FYS3150 Project 1

Solving the one-dimensional Poisson equation with Dirichlet boundary conditions using the Thomas algorithm and the LU decomposition.

Mira Mors and Elias Tidemand Ruud

Computational Physics I FYS3150/FYS4150

Department of Physics
University of Oslo
Norway
September 2020

Abstract

In this report, the Thomas algorithm and the LU decomposition are used to approximate the one-dimensional Poisson equation for $x \in (0,1)$. Both algorithms have high accuracy as the number of integration steps, n, increases (single precision accurate). For $n > 10^5$, the deviation increases again due to limitations in machine precision. Moreover, the numerical implementation of the LU decomposition is notably slower and is limited to $n < 10^5$. On the other hand, a specialized Thomas algorithm can shorten the CPU time.

Introduction

The aim of this project is to numerically solve the one-dimensional Poisson equation with Dirichlet boundary conditions. Efficiency and speed are key factors to successfully automatize complex mathematical problems. Therefore, two algorithms (Thomas algorithm and LU decomposition) are tested for their accuracy and effectiveness. Furthermore, the general Thomas algorithm is customized to the precise task in order improve efficiency.
The project begins with presenting the theoretical models for the Thomas algorithm and LU decomposition. Next, a general algorithm for the numerically implementation is given. Finally, the results are analyzed and discussed.
The code for the numerical computations is found on Github repository.

Method

Approximation of the second derivative Given the one-dimensional Poisson equation with Dirichlet boundary conditions

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

we approximate the second derivative of u with

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

where $f_i = f(x_i)$ and the discretized approximation to u is v_i with grid points $x_i = ih$ in the interval from x_0 to $x_{n+1} = 1$. Further, the step length, h, is defined as $h = 1/n$ and the boundary conditions set $v_0 = v_n = 0$.
For numerical computation, (1) must be written as a linear set of equations of the form

$$\mathbf{Av} = \tilde{\mathbf{g}},$$

where $\mathbf{A}$ is an $n \times n$ tridiagonal matrix.
The set of equation for $h^2 f_i$ is given by

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

Rearranging the equation gives

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

Determining the equations for the boundary condition, $i = 1$ holds

$$\begin{aligned} -v_0 + 2v_1 - v_2 &= h^2 f_1 \\ 0 + 2v_1 - v_2 &= h^2 f_1 \end{aligned} \tag{3}$$

since $v_0 = 0$. Given $v_{n+1} = 0$, similarly for $i = n$ applies

$$\begin{aligned} -v_{n-1} + 2vn - v_{n+1} &= h^2 f_n \\ -v_{n-1} + 2v_n - 0 &= h^2 f_n \end{aligned} \tag{4}$$

A combination of the above equations results in

$$\begin{aligned} 2v_1 - v_2 = h^2 f_1 \quad &= \tilde{g_1} \\ &... \\ -v_{i-1} + 2v_i - v_{i+1} = h^2 f_i \quad &= \tilde{g_i} \\ &... \\ -v_{n-1} + 2v_n = h^2 f_n \quad &= \tilde{g_n} \end{aligned} \tag{5}$$

Thus, there are $n-1$ equations that must be computed ($i \in [1, n-1]$). The set of equations can be rewritten in matrix-form, there $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{v}, \mathbf{g} \in \mathbb{R}^n$

$$\begin{bmatrix} 2 & -1 & 0 & \dots & \dots & \dots \\ -1 & 2 & -1 & \dots & \dots & \dots \\ & -1 & 2 & -1 & \dots & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ \dots \\ \dots \\ v_n \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ \dots \\ \dots \\ \dots \\ g_n \end{bmatrix}.$$

