FYS3150 - Project 1

Johannes K. Kjernlie

September 2016

Abstract

This project will consider the Poisson equation in one dimension. Various numerical methods for solving the discretized differential equation will be considered and their numerical efficiency will be compared. Furthermore, the approximate solutions will be compared to the exact solution. The results and source codes of the project can be found on GitHub: `https://github.com/Kjernlie/FYS3150/tree/master/clean/project1`

1 Introduction

This project will focus on techniques for solving a tridiagonal system of linear equations. The differential equation in question is

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

which is known as the general one-dimensional Poisson equation and has the following closed-form solution

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

The Poisson equation is a classical equation in electromagnetism, but the physics behind the equation will not be further discussed in this project. Three algorithms will be used to solve our equation, and the project will focus on the numerical performance of the algorithms.

The project report will be organized as follows. In the Sec. 2 a basic introduction to the three considered methods for solving our discretized differential equation will be given. Sec. 3 will consist of an example of an benchmark of our algortihm, and the implemention of the LU decomposition algorithm. Furthermore, in Sec. 4 we compare the computed results from our numerical methods, and compare them to the closed form solution. Additionally, we examine the relative error that occurs for different step sizes to find the step size yielding the best results. The author's conclusions and perspectives will be discussed in Sec. 5.

2 Method

2.1 Formulating the Problem

The discrete approximation of u is defined as v_i with grid points $x_i = ih$ in the interval from $x_0 = 0$ to $x_{N+1} = 1$. Furthermore, the step length is defined as $h = 1/(N+1)$.

The second derivative of u is discretized with

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

where $f_i = f(x_i)$ and $v_0 = v_{N+1} = 0$. This is a system of N linear equations, and can be written in matrix form

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

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

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

and $\tilde{b}_i = h^2 f_i$.

$\mathbf{A}$ can be rewritten in terms of one-dimensional vectors a, b, c of length $1 : N$. Our linear equation then reads

$$\mathbf{A} = \begin{pmatrix} b_1 & c_1 & 0 & \ldots & \ldots & \ldots \\ a_1 & b_2 & c_2 & \ldots & \ldots & \ldots \\ & a_2 & b_3 & c_3 & \ldots & \ldots \\ & \ldots & \ldots & \ldots & \ldots & \ldots \\ & & & a_{N-2} & b_{N-1} & c_{N-1} \\ & & & & a_{N-1} & b_N \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \ldots \\ \ldots \\ \ldots \\ v_N \end{pmatrix} = \begin{pmatrix} \tilde{b}_1 \\ \tilde{b}_2 \\ \ldots \\ \ldots \\ \ldots \\ \tilde{b}_N \end{pmatrix}. \tag{6}$$

Note that the endpoints are not included since the boundary conditions used results in a fixed value for v_i.

2.2 Gaussian Elimination

Gaussian elimination is the basic method used for solving systems of linear equations on the form $\mathbf{A}\mathbf{x} = \mathbf{y}$. It can be used for any system of linear equations, not just tridiagonal matrices, assuming a non-singular matrix A and that the matrix elements along the diagonal satisfy $a_{ii} \neq 0$ [1].

In Gaussian elimination we use the first equation to eliminate the first unknown x_1 from the remaining $N-1$ equations. Further, we use new the second

equation to eliminate the second unknown x_2 from the remaining $N-2$ equations. After $N-1$ eliminations like that, we find a so-called upper triangular set of equations. This step is called the forward substitution and the general expression for the new coefficients after each elimination is

$$a_{jk}^{(m+1)} = a_{jk}^{(m)} - \frac{a_{jm}^{(m)} a_{mk}^{(m)}}{a_{mm}^{(m)}} \qquad j,k \ = m+1,\ldots,N, \tag{7}$$

where $m = 1,\ldots,N-1$ and the new coefficients on the right hand side is given by

$$y_j^{(m+1)} = y_j^{(m)} - \frac{a_{jm}^{(m)} y_m^{(m)}}{a_{mm}^{(m)}} \qquad j \ = m+1,\ldots,N. \tag{8}$$

Note that the coefficients of the first equation is unchanged. We can find the solution $\mathbf{y}$ through backward substitution

$$x_j = \frac{1}{a_{mm}}\left(y_j - \sum_{k=j+1}^{N} a_{jk}x_k\right) \qquad j \ = \ N-1, N-2,\ldots,1. \tag{9}$$

Note that $x_N = \frac{y_N}{a_{NN}}$ [1].

