Project 1

Andrei Kukharenka and Anna Gribkovskaya
FYS 4150

September 22, 2016

Abstract

In this work we solve the one-dimensional Poisson equation with Dirichlet boundary conditions by rewriting it a set of linear equations. The latter was solved by some linear algebra methods: Gaussian elimination and LU decomposition. Both methods are applicable for such task, but differ in CPU time. Numerical precision is the same. Solution converges, except for the big matrix sizes where the convergence is destroyed by the round-off effects.

1 Introduction

In this project we are dealing with the Poisson equation. This is a second order inhomogeneous differential equation of elliptic type and we deal with its simple case in one dimension. We discretize this equation and rewrite it in form of system of linear equations and use some methods from linear algebra to find a numerical solution. This kind of equations is common in physics. For example it can describe an electrostatic or gravitational fields.
Below is a brief description of the project structure.
In this project we take a look at Poisson equation for electrostatic potential generated by a localized charge distribution. The mathematical details are discussed in Part 1. After we formulated the problem we consider some general and most useful methods we use to solve it in the Part 2. We are going to use linear algebra methods for the problem. A short description of the Gaussian elimination and LU decomposition will be given there, as well as some discussion of the pros and cons for each of them.
In Part 3. the results are presented. We took the most important data and present it with critical discussion of algorithms. The last part of the report is Conclusion where we sum up all results and come up with the possible topics for the further studies in this topic.

2 Problem formulation

The problem we deal with is a second order differential equation (DE) with inhomogeneous term. Generally it can be written in a following form:

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

where y is unknown function of x, $f(x)$ is inhomogeneous term and k^2 is a real function.
Our DE is a classical equation of electrostatic potential, which is generated by known charge distribution.

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

where Φ is electrostatic potential and $\rho(\mathbf{r})$ is a charge distribution.
This is equation for three dimensions. We simplify this to one dimension equation in r assuming the functions for electrostatic potential and charge being spherically symmetric. After some simple variable substitution we get a general form for one-dimensional Poisson equation. For our case we use a Dirichlet boundary conditions for this equation. It looks as follows:

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

Here we introduce a discretization of the derivative. In order to do it we discretize a domain and define grid points for the x. We have $x \rightarrow x_i$ which corresponds to discretized value of a function in this point $u(x_i) \to v_i$. For simplicity we have the same distance between all grid point and this is our step length h. Each grid poin can now be calculated as $x_i = ih$. The values h can be calculated as $h = 1/(n+1)$. And the boundary condition is now defined as $v_0 = v_{n+1} = 0$ where n is a number of steps. The discretized Poisson equation then will be

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

f_i is function value at grid point i - $f(x_i)$.
It's easy to show that our discretized equation can be written as a set of linear algebra equations (SLAE):

$$\begin{cases} 0 - v_1 - v_2 = f_1 h^2 \\ -v_1 + 2v_2 - v_3 = f_1 h^2 \\ -v_2 + 2v_3 - v_4 = f_1 h^2 \\ \dots \\ -v_{n-1} + 2v_n - 0 = f_1 h^2 \end{cases}, \tag{2}$$

here we use $v_0 = v_{n+1} = 0$ conditions and multiply the right hand side of the equation with an h^2. To simplify this we now introduce a new function $\tilde{b}_i = h^2 f_i$. This system can be now written in a matrix form as:

$$\mathbf{A}\mathbf{v} = \tilde{\mathbf{b}}, \tag{3}$$

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

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{pmatrix}, \tag{4}$$

In our case we have a tridiagonal matrix with the same elements along the main diagonal (as well as to other diagonals). However we will look at more general case, the tridiagonal matrix with different elements along diagonals.

$$\mathbf{A} = \begin{pmatrix} b_1 & c_1 & 0 & \dots & \dots & \dots \\ a_1 & b_2 & c_2 & \dots & \dots & \dots \\ & a_2 & b_3 & c_3 & \dots & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ & & & a_{n-2} & b_{n-1} & c_{n-1} \\ & & & & a_{n-1} & b_n \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ \dots \\ \dots \\ v_n \end{pmatrix} = \begin{pmatrix} \tilde{b}_1 \\ \tilde{b}_2 \\ \dots \\ \dots \\ \dots \\ \tilde{b}_n \end{pmatrix}. \tag{5}$$

This kind of SLAE can be solved using different linear algebra methods. In the next section we will discuss some of them.