For simplicity, v_i is from now on written as u_i. Furhter, the vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are defined as the matrix-elements, there $\mathbf{b}$ is placed along the diagonal, $\mathbf{a}$ is the lower diagonal and $\mathbf{c}$ is the upper diagonal. g_i stands for $h^2 f_i$, the solution for each equation (called source term)

$$\mathbf{A} = \begin{bmatrix} b_1 & c_1 & 0 & \dots & \dots & \dots \\ a_1 & b_2 & c_2 & \dots & \dots & \dots \\ & a_2 & b_3 & c_3 & \dots & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ & & & a_{n-2} & b_{n-1} & c_{n-1} \\ & & & & a_{n-1} & b_n \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \dots \\ \dots \\ \dots \\ u_n \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ \dots \\ \dots \\ \dots \\ g_n \end{bmatrix}.$$

In our project, the source term is predefined to be $f(x) = 100e^{-10x}$. Given the above defined boundary conditions, this differential equation has a closed-form solution given by $u(x) = 1 - (1 - e^{-10})x - e^{-10x}$. Hereafter, the function values for $u(x)$ are called **exact**.

This linear set of equations can be solved by for example using the Thomas algorithm or the LU decomposition.

Thomas Algorithm The Thomas algorithm can be divided into two steps: a forward substitution and a backward substitution [1].
In the forward substitution, the lower-diagonal elements are eliminated. Starting from the top, this is done by summing two successive appropriately scaled rows in the matrix. The mathematical expression for the new matrix elements is given as

$$\begin{aligned} \tilde{b}_i &= b_i - a_{j-1} \cdot c_{i-1}/\tilde{b}_{i-1} \quad \text{for } i = 2, ..., n-1 \\ \tilde{g}_i &= g_i - a_{j-1} \cdot \tilde{g}_{i-1}/\tilde{b}_{i-1} \quad \text{for } i = 2, ..., n-1 \end{aligned} \tag{6}$$

The first row remains unchanged.
The backward substitution starts at the last row of the matrix. Here, the expression for u_{n-1} becomes immediately clear

$$\begin{aligned} \tilde{b}_{n-1} u_{n-1} &= \tilde{g}_{n-1} \\ u_{n-1} &= \tilde{g}_{n-1}/\tilde{b}_{n-1} \end{aligned} \tag{7}$$

Going up row for row, u_i can be expressed as

$$u_i = (\tilde{g}_i - c_i u_{i+1})/\tilde{b_i} \quad \text{for } i = n-2, ..., 1 \tag{8}$$

Special Case For our computation, we set identical matrix elements along the diagonal and identical (but different) values for the non-diagonal elements. More specific, we set $b_j = 2$ and $a_j = c_j = -1$. Thus the Thomas algorithm can be specialized to our specific case. It can be shown that the elements along the diagonal can be precalculated with

$$\tilde{b_i} = \frac{i+1}{i} \quad \text{for } i = 1, ..., n-1 \tag{9}$$

That means that $\tilde{\mathbf{b}}$ no longer needs to be computed in the forward loop.
The forward loop is therefore given by

$$\tilde{g}_i = g_i - a_{j-1} \cdot \tilde{g}_{i-1}/\tilde{b}_{pre,i-1} \quad \text{for } i = 2, ..., n-1 \tag{10}$$

there $\tilde{b}_{pre}$ stands for the precalculation. Similarly, the backward loop is now also computed with the precalculated $\tilde{\mathbf{b}}$

$$u_i = (\tilde{g}_i - c_i u_{i+1})/\tilde{b}_{pre,i} \quad \text{for } i = n-2, ..., 1 \tag{11}$$

The tables below (Table 1 and Table 2) show what effect this precalculation has on the number of FLOPs in the forward and backward substitution.

Table 1: FLOPS for general algorithm

FLOPs	Forward	Backward	Total
b	$3(n-2)$	-	-
g	$3(n-2)$	-	-
u	-	$3(n-2)$	-
Total	$6(n-2)$	$3(n-2)$	$9(n-2)$

Table 2: FLOPS for special algorithm

FLOPs	Forward	Backward	Total
g	$2(n-2)$	-	-
u	-	$2(n-2)$	-
Total	$2(n-2)$	$2(n-2)$	$4(n-2)$

For the special algorithm, the $2(n-1)$ FLOPs for the precalculated **b** must be taken into consideration. If n is a large number, the following approximation applies: $n-2 \approx n-1 \approx n$. Thus, comparing the two algorithms, the number of FLOPs are reduced from $9n$ (general) to $6n$ (special).

