project 1 - Fys3150/4150

Simon Millerjord
Shako Farhad

September 2016

Introduction

In this project, we are going to solve the following linear second order differential equation: In this project we will solve the one-dimensional Poisson equation with Dirichlet boundary conditions by rewriting it as a set of linear equations.
To be more explicit we will solve the equation

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

and 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 $h = 1/(n+1)$. We have then the boundary conditions $v_0 = v_{n+1} = 0$. We approximate the second derivative of u with

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

where $f_i = f(x_i)$.

Project 1 Excercises

a

We need to show that $-\frac{v_{i+1} - 2v_i + v_{i-1}}{h^2} = f_i$ can be written as a $Ax = \tilde{b}$. We start with rewriting the finite difference and setting $i = 1, 2, 3, .., n$.

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

$$-v_0 + 2v_1 - v_2 = h^2 f_1$$
$$-v_1 + 2v_2 - v_3 = h^2 f_2$$
$$-v_2 + 2v_3 - v_4 = h^2 f_3$$
$$\vdots$$
$$-v_{n-1} + 2v_n - v_{n+1} = h^2 f_n$$

The first thing we observe is that $v_0 = v_{n+1} = 0$ from the initialconditions. And to see the pattern we need to extend the system of linear equations like this:

$$0 + 2v_1 - v_2 + 0 + 0 + 0 + \cdots + 0 = h^2 f_1$$

$$0 - v_1 + 2v_2 - v_3 + 0 + 0 + \cdots + 0 = h^2 f_2$$
$$0 + 0 - v_2 + 2v_3 - v_4 + 0 + \cdots + 0 = h^2 f_3$$
$$\vdots$$
$$0 + 0 + 0 + \ldots 0 + -v_{n-1} + 2v_n + 0 = h^2 f_n$$

We can see that the system of equations is zero in the first and last column and this is because $v_0 = v_{n+1} = 0$ are known. So we end up with this:

$$2v_1 - v_2 + 0 + 0 + 0 + \cdots + 0 = h^2 f_1$$
$$-v_1 + 2v_2 - v_3 + 0 + 0 + \cdots + 0 = h^2 f_2$$
$$0 - v_2 + 2v_3 - v_4 + 0 + \cdots + 0 = h^2 f_3$$
$$\vdots$$
$$0 + 0 + \ldots 0 + -v_{n-1} + 2v_n = h^2 f_n$$

We can now see the matrix structure and if we extract the v_i for $i = 1, 2, 3, \ldots, n$, we get this:

$$A = \begin{pmatrix} 2 & -1 & 0 & \ldots & \ldots & \ldots & 0 \\ -1 & 2 & -1 & \ldots & \ldots & \ldots & 0 \\ 0 & -1 & 2 & -1 & \ldots & \ldots & 0 \\ 0 & \vdots & -1 & 2 & -1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \vdots & \vdots & \vdots & -1 & 2 & -1 \\ 0 & 0 & 0 & \ldots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ \vdots \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} \tilde{b}_1 \\ \tilde{b}_2 \\ \tilde{b}_3 \\ \vdots \\ \vdots \\ \vdots \\ \tilde{b}_n \end{pmatrix}$$

b)

We start by doing a forward substitution. Which means that we first need to get rid of the a_i with the help of the row above the a_i that is getting removed. We will do forward substitution for a few steps and see a pattern. We denote the rows by R_i.

$$R_2 - \frac{a_1}{b_1} \cdot R_1 \quad \Longrightarrow \quad 0, \quad b_2 - \frac{a_1 c_1}{b_1}, \quad c_2, \quad f_1$$

$$R_3 - \frac{a_2}{b_2 - \frac{a_1 c_1}{b_1}} \cdot R_2 \quad \Longrightarrow \quad 0, \quad 0, \quad b_3 - \frac{a_2 c_2}{b_2 - \frac{a_1 c_1}{b_1}}, \quad c_3, \quad f_2 - \frac{a_1}{b_1} f_1$$

$$R_4 - \frac{a_3 c_3}{b_3 - \frac{a_2 c_2}{b_2 - \frac{a_1 c_1}{b_1}}} \cdot R_3 \quad \Longrightarrow \quad 0, \quad 0, \quad 0, \quad b_4 - \frac{a_3 c_3}{b_3 - \frac{a_2 c_2}{b_2 - \frac{a_1 c_1}{b_1}}}, \quad c_4, \quad f_3 - \frac{a_2}{b_2}(f_2 - \frac{a_1}{b_1} f_1)$$

