Project 1 FYS 4150

Kjetil Karlsen, Vilde Mari Reinertsen, Sigbjørn Lundesgaard Foss

September 12, 2017

Abstract

1 Introduction

2 Theory

In this project we will solve the general one-dimensional Poisson's equation (Eq. 1) for $f(x) = 100e^{-10x}$ and $x \in [0,1]$. The boundary conditions are Dirichlet bondary conditions. That means that $u(0) = u(1) = 0$.

2.1 Rewriting the problem to a matrix problem

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

We rewrite the second derivative by switching from the continuous $u(x)$ to n discrete number of points $v(x_i) = v_i$ and then using a Taylor expansion to obtain the new Poisson equation (Eq. 2).

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

where $f_i = f(x_i)$.

The next step is to rewrite Eq. 2 as a linear set of equations of the form of Eq. 3.

$$\mathbf{Av} = \mathbf{b}, \tag{3}$$

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

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 2 & -1 \\ 0 & 0 & \cdots & 0 & -1 & 2 \end{bmatrix},$$

and $b_i = h^2 f_i$.

Writing out the equations gives us:

$$\begin{aligned} i = 1 : &\; -v_0 + 2v_1 - v_2 = f_1 h^2 \\ i = 2 : &\; -v_1 + 2v_2 - v_3 = f_2 h^2 \\ &\vdots\!\vdots \\ i = n : &\; -v_{n-1} + 2v_n - v_{n+1} = f_n h^2 \end{aligned}$$

where $v_0 = 0$ and $v_{n+1} = 0$.

Adding some zeros to the equations for illustrative purposes:

$$\begin{aligned} i = 1 : &\quad 2v_1 - v_2 + 0 + 0 + 0 + 0 + \cdots + 0 = f_1 h^2 \\ i = 2 : &\quad -v_1 + 2v_2 - v_3 + 0 + 0 + \cdots + 0 = f_2 h^2 \\ i = 3 : &\quad 0 - v_2 + 2v_3 - v_4 + 0 + \cdots + 0 = f_3 h^2 \\ \vdots &\qquad\qquad \vdots \\ i = n-1 : &\quad 0 + \cdots + 0 - v_{n-2} + 2v_{n-1} - v_n = f_{n-1} h^2 \\ i = n : &\quad 0 + \cdots + 0 + 0 + 0 - v_{n-1} + 2v_n = f_n h^2 \end{aligned}$$

We can now recognize the equations as a product between a matrix and a vector that equals another vector.

We define vectors that fits like this:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \qquad \tilde{\mathbf{b}} = \begin{bmatrix} f_1 h^2 \\ f_2 h^2 \\ \vdots \\ f_n h^2 \end{bmatrix}$$

The matrix is the matrix $\mathbf{A}$. Then we have:

$$\begin{bmatrix} 2 & -1 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 2 & -1 \\ 0 & 0 & \cdots & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} f_1 h^2 \\ f_2 h^2 \\ f_3 h^2 \\ \vdots \\ f_{n-1} h^2 \\ f_n h^2 \end{bmatrix} \implies \mathbf{Av} = \mathbf{b}$$

2.2 An analytical solution

The Poisson equation (Eq. 1) with our function $f(x) = 100e^{-10x}$ has an analytical solution (Eq. 4).

$$u(x) = 1 - \left(1 - e^{-10}\right) x - e^{-10x} \tag{4}$$

That can be shown by putting it into the Poisson equation (Eq. 1). Then we first have to find the second derivative:

$$\begin{aligned} u(x) &= 1 - \left(1 - e^{-10}\right) x - e^{-10x} \\ u'(x) &= \left(1 - e^{-10}\right) + 10e^{-10x} \\ u''(x) &= -100e^{-10x} \end{aligned}$$

Second, we put the second derivative into Eq. 1:

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

3 Method

- algorithm
- Theoretical models and technicalities

3.1 Algorithm

The first algorithms that we used are based on Gaussian elimination. This method starts by making the matrix upper triangular, thereafter one can solve the set of linear equations by starting at the bottom of the matrix where there are only one unknown.

