Project 1

Sverre Manu Johansen, Søren Blåberg Tvingsholm, Lisa Julianne Nystad

August 2022

Git hub repository: Project 1 - files

Problem 1

The one-dimensional Poisson equation can be written as

$$-\frac{d^2u}{dx^2} = f(x) \quad (1)$$

Here is a known function (the source term). Our task is to find the function that satisfies this equation for a given boundary condition. The specific setup we will assume is the following:

source term:$f(x) = 100e^{-10x}$
range: $x \in [0,1]$
boundary condition: $u(0) = 0$ and $u(1) = 1$

Check analytically that an exact solution to Eq. (1) is given by

$$u(x) = 1 - (1 - e^{-10})x - e^{-10x}$$

To get the exact solution, we find the derivative of u(x) twice and compare it to f(x).

$$u(x) = 1 - (1 - e^{-10})x - e^{-10x}$$
$$u'(x) = e^{-10} - 1 + 10e^{-10x}$$
$$u''(x) = -100e^{-10x} \rightarrow -u''(x) = 100e^{-10x}$$

From this, we derive that $-\frac{d^2u}{dx^2} = f(x) = 100e^{-10x}$.

Problem 2

Make a plot of the exact solution above.
See opg2.cpp for exact solution and oppgave2.py for the plot, in Git Hub repository under Sources. The plot is represented as figure 1, at page 9.

Problem 3

Derive a discretized version of the Possion equation. You don't have to show every step of your derivation, but include the main steps and a couple of sentences to explain the logic.

Taylor expansion:

$$u(x \pm h) = \sum_{n=0}^{\infty} \frac{1}{(n!)} u^{(n)} h^n$$

We want to find the Taylor expansion of the second order. We need two equations for this, as this is a three point operation.

$$u(x+h) = u(x) + u'(x)h + \frac{1}{2}u''h^2 + O(h^3)u(x-h) = u(x) - u'(x)h + \frac{1}{2}u''h^2 + O(h^3)$$

To find the second order expansion, we simply add the two equations and solve for $u''(x)$.

$$u(x+h) + u(x-h) = 2u(x) + u''(x)h^2 + O(h^4)$$
$$u''(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + O(h^4)$$

The discretized version of this equation is:

$$u''(x) = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + O(h^2) = -f_i$$
$$f_i h^2 = -v_{i+1} + 2v_i - v_{i-1} \quad , v = u$$

Problem 4

Show that you can rewrite your discretized equation as a matrix equation on the form

$$\mathbf{A}\vec{v} = \vec{g}$$

where $\mathbf{A}$ is a tridiagonal matrix with the subdiagonal, main diagonal and superdiagonal specified by the signature $(-1, 2, -1)$. Make sure to specify how an element of $\vec{g}$ is related to the variables in the original differential equation.

We assume that n = 5, which gives us 6 points; i = 0,1,...,5 and we get the following equations for $\mathbf{A}\vec{v} = \vec{g}$:

$$\begin{aligned}
i = 1 \rightarrow 2v_1 - v_2 \quad & \quad | = f_1h^2 + v_0 = g_1 \\
-v_1 + 2v_2 - v_3 \quad & \quad | = f_2h^2 = g_2 \\
-v_2 + 2v_3 - v_4 \quad & \quad | = f_3h^2 = g_3 \\
i = 4 \rightarrow 2v_4 - v_3 \quad & \quad | = f_4h^2 + v_5 = g_4 \\
& \quad v_0 = v_5 = 0
\end{aligned}$$

The boundary conditions are known $u(0) = u(1) = 0$, we set $v_0 = v_5 = 0$. This makes up the vector $\vec{g}$.

$$\vec{g} = \begin{bmatrix} 2v_1 - v_2 \\ -v_1 + 2v_2 - v_3 \\ -v_2 + 2v_3 - v_4 \\ 2v_4 - v_3 \end{bmatrix}$$