2.3 Tridiagonal Systems of Linear Equations

For a tridiagonal system of linear equations, like the one in (2.1), we can use a rather simple solution algorithm. We are going to use a forward substitution with a loop over the elements i which updates the diagonal elements b_i given by the new diagonals $\hat{b}_i$

$$\hat{b}_i = b_i - \frac{a_i c_{i-1}}{\hat{b}_{i-1}}, \tag{10}$$

and the new right hand side $\hat{f}_i$ is given by

$$\hat{f}_i = \tilde{b}_i - \frac{a_i \hat{f}_{i-1}}{\hat{b}_{i-1}}. \tag{11}$$

Note that $\hat{b}_1 = b_1$ and $\hat{f}_1 = \tilde{b}_1$ always. The final solution v_i is found by backward substitution

$$v_{i-1} = \frac{\hat{f}_{i-1} - c_{i-1}v_i}{\hat{b}_{i-1}}, \tag{12}$$

and $v_N = \hat{f}_N/\hat{b}_N$, for the last point, $i = N$. Due to this algorithms simplicity the number of necessary floating point operations is proportional to $\mathcal{O}(N)$. The eact number can be found by seeing that the forward and backward sweeps requires $6(N-1)$ and $3(N-1)+1$ floating point operations, respectively, giving $9(N-1)+1$ for the whole system. For Gaussian elimination a number in the order of $2N^3/3 + \mathcal{O}(N^2)$ floating point operations is required. Hence, Gaussian elimination is inferior to the method explained in this section for a tridiagonal system of equations [1].

2.4 Specialized Case

For the specific matrix we are going to use in this project we can actually simplify the solution algorithm by precalculating the updated matrix elements $\hat{b}_i$. We will then get

$$\hat{b}_i = 2 - \frac{1}{\hat{b}_{i-1}} = \frac{i+1}{i}, \tag{13}$$

and the new right hand side $\hat{f}_i$ is given by

$$\hat{f}_i = \tilde{b}_i + \frac{(i-1)\hat{f}_{i-1}}{i}. \tag{14}$$

For our specialized case we have that $\hat{b}_1 = 2$ and $\hat{f}_1 = \tilde{b}_1$. The backward substitution to find the final solution is then given by

$$v_{i-1} = \frac{i-1}{i}(\hat{f}_{i-1} + v_i), \tag{15}$$

where $v_N = \hat{f}_N/\hat{b}_N$.

The forward and backward sweeps requires $2(N-1)$ and $2(N-1)+1$ floating point operations, respectively. This gives an altogether number of $4(N-1)+1$.

2.5 LU decomposition

A frequently used approach for solving linear system is to use LU decomposition. LU decomposition is a form of Gaussian elimination and involves factoring a matrix, $\mathbf{A}$, as the product of a lower triangular matrix, $\mathbf{L}$, and an upper triangular matrix, $\mathbf{U}$ [1,3].

To find $\mathbf{L}$ and $\mathbf{U}$ we calculate the elements, l_{ij} and u_{ij}, respectively, columnwise and start with the first column.

For each column, j:

1. Compute the first element in $\mathbf{U}$, u_{1j} by

$$u_{1j} = a_{1j}. \tag{16}$$

2. Calculate all elements $u_{ij},\ i = 2, \ldots, j-1$

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}. \tag{17}$$

3. Calculate the diagonal element u_{jj}

$$u_{jj} = a_{jj} - \sum_{k=1}^{j-1} l_{jk}u_{kj}. \tag{18}$$

4. Calculate the elements $l_{ij},\ i > j$

$$l_{ij} = \frac{1}{u_{jj}}\left(a_{jj} - \sum_{k=1}^{i-1} l_{ik}u_{kj}\right). \tag{19}$$

Note that the diagonal elements of $\mathbf{L}$ is 1 in this algorithm. Also, keep in mind that a value of u_{jj} close to zero can be dangerous as seen in Eq. 4 [1].