We can see the pattern that is like this:

$$\tilde{b}_1 = b_1, \quad \tilde{f}_1 = f_1$$

$$\tilde{b}_2 = b_2 - \frac{a_1 c_1}{\tilde{b}_1}, \quad \tilde{f}_2 = f_2 - \frac{a_1}{b_1} \tilde{f}_1$$

$$\tilde{b}_3 = b_3 - \frac{a_2 c_2}{\tilde{b}_2}, \quad \tilde{f}_3 = f_3 - \frac{a_2}{b_2} \tilde{f}_2$$

$$\tilde{b}_4 = b_4 - \frac{a_3 c_3}{\tilde{b}_3}, \quad \tilde{f}_4 = f_4 - \frac{a_3}{b_3}\tilde{f}_3$$

We get these general algorithms:

$$\text{Initial values:} \qquad \tilde{b}_1 = b_1, \quad \tilde{f}_1 = f_1$$

$$\text{General algorithm:} \qquad \tilde{b}_i = b_i - \frac{a_{i-1} c_{i-1}}{\tilde{b}_{i-1}}, \quad \tilde{f}_i = f_i - \frac{a_{i-1}}{b_{i-1}}\tilde{f}_{i-1}$$

After these operations, we end up with this matrix:

$$A = \begin{pmatrix} \tilde{b}_1 & c_1 & 0 & \dots & \dots & \dots & 0 \\ 0 & \tilde{b}_2 & c_2 & \dots & \dots & \dots & 0 \\ 0 & 0 & \tilde{b}_3 & c_3 & \dots & \dots & 0 \\ 0 & \vdots & 0 & \tilde{b}_4 & c_4 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \vdots & \vdots & \vdots & 0 & \tilde{b}_{n-1} & c_{n-1} \\ 0 & 0 & 0 & \dots & 0 & 0 & \tilde{b}_n \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ \vdots \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} \tilde{f}_1 \\ \tilde{f}_2 \\ \tilde{f}_3 \\ \vdots \\ \vdots \\ \vdots \\ \tilde{f}_n \end{pmatrix}$$

Now we can see that we need to do backwards substitution. We do it in a similar way as above:

$$\frac{1}{\tilde{b}_n} R_n \quad \Longrightarrow \quad 0, \quad 1, \quad \frac{\tilde{f}_n}{\tilde{b}_n}$$

$$R_{n-1} - c_{n-1} \cdot R_n \quad \Longrightarrow \quad 0, \quad \tilde{b}_{n-1}, \quad 0, \quad \tilde{f}_{n-1} - c_{n-1}\frac{\tilde{f}_n}{\tilde{b}_n}$$

$$\frac{1}{\tilde{b}_{n-1}} R_{n-1} \quad \Longrightarrow \quad 0, \quad 1, \quad 0, \quad \frac{\tilde{f}_{n-1} - c_{n-1}\frac{\tilde{f}_n}{\tilde{b}_n}}{\tilde{b}_{n-1}}$$

$$R_{n-2} - c_{n-2} \cdot R_{n-1} \quad \Longrightarrow \quad 0, \quad \tilde{b}_{n-2}, \quad 0, \quad 0, \quad \tilde{f}_{n-2} - c_{n-2}\frac{\tilde{f}_{n-1} - c_{n-1}\frac{\tilde{f}_n}{\tilde{b}_n}}{\tilde{b}_{n-1}}$$

$$\frac{1}{\tilde{b}_{n-2}} R_{n-2} \quad \Longrightarrow \quad 0, \quad 1, \quad 0, \quad 0, \quad \frac{\tilde{f}_{n-2} - c_{n-2}\frac{\tilde{f}_{n-1} - c_{n-1}\frac{\tilde{f}_n}{\tilde{b}_n}}{\tilde{b}_{n-1}}}{\tilde{b}_{n-2}}$$

We can see that there is a pattern, and following the pattern gives this general algorithms:

$$\text{Initial value given by:} \quad \tilde{\tilde{f}}_n = \frac{\tilde{f}_n}{\tilde{b}_n}$$

$$\text{General algorithm:} \quad \tilde{\tilde{f}}_i = \frac{\tilde{f}_i - c_i \cdot \tilde{\tilde{f}}_{i+1}}{\tilde{b}_i}$$