LU decomposition Another way to solve the set of linear equations is the LU decomposition [2]. This method factorizes the given square **A** matrix into two triangular matrices, one lower triangular matrix, **L**, and one upper triangular matrix, **U**. The original matrix **A** is then given by their matrix-product, i.e. $\mathbf{A} = \mathbf{LU}$. The Gauss Elimination Method is used on **A** to form **U**. The diagonal elements of **L** are set to one. For a 3×3 matrix **A**, the LU decomposition would look like

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

The LU decomposition allows to rewrite the equation $\mathbf{Ax} = \mathbf{g}$ into two systems

$$\begin{aligned} \mathbf{Ly} &= \mathbf{g} \\ \mathbf{Ux} &= \mathbf{y} \end{aligned} \tag{12}$$

Forward substitution can be used to solve the system $\mathbf{Ly} = \mathbf{g}$ and backward substitution can be used to solve $\mathbf{Ux} = \mathbf{y}$.
Looking at the number of FLOPS, the dominating term is the LU factorization [3]. Row reduction changes in total $(n-1) \cdot (n-1)$ elements. Each new element is calculated with 3 FLOPS, there 1 FLOPs can be precalculated. For a $\mathbf{A^{3\times 3}}$ matrix, the reduced second row would be given by subtracting the first row times an appropriate scalar

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \sim \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} - a_{21}/a_{11}a_{12} & a_{23} - a_{21}/a_{11}a_{13} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}.$$

The computation for the two equations below include 3 FLOPS each

$$a_{22} - a_{21}/a_{11}a_{12}$$

$$a_{23} - a_{21}/a_{11}a_{13}.$$

a_{21}/a_{11} can be precalculated, and therefore 2 FLOPS er needed to calculate the reduced second row. Similarly, the reduced third row is computed with 2

FLOPS, there a_{31}/a_{11} can be precalculated. In total, row reduction includes $2(n-1)(n-1)$ FLOPS. Now, the first column contains zeros below the first element. In order to create an upper diagonal matrix, this method has to be repeated $n-1$ times. The general mathematical expression for the number of FLOPS is

$$2\sum_{j=1}^{n} j^2 = \frac{2}{6}(2n^3 + 2n^2 + n) \sim \frac{2}{3}n^3 \sim O(2/3n^3). \tag{13}$$

In order words, if n is large, FLOPs are proportional to n^3.

General algorithm

Given below are pseudocodes for the Thomas algorithm and the LU decomposition [4].

Thomas Algorithm

1. Setting the input variables (x0, xn, a, b, c, n)
   x0 and xn confine the interval (here from 0 to 1).
   n stands for the number of integration points.
   In our case, the elements along the tridiagonal are constant, e.g. $a_i = a$, $b_i = b, c_i = c$ for all i.

2. Initialization: Defining the step size h.
   Allocation of memory for $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{g}, \mathbf{u}, \mathbf{exact}$
   $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{g}$ have length n, whereas $\mathbf{u}, \mathbf{exact}$ have length $n+1$
   for (i=1, n-1):
   general: set $a_i = a$, $b_i = b, c_i = c$
   special: set $a_i = a$, $b_i = (i+1)/i$, $c_i = c$
   compute **exact**

3. Thomas algorithm with forward substitution
   for (i=2, n-1):
   general case: update $\mathbf{b}, \mathbf{g}$
   special case: update $\mathbf{g}$

4. Thomas algorithm with backward substitution
   compute u_{n-1}
   while (i=n-2, 1):
   compute **u** (general / special case)

5. Testing analytic solution: compute relative error $\epsilon = \left| \frac{u_i - exact_i}{exact_i} \right|_{max}$

6. Measure average CPU time for the different algorithms

7. Write results to file and if applicable plot them

LU decomposition Using armadillo with its built-in functions

1. Initialize Setting the input variables (x0, xn, a, b, c, n)
   x0 and xn confine the interval (here from 0 to 1).
   n stands for the number of integration points.
   In our case, the elements along the tridiagonal are constant, e.g. $a_i = a$, $b_i = b, c_i = c$ for all i.