The procedure of solving a system of linear equations with LU decomposition is rather straightforward. You start with a system of linear equations written in matrix form

$$\mathbf{Ax} = \mathbf{b}, \tag{20}$$

where $\mathbf{A}$ and $\mathbf{b}$ are know. With LU decomposition we can write

$$\mathbf{LUx} = \mathbf{Pb}, \tag{21}$$

where P is the permutation matrix. The system can then be calculated in two steps

1. Calculate $\mathbf{y}$ of
$$\mathbf{Ly} = \mathbf{Pb} \tag{22}$$
2. Calculate the final solution $\mathbf{y}$ from
$$\mathbf{Ux} = \mathbf{y} \tag{23}$$

LU decomposition is a smart choice to implement for applications where the right hand side vector $\mathbf{b}$ keeps changing. This is due to the fact that in LU decomposition the factorization phase is decoupled with the solving phase. In the factorization phase we only need $\mathbf{A}$, while in the solving phase we use both the factored form of $\mathbf{A}$ and $\mathbf{b}$. Thus, once we have the factored form of $\mathbf{A}$ we can solve the system for different $\mathbf{b}$s at relatively moderate computational cost. The number of floating point operations needed for the factorization of $\mathbf{A}$ into $\mathbf{LU}$ is $\mathcal{O}(N^3)$, but the number of floating point operations required for the solving part is only in the order of $\mathcal{O}(N^2)$. So for a number of r right hand side vectors the computational cost of the problem scales as $\mathcal{O}(N^3 + rN^2)$, while it scales with $\mathcal{O}(rN^3)$ for Gaussian elimination, since each Gauss elimination independently requires a number of floating point operations proportional with $\mathcal{O}(N^3)$ [4].

2.6 Relative Error

The relative error is used to find the discrepancy between an exact value and some approximation of it. The relative error is the absolute error divided by

the magnitude of the exact value, where the absolute error is the magnitude of the difference between the exact and approximate value [1]. For a logarithmic scale the relative error is given by

$$\epsilon_i = log_{10}\left(\left|\frac{v_i - u_i}{u_i}\right|\right) \qquad \text{for } i = 1, \ldots, N. \tag{24}$$

3 Implementation

The full code and benchmarks can be found at GitHub: `https://github.com/Kjernlie/FYS3150/tree/master/clean/project1`.

3.1 Example Benchmark

Here, a small benchmark is included in the report. The results from the tridiagonal solver (Sec. 2.3) is compared to the results obtained from the LU decomposition (Sec. 2.5). We expect the result to be equal because of the numerical precision of our computations.

```
...
check = []
for i in range(len(x1)):
    check.append(v1[i]-v12[i])
print check

# Results
# [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
...
```

Here **v1** is the result from the tridiagonal solver, while **v12** is the result from the LU decomposition. The vector **check** consist of only zeros, which we expected.

3.2 LU Decomposition

For the implementation of the LU decompostion (Sec. 2.5), the C++ linear algebra library *Armadillo* [2] is used. With Armadillo, the LU decomposition can be implemented as follows.

```
using namespace arma;
...
lu(L,U,P,A);
    mat F = P * f;

    vec y = solve(L,F);
    vec z = solve(U,y);
...
```

Here **L** and **U** is the lower and upper triangular matrices of **A**, respectively. **P** is the permutation matrix, and **f** is our right hand side vector.

4 Results

In Fig. 1 the exact solution is plotted against approximate solutions computed from the tridiagonal matrix solver (Sec. 2.3) and LU decomposition (Sec. 2.5). The figure shows that the numerical methods yields a very good approximation for the higher mesh sizes, N. For $N = 10$ both methods underestimates the exact solution, but for $N = 100$ and $N = 1000$ the approximation is visually inseparably from the exact solution.