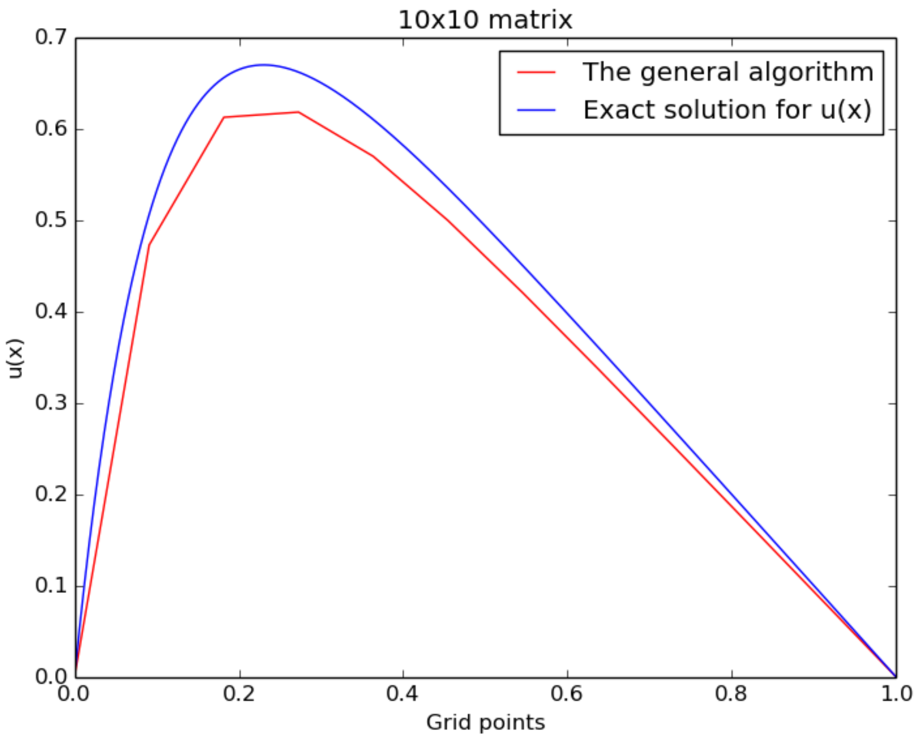

Figure 1: Here is a figure showing how well the solution for the 10x10 matrix when using our general algorithm corresponds to the exact solution to the second order differential equation.

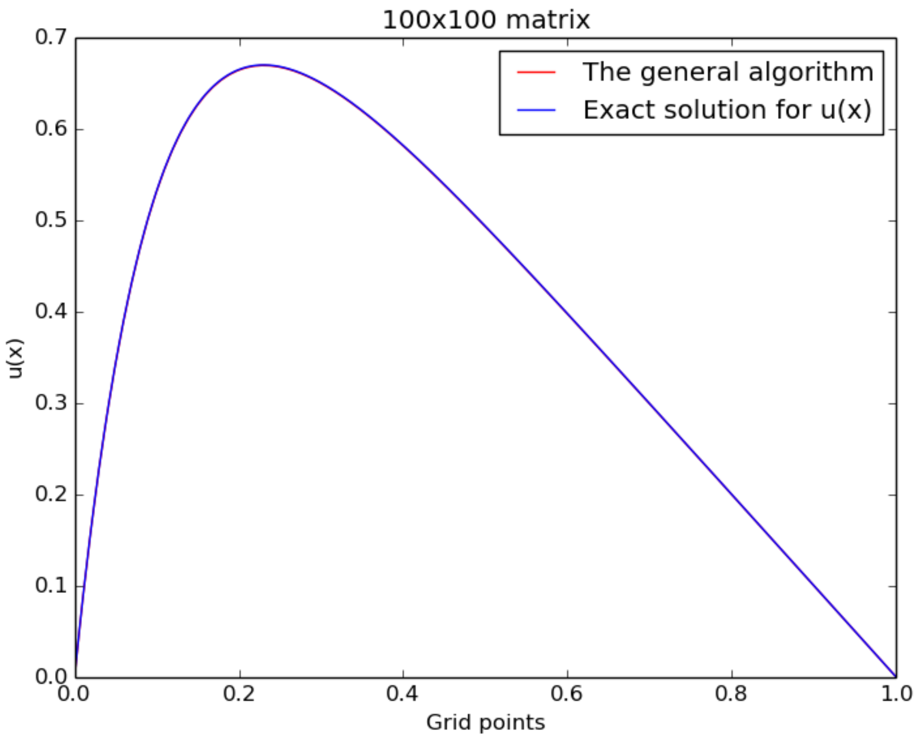

Figure 2: Here is a figure showing how well the solution for the 100x100 matrix when using our general algorithm corresponds to the exact solution to the second order differential equation.