The variable i shows us which column our expressions belong to in our matrix **A**. If we insert our equations into a matrix $\mathbf{A}\vec{v}$, we will get the following output:

$$\mathbf{A}\vec{v} = \begin{bmatrix} 2v_1 & -v_2 & 0 & 0 \\ -v_1 & 2v_2 & -v_3 & 0 \\ 0 & -v_2 & 2v_3 & -v_4 \\ 0 & 0 & -v_3 & 2v_4 \end{bmatrix}$$

If we split this up into a separate matrix **A** and vector $\vec{v}$, we get these expressions

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

$$\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}$$

Oppgave 5

a)

The relation between m and n can be written as m = n+2. This is because in matrix A, we do not take into consideration the end points v_0 and v_5.

b)

When solving the equation we find the part of our complete solution $\vec{v}^*$ that is equal to all the inner points of $\vec{v}^*$. In other words we find v_1 to v_4.

Oppgave 6

Now we'll forget about the Poisson equation for a moment, and simply consider the equation $\mathbf{A}\vec{v} = \vec{g}$ where $\mathbf{A}$ is a general tridiagonal matrix, with vectors $\vec{a}, \vec{b}$ and $\vec{c}$ representing the subdiagonal, main diagonal and superdiagonal, respectively. That A is a general tridiagonal matrix means that every element of $\vec{a}, \vec{b}$ and $\vec{c}$ can be different.

a) Write down an algorithm for solving $\mathbf{A}\vec{v} = \vec{g}$.

The *general* algorithm to solve an equation $\mathbf{A}\vec{v} = \vec{g}$, where $\mathbf{A}$ is a nxn-matrix with n=4 is solved by using Gaussian elimination, or more specifically forward substitution and backwards substitution on the following system:

$$\begin{bmatrix} \tilde{b_1} & c_1 & 0 & 0 & | \, g_1 \\ a_2 & b_2 & c_2 & 0 & | \, g_2 \\ 0 & a_3 & b_3 & c_3 & | \, g_3 \\ 0 & 0 & a_4 & b_4 & | \, g_4 \end{bmatrix}$$

We use forward substitution to make the matrix $\mathbf{A}$ upper triangular, that means removing the vector $\vec{a}$ along the subdiagonal:
We can start by substituting $b_1 = \tilde{b_1}$ and $g_1 = \tilde{g_1}$.

$$R_2 \to R_2 - \frac{a_2}{\tilde{b_1}} R_1$$

$$\Rightarrow \begin{bmatrix} \tilde{b_1} & c_1 & 0 & 0 & | \, \tilde{g_1} \\ 0 & b_2 - \frac{a_1 c_1}{\tilde{b_1}} & c_2 & 0 & | \, g_2 - \frac{a_1 g_1}{\tilde{b_1}} \\ 0 & a_3 & b_3 & c_3 & | \, g_3 \\ 0 & 0 & a_4 & b_4 & | \, g_4 \end{bmatrix}$$

$$\tilde{b_2} = b_2 - \frac{a_2 c_1}{\tilde{b_1}} \text{ and } \tilde{g_2} = g_2 - \frac{a_2 c_1}{\tilde{g_1}}$$

Continuing with the general case $R_i \to R_i - \frac{a_i}{\tilde{b_{i-1}}} R_{i-1}$ and substituting for $\tilde{b_i} = b_i - \frac{a_i c_{i-1}}{\tilde{b_1}}$ and $\tilde{g_i} = g_i - \frac{a_i g_{i-1}}{\tilde{b_{i-1}}}$, we end up with the upper triangular matrix:

$$\begin{bmatrix} \tilde{b_1} & c_1 & 0 & 0 & | \, \tilde{g_1} \\ 0 & \tilde{b_2} & c_2 & 0 & | \, \tilde{g_2} \\ 0 & 0 & \tilde{b_3} & c_3 & | \, \tilde{g_3} \\ 0 & 0 & 0 & \tilde{b_4} & | \, \tilde{g_4} \end{bmatrix}$$

Backwards substituting provides us with a row-reduced matrix with 1's along the main diagonal. Solving for $\vec{v}$ is also emerging from the matrix, as we are solving it. We know that $\tilde{b_4} v_4 = \tilde{g_4}$, giving us our first v: $v_4 = \frac{\tilde{g_4}}{\tilde{b_4}} = u_4$.

