Project 1, FYS-4150

Kosar Nozari Mirarkolaei

October 3, 2018

Abstract

The goal for this project is to solve the general one-dimentional Poisson equation with two different numerical methods and compare with the exact analythical solution.

The two numerical methods used to solve the equations is forward/backward substitution and LU-decomposition. Both methods are using linear algebra to turn the problem into a set of many linear equations which can be represented by matrixes.

We will see that the execution time and relative error varies for the two methods and that increasing n gives smaller error up to a point. I use *python* for coding and you can see the python code and all related files in my git-hub address : `https://github.com/Kosarnm/FYS4150`

1 Theory

1.1 Poisson's equation

Many important differential equations in the Sciences can be written as linear second-order differential equations

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

where f is normally called the inhomogeneous term and k^2 is a real function. It is therefore of special interest to be able to solve these kinds of equations.

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

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

This can be simplified with a spherically symmetric Φ and $\rho(\mathbf{r})$ to:

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

which is a simple one-dimensional equation. Simplifying further via a substitution $\Phi(r) = \phi(r)/r$ the equation reads:

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

We rewrite this equation again by letting $\phi \rightarrow u$ and $r \rightarrow x$. Then general one-dimensional **Poisson equation** reads:

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

where the inhomogeneous term f (or source term) is given by the charge distribution ρ multiplied by r and the constant -4π.
To solve equation 1 we will use Dirichlet boundary conditions and rewrite the equation as a set of linear equations.

In specific we will solve:

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

where we assume that the the source term is given by $f(x) = 100e^{-10x}$. Then the above differential equation has a closed-form analytical solution given by:

$$u(x) = 1 - (1 - e^{-10})x - e^{-10x} \tag{2}$$

For the numerical methods we define the discretized approximation to u as v_i with grid points $x_i = ih$ in the interval from $x_0 = 0$ to $x_{n+1} = 1$. The step length or spacing is defined as $h = 1/(n+1)$. The boundary conditions gives $v_0 = v_{n+1} = 0$. We approximate the second derivative of u with

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

where $f_i = f(x_i)$.

We can rewrite this equation as a linear set of equations of the form

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

by rewriting equation 3 as

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

so that $\mathbf{A}$ is an $n \times n$ tridiagonal matrix which we write as

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{pmatrix} \tag{5}$$

and the left hand side is given by $\tilde{b}_i = h^2 f_i$. The total set of matrixes must then be given by:

$$\mathbf{A}\mathbf{v} = \begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \dots \\ v_{n-1} \\ v_n \end{pmatrix} = \begin{pmatrix} h^2 f_1 \\ h^2 f_2 \\ h^2 f_3 \\ \dots \\ h^2 f_{n-1} \\ h^2 f_n \end{pmatrix} = \begin{pmatrix} \tilde{b}_1 \\ \tilde{b}_2 \\ \tilde{b}_3 \\ \dots \\ \tilde{b}_{n-1} \\ \tilde{b}_n \end{pmatrix} = \tilde{\mathbf{b}} \tag{6}$$

1.2 Gaussian elimination

1.2.1 Forward and backward substitution

Given the following tridiagonal matrix:

1.2.2 LU decomposition

Every square matrix A can be decomposed into a product of a lower triangular matrix L and a upper triangular matrix U , as described in LU decomposition.

$$A = LU$$

$$L = \begin{pmatrix} l_{11} & 0 & 0 & \cdots & \cdots & 0 \\ l_{21} & l_{22} & 0 & 0 & \cdots & \cdots \\ l_{31} & l_{32} & l_{33} & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & \cdots & \cdots & l_{n-1,n-1} & 0 \\ 0 & \cdots & \cdots & \cdots & l_{n,n-1} & l_{nn} \end{pmatrix} \tag{7}$$

$$
U = \begin{pmatrix}
u_{11} & u_{12} & u_{13} & \dots & u_{1n-1} & u_{1n} \\
0 & u_{22} & u_{23} & \dots & u_{2n-1} & u_{2n} \\
0 & 0 & u_{33} & \dots & u_{3n-1} & u_{3n} \\
 & \vdots & & \ddots & \vdots & \\
0 & 0 & 0 & \dots & u_{n-1n-1} & u_{n-1n} \\
0 & 0 & 0 & \dots & 0 & u_{nn}
\end{pmatrix} \tag{8}
$$

Now A can be substitute with LU in equation 4.

$$
Av = LUv = \tilde{b} \tag{9}
$$

Now we have two set of linear equations:

$$
Uv = y \qquad Ly = \tilde{b} \tag{10}
$$

Number of flops using LU decomposition is:

$$
N_{LU} = \frac{2}{3}n^3 - 2n^2 = \mathcal{O}(\frac{2}{3}n^3) \tag{11}
$$

1.3 Relative Error

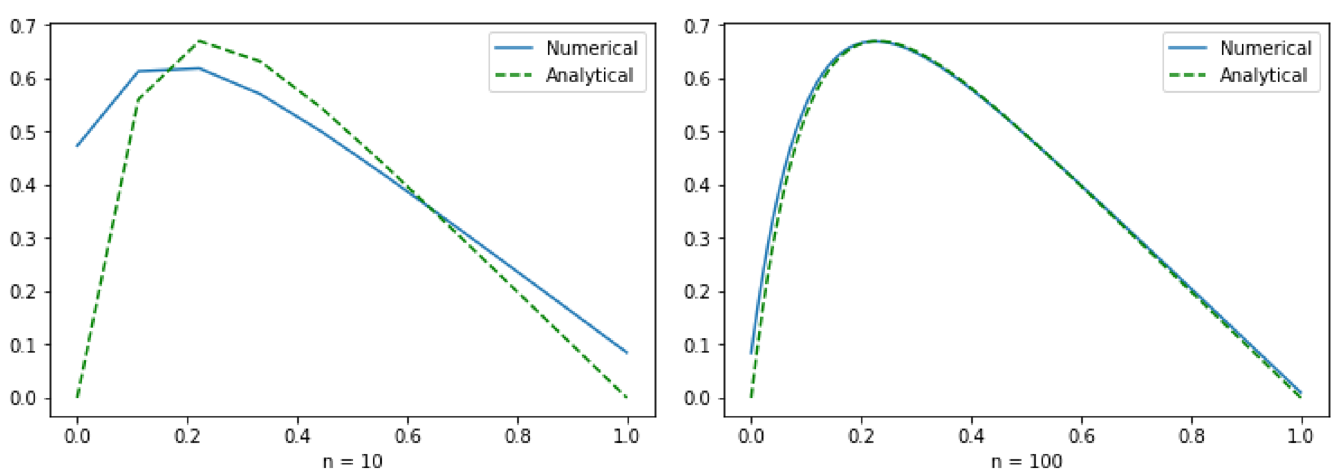
The relative error in the data set in relation to the analytical solution can be computed by

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

The error will be calculated as function of step lenghth h.

2 Results and discussions

Figure 1: Plots of the numerical and analytical solutions for $n = 10, 100$.



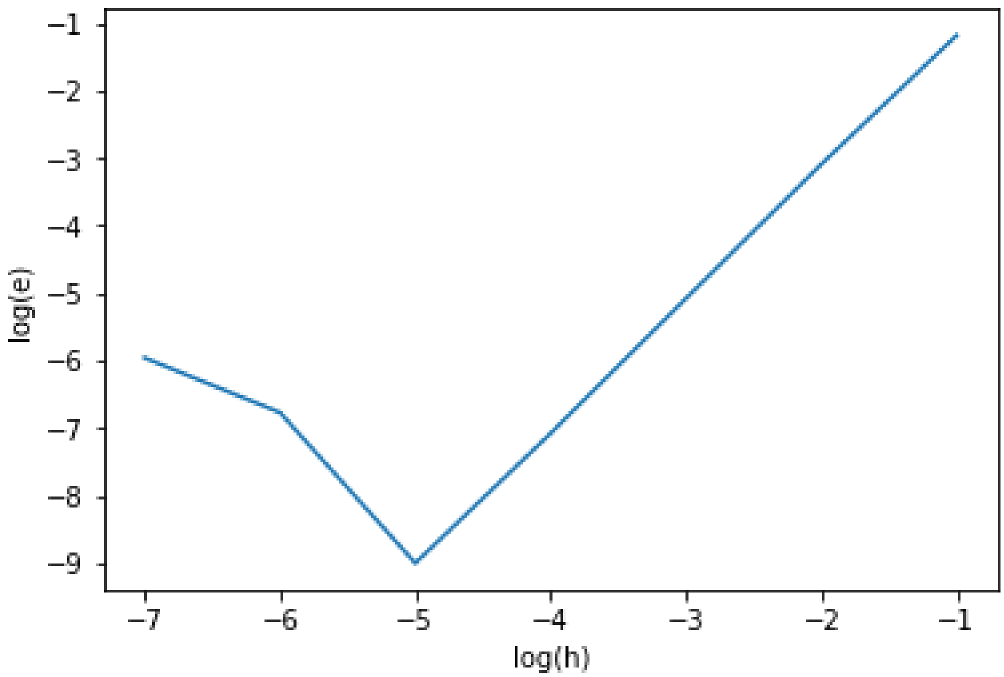


Figure 2: Relative Error

2.1 Comparing Numerical and Analytical Solutions

Numerical and Analytical solutions has been computed for $n = 10, 100$and 100. Figure one shows results for $n = 10$and100. Comparing these two plots shows that for large n, numerical and analytical solotions are getting closer.

2.2 Error Analysis

The relative error has been calculated for $n = 10, 10^2, 10^3, 10^4, 10^5, 10^6$and 10^7. Figure 2 shows the relative error with respect to $\log(h)$. It shows that for $n > 10^5$ the relative error increases. It happens due to accumulation of round off errors.

2.3 Timing Analysis

Table 1 shows the run time of different algorithm. It shows that Tridiagonal solver is faster than the LU decomposition solver. In addition, for $n > 10^3$ it is not possible to measure the run time because the computer has not enough memory.

3 Conclusion

As one might expect, for large number of n we have memory overflow the LU-decomposition method. As discussed, this is due to the nature of LU-decomposition which requires $O(n^3)$ floating point operations. Trying to execute the program with 10^5 grid points would then require 10^{15} FLOPS, in turn requiring a tremendous amount of memory in the computer. Also, we have shown that we can solve problem in higher precision and lower number of flops.

n	Tridiagonal solver	LU decomposition
10	0.000180056	0.0010015
10^2	0.00070021	0.001000165939
10^3	0.0650193	0.41002345085
10^4	0.45313572	MemoryError
10^5	4.2004196	MemoryError
10^6	25.109510	MemoryError

Table 1: Run time of different algorithm

Lower number of flops lead to a much faster run time.
We have also seen that large n lead to accumulation of round off error.