Our matrix is a tridiagonal matrix and therefore, just a few matrix operations were necessary to make the matrix upper triangular. Those are called forward substitution. The process of solving the set of equation from the bottom is called backward substitution.

3.2 The general tridiagonal matrix

The algorithm is first written for an equation with a general tridiagonal matrix on the short form:

$$\mathbf{Av} = \mathbf{b}$$

The full form looks like this:

$$\begin{bmatrix} d_1 & e_1 & 0 & \cdots & 0 & 0 \\ c_1 & d_2 & e_2 & 0 & \cdots & 0 \\ 0 & c_2 & d_3 & e_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & c_{n-2} & d_{n-1} & e_{n-1} \\ 0 & 0 & \cdots & 0 & c_{n-1} & d_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix}$$

The method used to solve the equation numerically was to first to a forward substitution to obtain a upper triangular matrix and then a backwards substitution to obtain the solution.

The algorithm preformed on the matrix values was:

The forward substitution:
The values on the diagonal:

$$\tilde{d}_i = d_i - \left(\frac{c_i \cdot e_i}{\tilde{d}_{i-1}} \right)$$

The values on the right side of the equation:

$$\tilde{b_i} = b_i - \left(\frac{\tilde{b}_{i-1} \cdot e_i}{\tilde{d}_{i-1}} \right)$$

for $i = 2, \ldots, n$

The backward substitution:

$$v_i = \frac{\left(\tilde{b}_i + v_{i+1} \right)}{\tilde{d}_i}$$

for $i = 2, \ldots, n$. The boundary values are known ($v_0 = v_{n+1} = 0$). v_n was calculated first and like this:

$$v_n = \frac{\tilde{b}_n}{\tilde{d}_n}$$

3.3 This specific tridiagonal matrix

The matrix in our case had the same values on the different diagonals:

$$d_i = 2 \qquad e_i = -1 \qquad c_i = -1$$

We can then pre-calculate some of the operations in the algorithm and if becomes then:

The forward substitution:
The values on the diagonal:

$$\tilde{d}_i = d_i - \left(\frac{c_i \cdot e_i}{\tilde{d}_{i-1}} \right) = 2 - \left(\frac{(-1) \cdot (-1)}{\tilde{d}_{i-1}} \right) =$$

$$2 - \frac{1}{\tilde{d}_{i-1}} = \frac{i+1}{i}$$

The last expression can be shown to be true, by calculating some of the first values:

$$\tilde{d}_2 = 2 - \frac{1}{2} = \frac{3}{2} = \frac{2+1}{2}$$

$$\tilde{d}_3 = 2 - \frac{1}{\frac{3}{2}} = 2 - \frac{2}{3} = \frac{12}{6} - \frac{4}{6} = \frac{8}{6} = \frac{4}{3} = \frac{3+1}{3}$$

The values on the right side of the equation:

$$\tilde{b_i} = b_i - \left(\frac{\tilde{b}_{i-1} \cdot e_i}{\tilde{d}_{i-1}} \right) = b_i - \left(\frac{\tilde{b}_{i-1} \cdot (-1)}{\tilde{d}_{i-1}} \right) = b_i + \left(\frac{\tilde{b}_{i-1}}{\tilde{d}_{i-1}} \right)$$

for $i = 2, \ldots, n$.

The backward substitution:

$$v_i = \frac{\left(\tilde{b}_i + v_{i+1} \right)}{\tilde{d}_i}$$

for $i = 2, \ldots, n-2$. The boundary values are known ($v_0 = v_{n+1} = 0$). v_n was calculated first and like this:

$$v_n = \frac{\tilde{b}_n}{\tilde{d}_n}$$

3.4 LU-composition

The last method that we tried out is called LU decomposition. When using this method we factorized our matrix into two other matrices, one a lower triangular matrix and the other a upper triangular matrix. Here is an example for a 4×4-matrix:

$$A = LU = \begin{bmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix}$$

LU decomposition exists it the determinant of the matrix is non-zero.