2. Setting up matrix $\mathbf{A}^{n-1 \times n-1}$ using built-in function (i.e. mat A = zeros<mat>(n-1,n-1))
   for(1, n-2):
   Fill up matrix
   First and last row must be set separately

3. Compute **L** and **U** (built-in function)

4. Compute **y** in $\mathbf{Ly} = \mathbf{g}$ using forward substitution (built-in function)

5. Compute **x** in $\mathbf{Ux} = \mathbf{y}$ using backward substitution (built-in function)
   x is the solution-vector **u**

6. Measure average CPU time

Results

To ensure that all 3 algorithms have been implemented correctly (found on Github repository), the computed solution for the general and special Thomas algorithm and LU method, for n=10, are given below (Table 3). As a benchmark, the correct solution is also shown.

Table 3: Computed values for all 3 algorithms (n=10)

x_val	General	Special	LU	exact
0.1	0.489914	0.489914	0.4899	0.532125
0.2	0.611948	0.611948	0.6119	0.664674
0.3	0.598646	0.598646	0.5986	0.650227
0.4	0.535558	0.535558	0.5356	0.581703
0.5	0.454154	0.454154	0.4542	0.493285
0.6	0.366012	0.366012	0.3660	0.397549
0.7	0.275392	0.275392	0.2754	0.299120
0.8	0.183859	0.183859	0.1839	0.199701
0.9	0.091991	0.091991	0.0920	0.099917

The values for the general, special algorithm and LU decomposition are the same (with LU being rounded off). In addition, these values correspond well with the exact solution. It can therefore be assumed that the implementation is

done correctly.

The following figures show the results for computing the general Thomas algorithm for the matrices of the size 10×10, 100×100, 1000×1000 (corresponding to $n = 10$, $n = 100$ and $n = 1000$), in the interval $x \in (0, 1)$. The computed solution is graphed together with the exact solution.

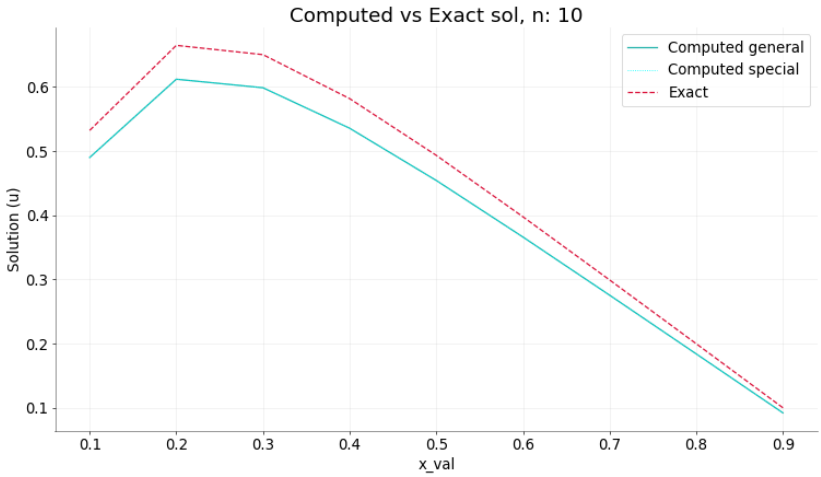

Figure 1: general Thomas algorithm graphed with exact solution (n=10)