3 Theory and methods

3.1 Gaussian elimination (GE)

When we have a SLAE formulated as in (3), we would like to change the matrix A so it become all zeros below the diagonal. The most straightforward way to do it is Gauss elimination.
The algorithm for the Gaussian elimination can be described as to main operations the forward substitution and the backward substitution. In the first part, the forward substitution, we would transform the original matrix A to some other matrix, with zeros below the main diagonal. This can be done using some operations with matrix rows. We are using the first row to get rid of the elements below the diagonal in the first column. This operation however will change the other matrix elements. After this we are using the second row to eliminate all the elements below the diagonal in the second column and so on. Gaussian elimination require n^3 FLOPS. In our case we are lucky to have tridiagonal matrix, so the only elements to be changed are the elements along the diagonal. So in our algorithm we can treat A as a three vectors (or arrays), not as a square matrix $n \times n$. This will make our life much easier and the algorithm will be faster. The new matrix will look as follows

$$\mathbf{A} = \begin{pmatrix} b_1 & c_1 & 0 & \dots & \dots & \dots \\ 0 & {b_2}' = b_2 - c_1 \frac{a_1}{b_1} & c_2 & \dots & \dots & \dots \\ & 0 & {b_3}' = b_3 - c_2 \frac{a_2}{b'_2} & c_3 & \dots & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ & & & 0 & \dots & c_{n-1} \\ & & & & 0 & {b_n}' = b_n - c_{n-1} \frac{a_{n-1}}{b'_{n-1}} \end{pmatrix}, \tag{6}$$

After this we also need to change the right hand side part of the SLAE, the vector $\tilde{b}$. It will now be written as follows

$$\begin{pmatrix} \tilde{b}_1 \\ \tilde{b}_2{}' = \tilde{b}_2 - \tilde{b}_1 \frac{a_1}{b_1} \\ \dots \\ \dots \\ \dots \\ \tilde{b}_n{}' = \tilde{b}_n - \tilde{b}_{n-1} \frac{a_{n-1}}{b'_{n-1}} \end{pmatrix}. \tag{7}$$

What we have done so far is so called forward substitution. The GE consist of two big parts: the forward substitution (FS) and the backward substitution(BS). After we done the FS we should perform the BS to find the unknown vector v form (3).

We start with finding v_n and go up to v_1

$$\begin{cases} v_n = \frac{\tilde{b}'_n}{b'_n} \\ v_{n-1} = \frac{\tilde{b}'_{n-1} - c_{n-1} v_n}{b'_{n-1}} \\ \dots \\ \dots \\ v_1 = \frac{\tilde{b}'_1 - c_1 v_2}{b'_1} \end{cases}, \tag{8}$$

This is brute force algorithm for GE in case of tridiagonal matrix (sometimes called Thomas algorithm). This algorithm require $9n - 8$ floating points operations (FLOPS). In the program we will, however test two algorithms. The second one is adjusted to our case where we have same elements along the diagonals, so we can just pre-calculate some coefficients. We call this implementation the optimized GE and it is easy to show that such optimization reduces the number of FLOPS to $7n - 6$. In the results section we will go back to it and compare the time needed to implement the algorithms in a adjusted and in a brute force way. Code section with implementation of brute force GE algorithm can be found at `https://github.com/andrei-fys/fys4150_project1/blob/master/gauss.cpp#L64:L77`. Optimized algorithm code can is accessible via following link: `https://github.com/andrei-fys/fys4150_project1/blob/master/gauss.cpp#L115:L128`.

3.2 LU decomposition

This is another way to solve (3) using the linear algebra methods. In this case we will look for a way to write a matrix A in a following form

$$A = LU \tag{9}$$

here L is lower triangular matrix and U is upper triangular matrix. It's important that L to have only 1 on the main diagonal. Easy to show that $\det A$ in this case is equal to $\det U$. In order to implement this method we are going to use one of the libraries in C++. We will not derive the algorithm here. For the detailed mathematical description please refer to [1]. The detailed programming implementation is discussed in [2]. The most important thing here is that we will need the whole matrix A, not the three arrays that represent the diagonals as we can do it in the GE. The number of FLOPS for such algorithm is proportional to n^3.