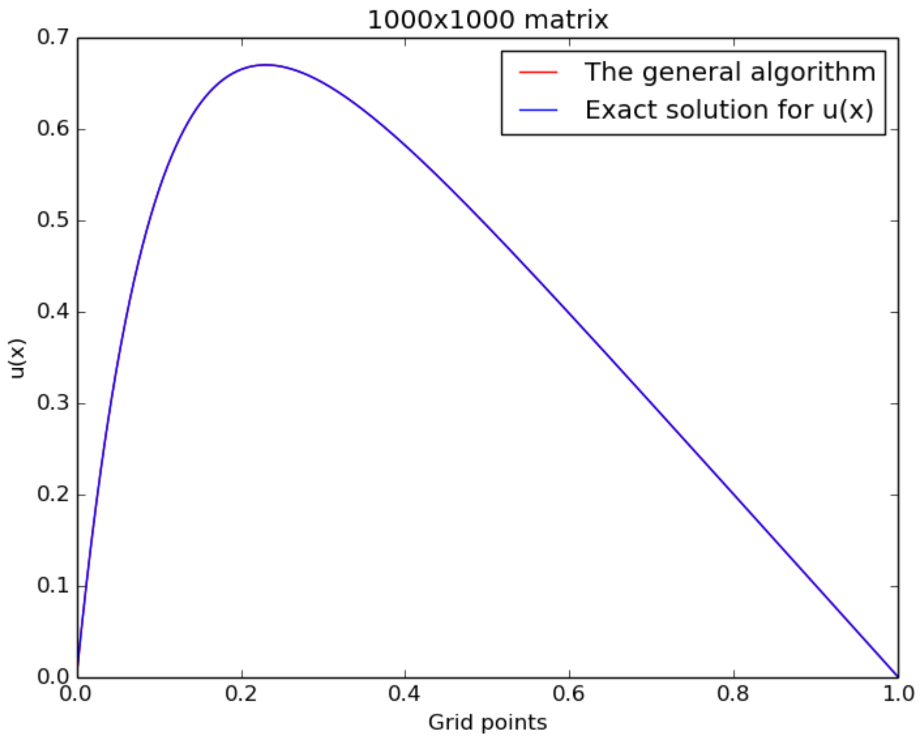

Figure 3: Here is a figure showing how well the solution for the 1000x1000 matrix when using our general algorithm corresponds to the exact solution to the second order differential equation.

We see from the plots that the error between the values from a 10x10 matrix and the exact values are visually noticeable, and the difference in precision between a 10x10 matrix and a 100x100 matrix is great.
The error in the 100x100 matrix and 1000x1000 matrix plots are not visually noticeable, and even though there is a better precision for the 1000x1000 matrix than for the 100x100 matrix, the difference between the errors are incredibly small compared to the difference between a 10x10 matrix and a 100x100 matrix.

c)

In the specialized case, we set $b_i = 2$ and $a_i = c_i = -1$, $g = h^2 f_i$.

$$\tilde{b}_1 = 2, \quad \tilde{b}_2 = 2 - \frac{(-1)(-1)}{2} = \frac{3}{2}$$

$$\tilde{b}_3 = 2 - \frac{1}{\frac{3}{2}} = \frac{4}{3}, \quad \tilde{b}_4 = 2 - \frac{1}{\frac{4}{3}} = \frac{4}{3}$$

$$\tilde{b}_5 = 2 - \frac{1}{\frac{5}{4}} = \frac{6}{5} \quad \Longrightarrow \quad \tilde{b}_i = \frac{i+1}{i}$$

$$\tilde{f}_i = f_i + \frac{i}{i+1}\tilde{f}_{i-1}$$

where $\frac{i}{i+1}$ is already calculated.
$\tilde{b}_i$ has 2 floating point operations in the specialized case.
The total number of floating point operations is then:

$$2n + 2n + 3n - 2 - 1 - 3 = 7(n-1) + 2$$

The following is the computing-time for the different algorithms:

```
n = 1000,         time = 0.00
n = 10000,        time = 0.00
n = 100000,       time = 0.00
n = 1000000,      time = 0.05
n = 10000000,     time = 0.41
n = 100000000,    time = 4.09
```