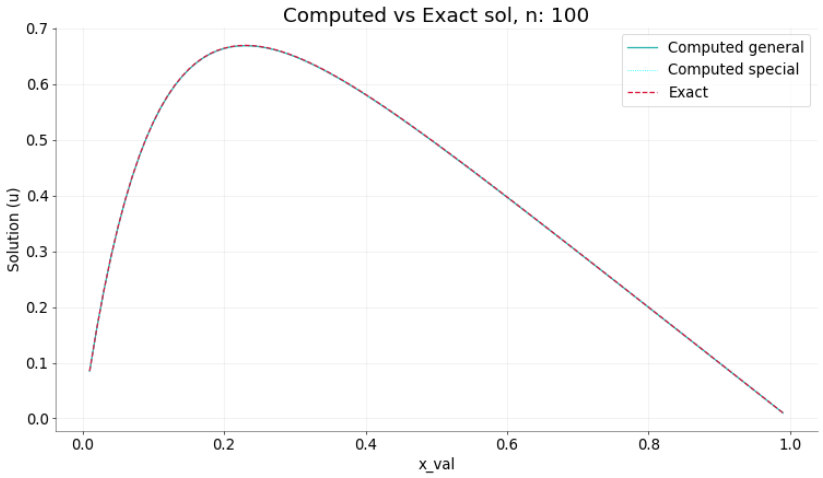

Figure 2: general Thomas algorithm graphed with exact solution (n=100)

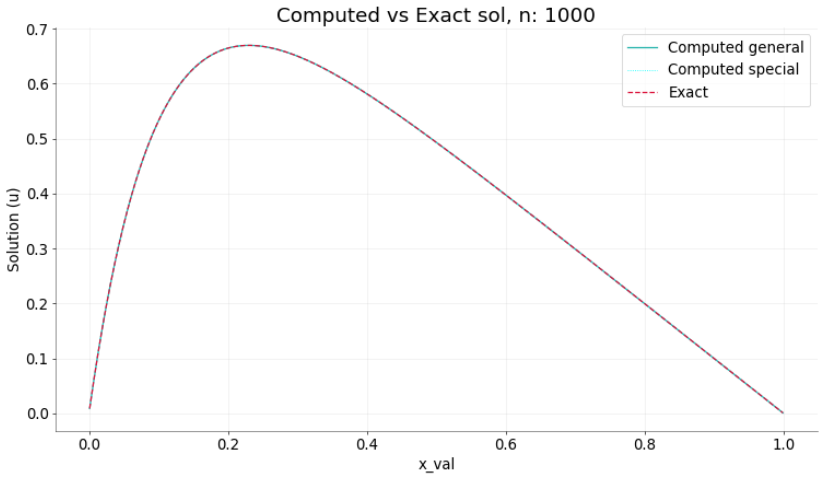

Figure 3: general Thomas algorithm graphed with exact solution (n=1000)

In general, the computed values match the exact graph better as n increases. Furthermore, the approximation worsens for a varying slope. This becomes

obvious when looking at Figure . Given a peak, the slope changes a lot, which is why the error is max there. As n increases, there is no visible difference between the two curves.

To examine the accuracy of the general Thomas algorithm, the max relative error, $\epsilon = \left|\frac{u_i - exact_i}{exact_i}\right|_{max}$, in the data set $i = 1, ...n-1$, is given in Table 4.

Table 4: logarithmic error for the general Thomas algorithm

n	Max LogErr
10.0	-1.100582
100.0	-3.079398
1000.0	-5.079183
10000.0	-7.079198
100000.0	-8.842964
1000000.0	-6.075508
10000000.0	-5.52523

The relative error decreases initially with increasing n as expected. Further, the error declines by a factor of 10^2 for each each step until $n = 10^4$. For the lower values of n, the mathematical error due to truncation dominates. It continues to decrease until $n = 10^5$ (not at the same pace), but then starts to increase. This is due to the machine rounding error exploding, which is proportional to $1/h^2$ [3]. In general, the computed solution is quite precise. In comparison, single machine precision is in order of 10^{-7}.

Next, the table below (Table 5) shows the CPU time for the general and special Thomas algorithm for matrices up to $n = 10^6$. As the CPU time measurement can be random, the average time of 40 repetitions is used.