Table 1: CPU time for Gaussian elimination(both brute force and optimized) and LU decomposition for tridiagonal matrices of N-dimensionality. Calculations were performed on Intel architecture CPU i7-3615QM.

N	Gaussian el., s	Gaussian el., optimized, s	LU decomposition, s
10	3×10^{-6}	1×10^{-6}	1.9×10^{-5}
10^2	5×10^{-6}	5×10^{-6}	2.7×10^{-3}
10^3	5.2×10^{-5}	4.7×10^{-5}	1.8
10^4	3.2×10^{-4}	2.5×10^{-4}	2.6×10^3
10^5	2.8×10^{-3}	2.7×10^{-3}	-

4 Results and discussion

In this part we present results obtained by Gaussian elimination and LU decomposition numerical methods. We compare efficiency of these two methods and make rough estimation for the convergence rate. Table 1 represents time measurement results for three algorithms: the brute force GE for tridiagonal matrices, the optimized GE for tridiagonal matrices and LU decomposition. As we expect the slowest algorithm is LU decomposition while both GE algorithms give slightly different result of computation time. One can see that CPU time correlates with number of floating point operations discussed in section 2. Both GE algorithms have the same factor of ten, that corresponds to what we have predicted from the theoretical approach. Number of floating point operations for these two methods differ just a bit. On the other hand it is difficult to analyze LU decomposition algorithm CPU time and CPU time for GE as LU in our particular case has much more RAM read-write operations which add extra-time. We used three vectors in our GE algorithm realization while for LU we initialized whole matrix of size $n \times n$. It leads to significant RAM requirements for big matrices regarding LU decomposition. We were unable to use LU decomposition for matrix of size $10^5 \times 10^5$ on our computer with just 8 GB of RAM while it could be possible with 80 GB of RAM. However a strong advantage of LU decomposition method is that right-hand sides are not needed to be known in advance as in GE method. In our particular case LU decomposition is rather overkill and not optimal usage of computer resources nevertheless it can be used for a smaller values of N to compare results of our GE algorithm as both methods have same numerical precision.
On figure 1 numerical(V_i) and exact(U_i) solutions plotted for 100 grid points. One can see computed values fits quite good exact solution. Relative error is no larger than 0.82% for 100 grid points. Relative error decreases with slope 2.2 with step size decrease. Deceasing trend continues till 10^5 grid points as it shown on figure 2. With a number of grid points larger then 10^5 one can see domination of round-off error becomes stronger.