This is the cpu time for the specialized algorithm

```
n = 1000,         time = 0.00
n = 10000,        time = 0.00
n = 100000,       time = 0.01
n = 1000000,      time = 0.05
n = 10000000,     time = 0.45
n = 100000000,    time = 4.44
```

This is the cpu time for general algorithm

The time used by our algorithms are so small that the computer gives a cpu-time = 0 in both instances. The difference between the algorithms are even smaller, so it will be difficult to represent that time with a computer with limited bits. That is why we ran the program for a 100-times bigger n. This gave us the above results. It is clear that the specialized algorithm is somewhat faster than the general algorithm. This is what we expected.

d)

We decided to take a different approach to checking how well our scheme solves the system. We run the scheme many times with, halving the value of $h = \frac{1}{n+1}$ before each new run. This is basically the same as doubling n before each new run. Then we use the L^2-norm to find the error for each run like this:

$$E_i = \sqrt{\sum_{j=0}^{n+1} (v_j - u_j)^2}$$

Afterwards we make assume that we can write the error in this form:

$$E_i = h_i^r$$

where r is the convergence rate we want to find. But this is not enough if we want to compare previous runs. So we look at this ratio:

$$\frac{E_{i+1}}{E_i} = \frac{h_{i+1}^r}{h_i^r}$$

Since it is r we want to find, we will solve the equation and get an expression for r.

$$ln(\frac{E_{i+1}}{E_i}) = ln(\frac{h_{i+1}^r}{h_i^r})$$

$$ln(\frac{E_{i+1}}{E_i}) = ln(h_{i+1}^r) - ln(h_i^r)$$

$$ln(\frac{E_{i+1}}{E_i}) = r(ln(h_{i+1}) - ln(h_i))$$

$$ln(\frac{E_{i+1}}{E_i}) = r \cdot ln(\frac{h_{i+1}}{h_i})$$

$$r = \frac{ln(\frac{E_{i+1}}{E_i})}{ln(\frac{h_{i+1}}{h_i})}$$

Now we also see that we will get a different r for each new run, therefore we add the i subscript.

$$r_i = \frac{ln(\frac{E_{i+1}}{E_i})}{ln(\frac{h_{i+1}}{h_i})}$$

We run the scheme 4 times and with $n = 100$ as the start value. This is what we get.

Step length:		*Convergence rate:*
h = 0.00990099—-> h = 0.00249377	*gives*	*r = 1.49967*
h = 0.00249377—-> h = 0.00062461	*gives*	*r = 1.49998*
h = 0.00062461—-> h = 0.000156226	*gives*	*r = 1.50002*
h = 0.000156226—-> h = 3.9061e-05	*gives*	*r = 1.50078*

Then we run the scheme again 5 times, now with $n = 1000000$:

Step length:		*Convergence rate:*
h = 9.9999e-06—-> h = 2.49999e-06	*gives*	*r = -3.33604*
h = 2.49999e-06—-> h = 6.25e-07	*gives*	*r = -2.39359*
h = 6.25e-07 —-> h = 1.5625e-07	*gives*	*r = -0.382724*
h = 1.5625e-07—-> h = 3.90625e-08	*gives*	*r = -3.65135*

We can see that r_i is converging towards $\frac{3}{2}$, and this means that our scheme converges towards the exact solution at the order of $\frac{3}{2}$. But for large n or very small h, r diverges. This indicates that our scheme does not approximate better with smaller step lengths. Also the convergence rate should have been converging to 2 because we use two parts of the Taylor series, but it is converging to $\frac{3}{2}$ instead. This either indicates that we have implemented something wrong, or that our method is not optimal.
We also implemented the ϵ_i function. The results did not tell us anything. This is the output of the program:

Step length:		*Maximum Error:*
h = 9.99999e-07	*gives*	*error = 0*
h = 1e-07	*gives*	*error = 0*

As you can see, the program did not give us any informasjon. Most of the values from the log function is negative, and therefore the highest value is 0. And we do not know what this tells us exactly.

e)

We will compare our results from our tridiagonal solver to the LU-decomposition results to see which of the two is the fastest. For the LU-decomp., we use the *lu*-function from Armadillo library for C++.
The time used for the LU-decomposition from Armadillo, and our spesialized algorithm is diplayed below. It is a significant difference in time.
LU decomposition solver time was: 3.24
Specialized algorithm time was: 0