Table 5: CPU time for all 3 algorithms

n	General time[s]	Special time[s]	LU time[s]
10	2.250E-06	2.425E-06	8.593E-05
100	8.250E-06	7.250E-06	1.251E-03
1000	7.240E-05	6.593E-05	1.270E-01
10000	9.013E-04	7.289E-04	8.675E+01
100000	8.585E-03	6.494E-03	NAN
1000000	8.745E-02	6.564E-02	NAN
10000000	9.784E-01	8.487E-01	NAN

As expected, the special Thomas algorithm is the most rapid and LU decomposition the slowest. Since the LU method makes use of a whole matrix (instead of only allocating the tridiagonal using arrays), the memory space is to small if $n > 10^4$. That is, we only get results for $n < 10^5$. The exceeding of memory space can be explained with the following calculations. Double precision floating

point numbers use 64 bits per number, which equals 8 bytes per number. For a $10^4 \times 10^4$ matrix $8 \cdot 10^8$ bytes are needed. This takes up almost 1 GB of memory space. For a $10^5 \times 10^5$ matrix, more than 70 GB are needed. A standard laptop has 8-16 GB RAM, the available memory is exceeded for larger n.
Further, for every step, the LU time increases notably. That is no surprise, since FLOPs are proportional to n^3.
Even though the FLOPS for the special algorithm are reduced from $9n$ (general algorithm) to $6n$, other factors such as fetching variables must be considered. Moreover, the measured time is affected by the PC's granularity, temperature of the processor, number of chores the CPU is processing and so forth. This might explain why the time difference between the general and special algorithm is less than expected.

Improvement of the code implementation

The code consists of mainly three files; algo.cpp, main.cpp and visualize.py. Even though the code runs well there is certainly improvements to be made. The initial structure was built mainly to solve for the Thomas algorithm. Which in turn, when adding the LU decomposition, made certain methods such as writetofile and print* not usable for the this method. The same method SolveLU also clashed with the way initialize was built, so having to have another three parameters for this method should be excessive. However, a point was made to not have an obscene amount of class variables to store, which could make it confusing to which parameter belongs to which method.One could also make main method more user friendly, perhaps by creating a new class just to create all the necessary results in a neat way and make it accessible. Again, we did not want to overcomplicate the structure by making another class.
The visualize file is separated in several parts. As this file ended up quite long towards the end, it could benefit to either split it up in several files or another class. All in all, there is room for improvement to make the code easier to read and more effective, although in simple cases where, we just solve the basic problems over a loop, it is not too complicated.

Conclusion

In this project ,we have shown that by taking on a second order differential equation, by discretizing and using numerical methods, its possible to reach a very high precision. The result shows that their accuracy is about single machine precision high. For each n-value the log of the relative error decreases with 2, until a threshold is reached, at about $n > 10^5$. This is due to machine precision which begins falsifying the results, especially as the steps size, h, becomes small. Moreover, if $n > 10^4$, the LU method fails due to exceeding memory capacity. In general, the LU decomposition is much slower than the other two algorithms. The CPU time for the general and special Thomas algorithm is the same magnitude, although the special method consistently beats its counterpart by a slight margin.

This roots in the different amount of computations needed, as LU FLOPs scales with n^3 opposed to just n. This is why LU decomposition exceeds the available memory at its disposal. Its important to mention that external factors will also affect the time estimations.

References

[1] "Numerical methods for engineers-tridiagonal systems of algebraic equations." http://folk.ntnu.no/leifh/teaching/tkt4140/._main040.html, January 2020. Accessed on 2020-09-07.

[2] "Lu decomposition of matrices." http://www.math4all.in/public_html/linear%20algebra/chapter2.7.html, July 2015. Accessed on 2020-09-07.

[3] "Fys3150 lecture notes." https://github.com/CompPhysics/ComputationalPhysics/tree/master/doc/HandWrittenNotes. Accessed on 2020-09-02.

[4] "Code examples for thomas algorithm and lu decomposition." https://github.com/CompPhysics/ComputationalPhysics/tree/master/doc/Projects/2020/Project1/CodeExamples. Accessed on 2020-09-04.