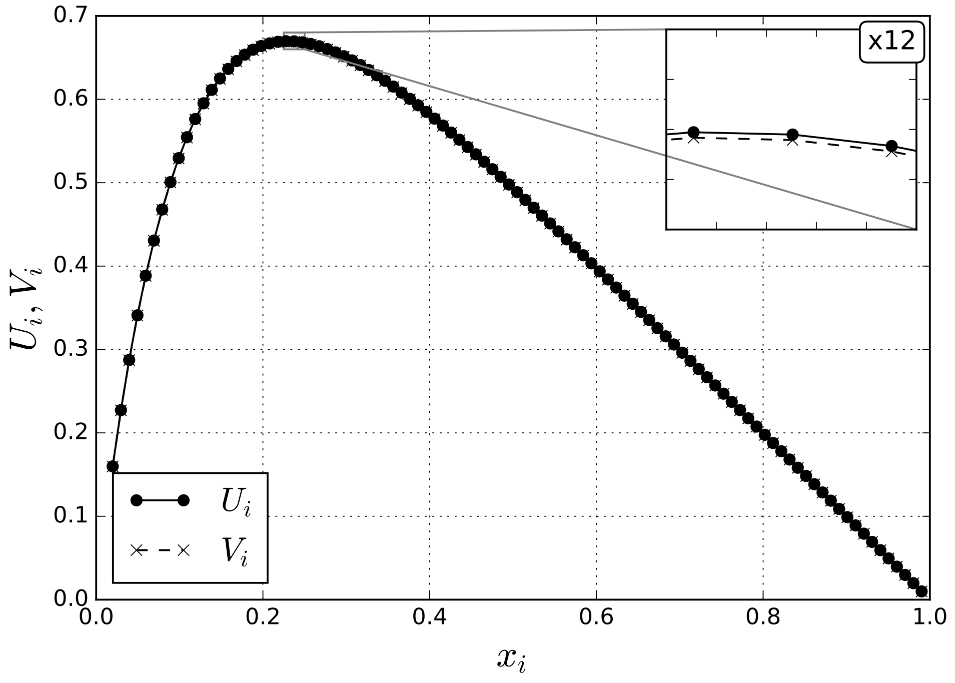

Figure 1: Numerical(V_i) and exact(U_i) solutions plotted for 100 grid points. The numerical solution was obtained by Gauss elimination algorithm. To show the difference between numerical and exact solution at given grid size we represent magnified(12 times) part of a graph.

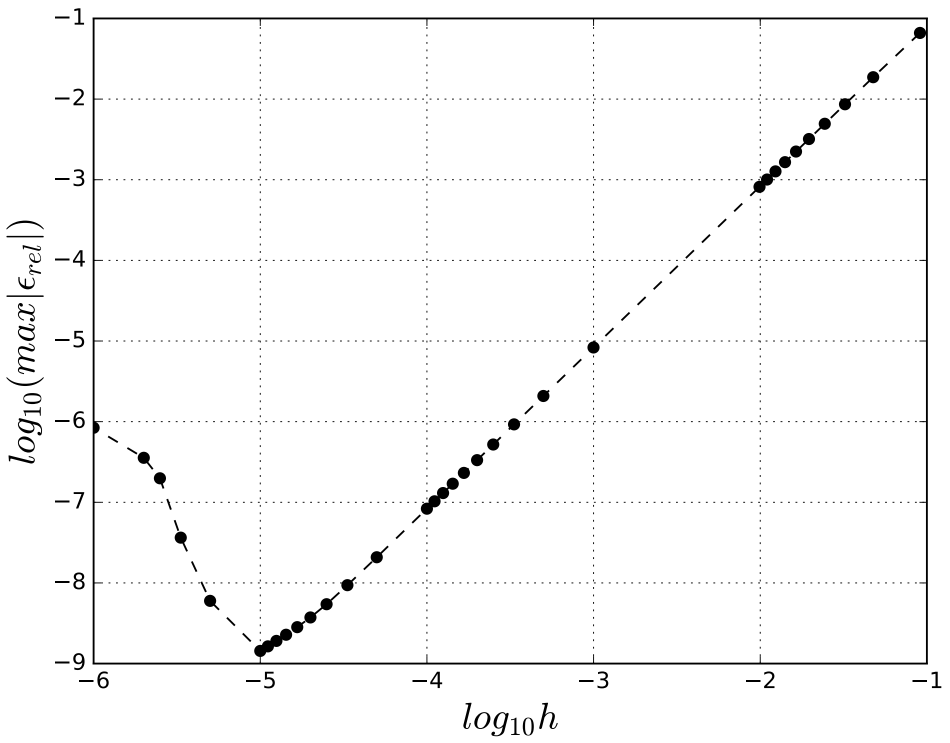

Figure 2: Dependency between maximum relative error and domain discretization step size on logarithmic scale. Roughly estimation of convergence rate is 2.2. Round-off errors dominate for the small step size (less then 10^{-5}).

5 Conclusion and further research

In this project we apply the linear algebra methods for the SLAE obtained after discretization of Poisson equation. It turns out that both Gaussian Elimination and LU decomposition gives us the same numerical precision. However, the LU decomposition takes much more CPU time. It was shown that CPU time correlates with the number of FLOPS for the method.
However, even LU turns out to be not the best choice for our problem, it's still quite a good method. The problem we solved was quite simple, for more complicated tasks, for example, finding the inverse of the matrix LU should be a much better tool to be used.

References

[1] Morten Hjorth-Jensen. *Computational Physics* . Lecture Notes Fall 2015, August 2015. W. Press, B. Flannery, S. Teukolsky, W. Vetterling, Numerical Recipes in C++, The art of scientific Computing (Cambridge University Press, 1999)

[2] W. Press, B. Flannery, S. Teukolsky, W. Vetterling *Numerical Recipes in C++, The art of scientific Computing.* Cambridge University Press, 1999.