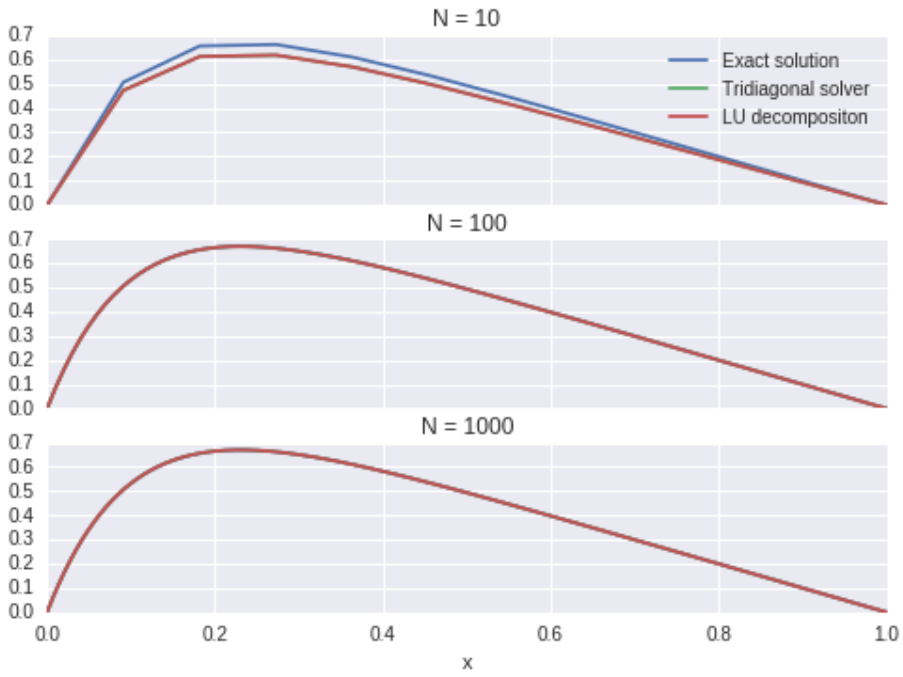

Figure 1: The exact solution of the Poisson equation is plotted against the approximate solution for three different number of steps, $N = 10$ (top), $N = 100$ (middle) and $N = 1000$ (bottom). The approximate solution is solved using forward and backward substitution of a tridiagonal system of linear equation, like explained in Sec. 2.3.

Table 1 the CPU time from the methods explained in Sec. 2.3, 2.4 and 2.5 is presented. We can see that the specialized case from Sec. 2.4 is the fastest algorithm, while the LU decomposition form Sec. 2.5 is the slowest. For a large grid sizes, LU decomposition becomes unusable. This is due to the high number of floating point operations required to do LU decomposition. Thus, for large mesh sizes it will use all of the computer's RAM memory and stop working.

In Table 2 we can see the log_{10} to the relative errors from different mesh sizes. The relative error is presented in a logarithmic scale, and is computed

N	Standard (s)	Special (s)	LU (s)
10^1	0.0000846	0.0000007	0.0001342
10^2	0.000116	0.0000296	0.0044
10^3	0.0002572	0.0002056	2.385
10^4	0.002022	0.001723	N/A
10^5	0.01837	0.01792	N/A
10^6	0.165	0.162	N/A

Table 1: Used CPU time for six different mesh sizes. Three solution procedures are used, the standard tridiagonal matrix solver from Sec. 2.3, specialized case for this projects matrix from Sec. 2.4 and LU decomposion from Sec. 2.5. The presented times are averages of five runs. Took a mean after 5 runs.

with the general tridiagonal solver presented in Sec. 2.3. The values show that the error decreases for a larger mesh sizes, until it reaches $N = 10^7$ where it starts to increase again. This is due to round-off errors and loss of precision which occurs when the computer does arithmetic operations with very small numbers. In Fig. 2 the log_{10} to the relative error is plotted against the mesh size N [1].