$$\begin{bmatrix} \tilde{b_1} & c_1 & 0 & 0 & |\, \tilde{g_1} \\ 0 & \tilde{b_2} & c_2 & 0 & |\, \tilde{g_2} \\ 0 & 0 & \tilde{b_3} & c_3 & |\, \tilde{g_3} \\ 0 & 0 & 0 & 1 & |\, \frac{\tilde{g_4}}{\tilde{b_4}} \end{bmatrix}$$

Substituting $u_4 = \frac{\tilde{g_4}}{\tilde{b_4}}$, and continuing with $R_3 \to \frac{R_3 - c_3 R_4}{\tilde{b_3}}$.

$$\begin{bmatrix} \tilde{b_1} & c_1 & 0 & 0 & |\, \tilde{g_1} \\ 0 & \tilde{b_2} & c_2 & 0 & |\, \tilde{g_2} \\ 0 & 0 & \frac{\tilde{b_3} - c_3 * 0}{\tilde{b_3}} & \frac{c_3 - c_3 * 1}{\tilde{b_3}} & |\, \frac{\tilde{g_3} - c_3 * u_4}{\tilde{b_3}} \\ 0 & 0 & 0 & 1 & |\, u_4 \end{bmatrix} \to \begin{bmatrix} \tilde{b_1} & c_1 & 0 & 0 & |\, \tilde{g_1} \\ 0 & \tilde{b_2} & c_2 & 0 & |\, \tilde{g_2} \\ 0 & 0 & 1 & 0 & |\, \frac{g_3 - c_3 * u_4}{\tilde{b_3}} \\ 0 & 0 & 0 & 1 & |\, u_4 \end{bmatrix}$$

Substituing for u_3 we get the following equation:

$$u_3 = \frac{\tilde{g_3} - c_3}{\tilde{b_3}} u_4$$

We can conclude that the generalized version of the backwards substitution looks like this:

$$u_i = \frac{\tilde{g_i} - c_i u_{i+1}}{\tilde{b_i}}$$

For i=n-1, n-2,..., 1.

We end up with the matrix,

$$\begin{bmatrix} 1 & 0 & 0 & 0 & |\, u_1 \\ 0 & 1 & 0 & 0 & |\, u_2 \\ 0 & 0 & 1 & 0 & |\, u_3 \\ 0 & 0 & 0 & 1 & |\, u_4 \end{bmatrix}$$

giving us the vector $\vec{v}$:

$$\mathbf{A}\vec{v} = \vec{g} \Rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} \frac{\tilde{g_1} - c_1 u_2}{b_1} \\ \frac{\tilde{g_2} - c_2 u_3}{b_2} \\ \frac{\tilde{g_3} - c_3 u_4}{b_3} \\ \frac{\tilde{g_4}}{\tilde{b_4}} \end{bmatrix}$$

We get the system of equations:

$$v_1 = \frac{\tilde{g_1} - c_1 u_2}{b_1} \tag{1}$$
$$v_2 = \frac{\tilde{g_2} - c_2 u_3}{b_2} \tag{2}$$
$$v_3 = \frac{\tilde{g_3} - c_3 u_4}{b_3} \tag{3}$$
$$v_4 = \frac{\tilde{g_4}}{b_4} \tag{4}$$

Our general algorithm for solving $\mathbf{A}\vec{v} = \vec{g}$ is the product of using our generalized versions for forward and backward substitution on the equation(see problem 7).

b)

To find the number of floating-point operations (FLOPs) we will look at the amount of operations used in our general algorithm for forward and backward substitution:

$$\tilde{b_i} = b_i - \frac{a_i c_{i-1}}{\tilde{b_1}}, \tilde{g_i} = g_i - \frac{a_i c_{i-1}}{\tilde{g_{i-1}}}$$

For i=2, 3,..., n. And

$$v_i = \frac{\tilde{g_i} - c_i}{\tilde{b_i}} v_{i+1}$$

For i=n-1, n-2,..., 1.
We count operations and In the forward equation we do 3 FLOPs two times for n-1 iterations and for the backwards we also use 3 FLOPs once for n-1 iterations, but prior to that we still need to do one iteration to get v_n. Adding these together we get

$$3(n-1) + 3(n-1) + 3(n-1) + 1 = 9n - 8$$

For a high n this ends up being about 9n FLOPs.

