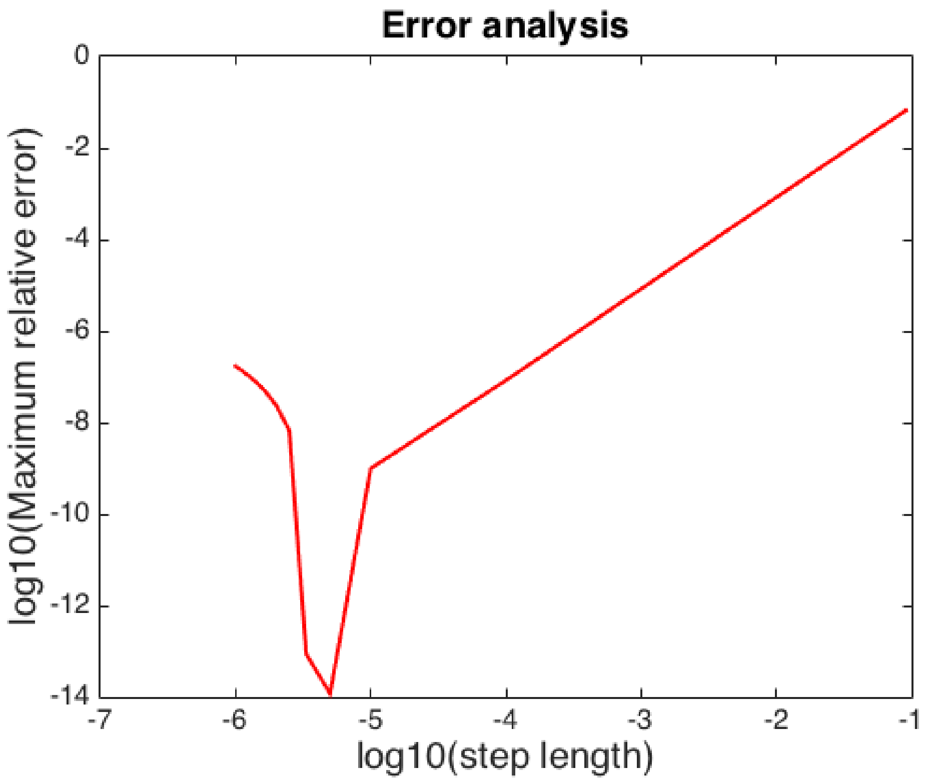


Figure 4: Plot of maximum relative error for the numerical solutions, found by use of the algorithm in Listing 1, for different numbers of grid-points (step lengths).