N	ϵ
10^1	-1.179698
10^2	-3.088037
10^3	-5.080052
10^4	-7.079270
10^5	-9.080322
10^6	-10.163609
10^7	-9.0899770

Table 2: Log_{10} to the maximum relative error for seven different mesh sizes. The relative error is computed from the formula in Sec. 2.6

5 Discussion

In this project we have implemented three different numerical methods for solving the Poisson equation in one-dimension, a solver for tridiagonal systems of linear equations (Sec. 2.3), a specialized solver for the specific set of linear equations used in this project (Sec. 2.4) and LU decomposition (Sec. 2.5). The results show that each method yield satisfying result, although at different computational cost.

We have determined that the number of floating point operations is very important when determining the computational efficiency of numerical methods.

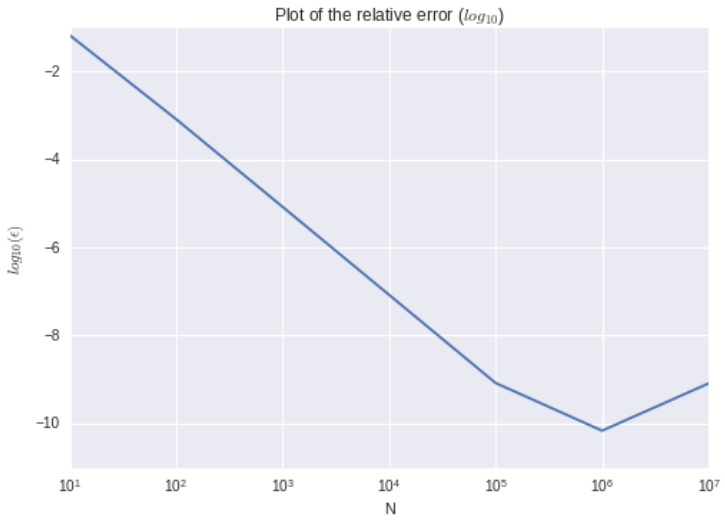

Figure 2: Plot of the relative error (log_{10}) against seven different mesh sizes. The relative error is computed using the formula in Sec. 2.6

Our specialized algorithm for our specific matrix is clearly the superior method of the three we have implemented. It requires the lowest number of floating point operations, and it also has the lowest CPU time. The LU decomposition performs the worst. It has the highest required number of floating point operations and the longest CPU time. This is expected since it is a general algorithm. Our tridiagal matrix solver is superior to the LU decompositon but it is slightly less computationally efficient than the specialized algorithm.

In Sec. 4 I mentioned that LU decomposition doesn't work for large mesh sizes, because it takes up too much memory. The tridiagonal matrix solver and our specialized solver work for these mesh sizes because it doesn't have to store each element of our matrix, they only store the diagonal vectors. The specialized method also takes into account that our diagonal vectors are constant. Hence, making them more memory preserving than the general LU decomposition.

5.1 Numerical errors

Gaussian elimination will yield the correct answer for a non-singular matrix **A** and exact arithmetics, but since computer arithmetics is not completely exact we will always get truncations and possible loss of precision, which is worth keeping in mind [1]. The same is true for our more specialized methods.

From Fig. 2 and Table 2 we also saw that the relative error will eventually begin to increase for increasing mesh sizes. This is because of the loss of precision due to round-off errors that occurs when the computer does arithmetic operations for very small numbers.

5.2 Future Work

This is a field where a lot of research previously has been done, so the creation of new superior numerical methods is, in the author's mind, not a realistic goal for future work. Still, relevant further work aimed for learning could be to do a literature search of the research field, and implement and test better methods if existing.

References

[1] Morten Hjorth-Jensen. Computational physics. *Lecture notes*, 2015.

[2] Conrad Sanderson. Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments. 2010.

[3] Alan M Turing. Rounding-off errors in matrix processes. *The Quarterly Journal of Mechanics and Applied Mathematics*, 1(1):287–308, 1948.

[4] user17762. Necessity/advantage of lu decomposition over gaussian elimination. Mathematics Stack Exchange. URL:http://math.stackexchange.com/q/266360 (version: 2012-12-28).