Problem 7

a)

See oppgave7.cpp in Git Hub repository under Sources.

b)

See lesfil7b8ab.py in Git Hub repository under Sources. See figure 2. at the bottom of page 9.

Problem 8

a)

See lesfil7b8ab.py in Git Hub repository under Sources. See figure 3 at the top of page 10.

b)

See lesfil7b8ab.py in Git Hub repository under Sources. See figure 4. at the bottom of page 10.

c)

Our maximum relative error for different n are visible in Figure 3. For n=10 we have about -3.6, for n=100 we have about -8 and for n=1000 we have about -10.2 error. We expect our maximum relative error to become bigger for higher values of n since errors mostly consist of round-off errors, and the amount of round-off errors rise with the amount of FLOPs.

Problem 9

a)

Spezialice the algorithm from problem 6 for our special case where **A** is specified by the signature (-1, 2, -1), that is, with $\vec{a} = (-1, -1, ..., -1)$, $\vec{b} = (2, 2, ..., 2)$ and $\vec{c} = (-1, -1, ..., -1)$.

Our special algorithm will is derived from our general solution, but using a, b and c as constants -1, 2 and -1. If we do our forward substitution we'll see some simplifications:

$$\tilde{b_1} = b_1 = 2$$
$$\tilde{g_1} = g_1$$

$$\tilde{b_i} = b_i - \frac{a_i}{\tilde{b}_{i-1}} c_{i-1} = 2 - \frac{1}{\tilde{b}_{i-1}}$$
$$\tilde{g_i} = g_i - \frac{a_i}{\tilde{b}_{i-1}} gi - 1 = g_i + \frac{\tilde{g}_{i-1}}{\tilde{b}_{i-1}} \qquad i = 2, 3, ..., n$$

Let's make the same simplifications for our backward substitution, then we get the following expression:

$$v_i = \frac{\tilde{g}_i - c_i v_{i+1}}{\tilde{b}_i} = \frac{\tilde{g}_i + v_{i+1}}{\tilde{b}_i} \qquad\qquad i = n-1, n-2, ..., 1$$

With these simplifications, we have made the new special algorithm for solving our equation.

b)

Since we are working with constants in this case we can substitute a, b and c for a constant to cut down on our operations. For the forward equation we count 4 FLOPs for $n-1$ iterations. This gives: $4(n-1) = 4n-4$.
For the bacward equation we get 2 FLOPs per $n-1$ iterations plus one FLOP for finding v_n which gives us $2(n-1)+1 = 2n-1$. We add these two together and we now have: $6n-5$ FLOPs.

c)

See oppgave910.cpp in Git Hub repository under Sources.

Problem 10

See oppgave910.cpp in Git Hub repository under Sources. These were the times we got from running the codes 3 different times with $n = 10^6$:

n	General algorithm	Specialized algorithm
10^6	0,14366 sec	0,17430 sec
10^6	0,12037 sec	0,15504 sec
10^6	0,15507 sec	0,13874 sec

The codes use about the same amount of time to calculate, but it seems the general algorithm used a shorter amount of time in general. We expected the specialized algorithm to be faster, and after a huge number of tries it might be generally faster. There may be other reasons for this since the programs are different from each other.

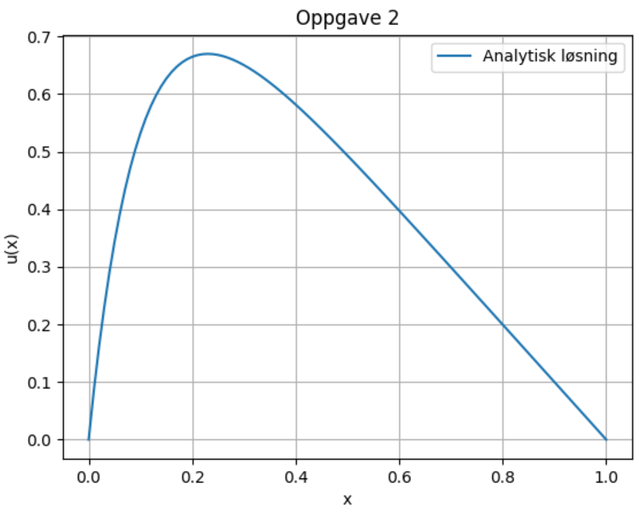

Figure 1: Analytical solution of u(x).