The method goes like this:

$$\mathbf{Av} = \mathbf{b} \implies \mathbf{LUv} = \mathbf{b}$$

$$\mathbf{Uv} = \mathbf{L}^{-1}\mathbf{b} = \mathbf{w}$$

We first solve:

$$\mathbf{L}^{-1}\mathbf{b} = \mathbf{w}$$

Then we can easily get $\mathbf{x}$, because of the shape of $\mathbf{U}$, from:

$$\mathbf{Ux} = \mathbf{w} \implies \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

The bottom equation has only one unknown, so we start there.

4 Result

4.1 The computer time

This section should include the CPU time of the three different methods.

n	general(s)	special(s)	LU(s)
1e1	9.0e-06	1.0e-06	1.8e-04
1e2	1.0e-05	3.5e-05	2.6e-03
1e3	5.9e-05	4.5e-05	2.5e-01
1e4	4.4e-04	2.9e-04	1.4e+02

Table 4.1: The total time of the algorithm in the general case, the special case and using LU-decomposition

These numbers are for calculating $n = 10$ to $n = 10^7$ and not printing anything.

Unspecialised: 'CPU time: 2.17188 s'

Specialized: 'CPU time: 2.04688 s'

4.2 The error

Too find out how the error developed with the number of mesh points, the step length, we calculated the maximum relative error with different step lengths. The results from these calculations are in Tb. 4.2 and in Fig. 4.1.

Table 4.2: This is a table listing the log of the error and the log of the associated h value. We can see that the smallest error, log(RelativeError) = −7, is accomplished when $\log(h) = -4$.

log(h):	log(RelativeError):
-1.04	-1.18
-2.00	-3.09
-3.00	-5.08
-4.00	-7.09
-5.00	-6.57
-6.00	-4.65

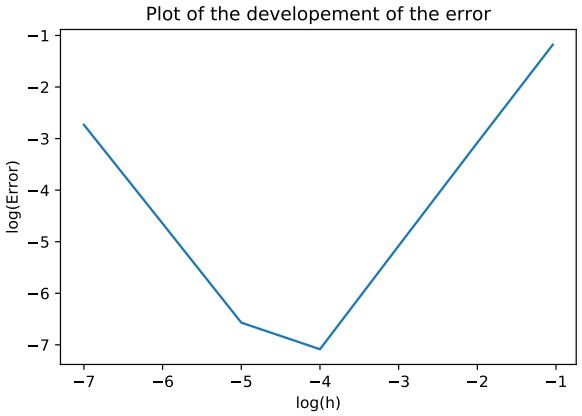

Figure 4.1: This is a plot of the log of the error versus the log of the associated h value. We can see that the smallest error, log(RelativeError) = −7, is accomplished when $\log(h) = -4$.

5 Discussion

5.1 Floating point operations

A floating point operation is either addition, subtraction, multiplication or division. Comparing the number of floating point operations is a way to compare the cost of different algorithms. That way, one can find the most efficient algorithm to solve a problem.

5.1.1 Algorithm of the general tridiagonal matrix

The number of floating point operations in the Gaussian elimination method with the general tridiagonal can be found from the algorithm. The forward substitution looked like this:

$$\tilde{d}_i = d_i - \left(\frac{c_i \cdot e_i}{\tilde{d}_{i-1}}\right)$$

$$\tilde{b}_i = b_i - \left(\frac{\tilde{b}_{i-1} \cdot e_i}{\tilde{d}_{i-1}}\right)$$

for $i = 2, \ldots, n$

The first expression involves three floating point operations and these were preformed $n - 1$ times.

The second expression also involves three floating point operations, so the sum of floating point operations from the forward substitution is:

$$3(n-1) + 3(n-1) = 6(n-1)$$

The algorithm for the backward substitution looked like this:

$$v_i = \frac{\left(\tilde{b}_i + e_i v_{i+1}\right)}{\tilde{d}_i}$$

for $i = 2, \ldots, n-1$. The last position, calculated first:

$$v_n = \frac{\tilde{b}_n}{\tilde{d}_n}$$

The first expression involves three floating point operations and is preformed $n-2$ times. The last involves one floating point operation and is preformed only once. The sum of floating point operations for the backward substitution is then:

$$3(n-2)+1 = 3n-5$$

The sum of floating point operation for the whole Gaussian elimination is then:

$$6n-6+3n-5 = 9n-11 \approx 9n$$

5.1.2 Algorithm of the specific tridiagonal matrix

The number of floating point operations in the Gaussian elimination method with the specific tridiagonal can be found from the algorithm. The forward substitution looked like this:

$$\tilde{d}_i = \frac{i+1}{i}$$

$$\tilde{b}_i = b_i + \frac{\tilde{b}_i}{\tilde{d}_i}$$

for $i = 2, \ldots, n$

The first expression involves no floating point operations, it can be pre-calculated and does not count.

The second expression also involves two floating point operations, so the sum of floating point operations from the forward substitution is:

$$2(n-1) = 2n-2$$

The algorithm for the backward substitution looked like this:

$$v_i = \frac{\left(\tilde{b}_i + v_{i+1}\right)}{\tilde{d}_i}$$

for $i = 2, \ldots, n-1$. The last position, calculated first:

$$v_n = \frac{\tilde{b}_n}{\tilde{d}_n}$$

The first expression involves two floating point operations and is preformed $n-2$ times. The last involves one floating point operation and is preformed only once. The sum of floating point operations for the backward substitution is then:

$$2(n-2)+1 = 4n-3$$

The sum of floating point operation for the whole Gaussian elimination is then:

$$2n-2+2n-3 = 4n-5 \approx 4n$$

5.1.3 Gaussian elimination in general

The Gaussian elimination method of solving a set of linear equations requires n^3 floating point operations for a regular $n \times n$-matrix, that is not tridiagonal or any other special type of matrix.

5.1.4 LU decomposition

The floating point operations for the LU decomposition method can be separated in two parts; the decomposing of the matrix into an upper triangular matrix multiplied with a lower triangular matrix and the calculation of the set of equation with different 'right hand side'-vectors.

The number of floating point operations required to LU decompose a matrix scales to n^3 for a $n\times$-matrix. After the decomposition the floating points of the calculation of the set of equations scales to n^2 for a $n \times n$-matrix.

If one was to use the LU decomposition method for a $10^5 \times 10^5$-matrix it would require $\left(10^5\right)^3 + \left(10^5\right)^2 \approx 10^{15}$ floating point operations. If a computer calculates 10^9 floating points per second (flops) it would take $\frac{10^{15}\text{flop}}{10^9\text{flops}} = 10^6$ seconds, which is approximately 12 days. That is too long.

5.1.5 Comparison

It is easy to see that the Gaussian elimination method is the most effective method for our problem with a tridiagonal matrix. It needs the least number of floating point operations. It is also possible to make the algorithm even more effective by specializing it to our tridiagonal matrix which has the same values on the diagonal and on the super- and sub-diagonal.

If however we had a regular random matrix and were to solve the equations for many different 'right-hand-side'-vectors, the best method of these would be the LU decomposition. Then we would only have to decompose the matrix once, which require n^3 floating point operations for an $n\times n$-matrix, and for every new 'right-hand-side'-vector it would only require n^2 floating point operations. Compared with the Gaussian elimination method that requires n^3 floating point operations to solve the set of equations every time.

6 Conclusion

7 References