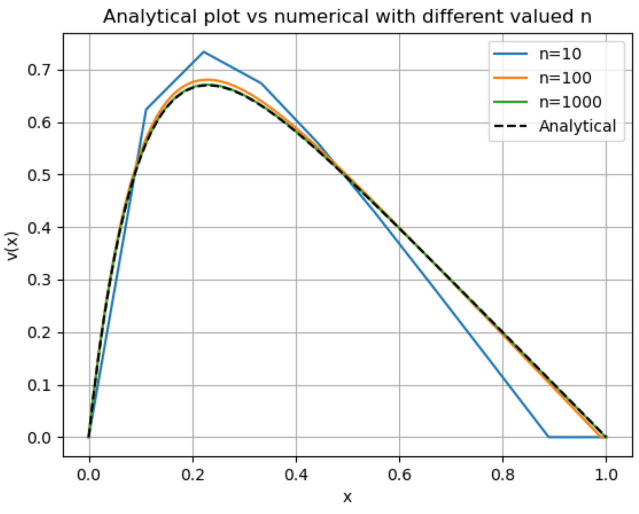

Figure 2: Analytical solution of u(x) plotted against our numerical approximations for n=10, 100 and 1000.

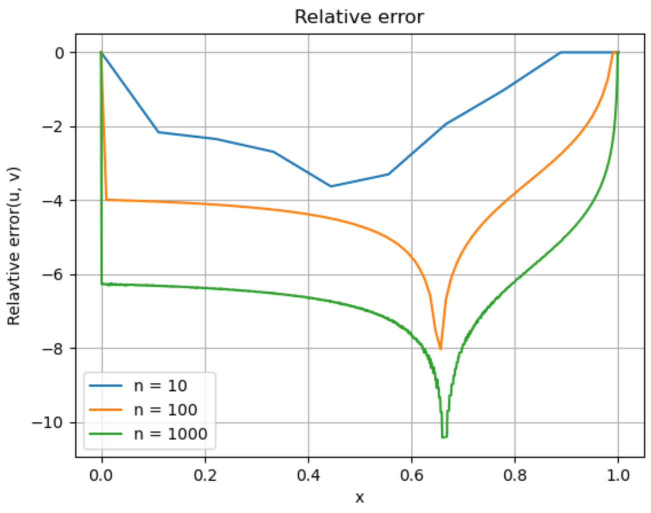

Figure 3: Plot of the relative error for n=10, 100 and 1000.

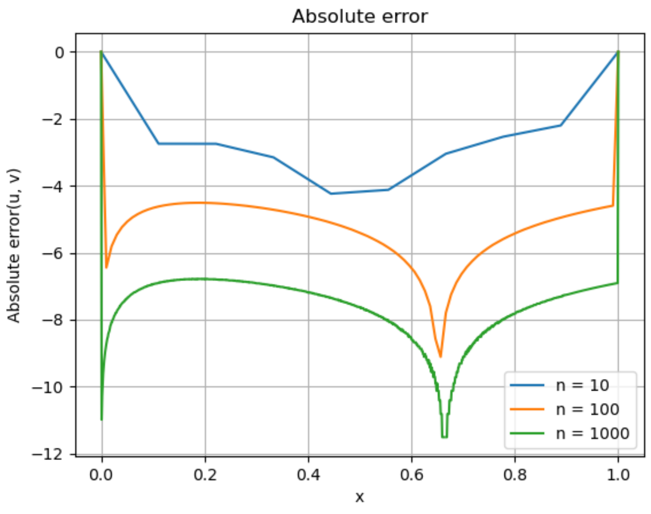

Figure 4: Plot of absolute error for n=10, 100 and 1000.