FYS3150 Project 1

Sylvien Wang, Sebastian Ranum Tenmann and Cecilie Klarpås
(Dated: September 13, 2021)

https://github.com/cecilmkl/fys3150/tree/main/prosjekt1
Objective: Solving the one-dimensional Poisson equation numerically.

$$-\frac{\mathrm{d}^2u}{\mathrm{d}x^2} = f(x) = 100e^{-10x} \tag{1}$$

PROBLEM 1

$$\begin{aligned}
u(x) &= \iint -100e^{-10x}\,dx\,dx \\
&= -100\int [\int e^{-10x}dx]\,dx \\
&= -100\int (-\frac{1}{10}e^{-10x} + c_1)\,dx \\
&= 10\int e^{-10x} + c_1\,dx \\
&= 10(-\frac{1}{10}e^{-10x} + c_1x + c_2) \\
&= -e^{-10x} + c_1x + c_2
\end{aligned}$$

We use the border values to find the unknown constants:

$$\begin{aligned}
u(0) &= 0 \\
-e^{-10*0} + c_1 * 0 + c_2 &= 0 \\
-1 + c_2 &= 0 \\
c_2 &= 1
\end{aligned}$$

$$\begin{aligned}
u(1) &= 0 \\
-e^{-10*1} + c_1 * 1 + 1 &= 0 \\
-e^{-10} + c_1 + 1 &= 0 \\
c_1 &= e^{-10} - 1
\end{aligned}$$

Which gives the analytical solution of the Poisson equation:

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

PROBLEM 2

For this problem we created a c++ code, linked to in the list of GitHub repository, evaluating the exact solutions for u(x) given various x-values using Equation 2. Afterwards we used python to make a plot of the data output produced by this code, shown in Figure 1.

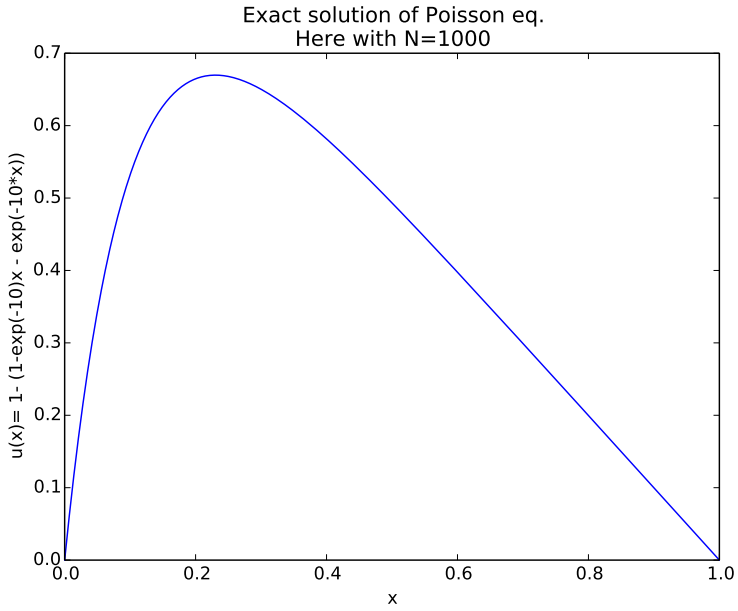

FIG. 1. This plot shows the exact solutions of u(x) for various x-values $\in [0, 1]$ using Equation 2 in Problem 1. This plot has m=N+1 = 1001 x-values, including the two boundary points (x=0, x=1).

PROBLEM 3

We start off with the Poisson equation (Equation 1) and discretize the left hand side using the double derivative:

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

where h is the step-size. Which in the case of this Poisson equation results in:

$$-\left[\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + O(h^2)\right] = f(x_i)$$

Then we approximate by ignoring the higher order terms, and change the notation of u to v. By rearranging this equation and replacing $f(x_i)$ with $100e^{-10x_i}$, as given in Equation 1, we get:

$$-v_{i-1} + 2v_i - v_{i+1} = h^2 100e^{-10x_i} \tag{3}$$

PROBLEM 4

We write Equation 3 in a table for each iteration between the boundary points:

$$\begin{array}{lccccccc}
(\mathrm{i}=1) & -v_0 & +2v_1 & -v_2 & & & = & h^2100e^{-10x_1} + v_0 \\
(\mathrm{i}=2) & & -v_1 & +2v_2 & -v_3 & & = & h^2100e^{-10x_2} \\
(\mathrm{i}=\ldots) & & & \ddots & \ddots & \ddots & = & \vdots \\
(\mathrm{i}=\text{n-1}) & & & & -v_{n-2} & +2v_{n-1} \quad -v_n & = & h^2100e^{-10x_{n-1}} \\
(\mathrm{i}=\mathrm{n}) & & & & & -v_{n-1} \quad +2v_n \quad -v_{n+1} & = & h^2100e^{-10x_n} + v_{n+1}
\end{array}$$

where we move the known boundary points v_0 and v_{n+1} to the RHS.
This can then be expressed as a matrix equation, separating the LHS into the prefixes and v-values, and on the RHS

substituting the values above with g_i:

$$\begin{bmatrix} 2 & -1 & 0 & 0 & \cdots \\ -1 & 2 & -1 & 0 & \cdots \\ \ddots & \ddots & \ddots & \ddots & \cdots \\ \cdots & 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{bmatrix} \tag{4}$$

From this we can see that the discretized Poisson equation can be written in the matrix form:

$$\mathbf{A}\vec{v} = \vec{g} \tag{5}$$

PROBLEM 5

Problem a

The complete solution of the Poisson equation will include the boundary points $u(0) = 0$ and $u(1) = 0$, such that $v^*(0) = u(0)$ and $v^*(m) = u(1)$ while the remaining $m-2$ values v_j^* for $j = 1, ..., m-1$ are unknown. These are found by solving the matrix equation $A\vec{v} = \vec{g}$ for $A \in \mathbb{R}^{nxn}$, $\vec{v} \in \mathbb{R}^n$ and $\vec{g} \in \mathbb{R}^n$ where n is the number of unknowns. Therefore, the number of unknowns must be the same and $n = m - 2$.

b

As mentioned above, the solution of the matrix equation $A\vec{v} = \vec{g}$ is the vector $\vec{v}$ which corresponds to the $m-2$ unknown values v_j^* for $j = 1, ..., m-1$ of the complete solution of the Poisson equation, $\vec{v}^*$. So,

$$\vec{v}^* = \begin{bmatrix} v_0^* & v_0 & \dots & v_n & v_m^* \end{bmatrix}$$

PROBLEM 6

Problem a

The algorithm is found using the same procedure as for Problem 4 but the sub-, main- and superdiagonal elements of matrix A is now going to be expressed as variables a_i, b_i and c_i for $i = 0, ..., n$ in vectors $\vec{a}, \vec{b}$ and $\vec{c}$ instead of by $-1, 2$ and -1 respectively. Note that due to vector indices beginning at 0 in C++, the indices of each vector will go from $0, ..., n-1$ instead of $1, ..., n-1$.

Algorithm 1 General algorithm

Want to solve matrix equation $A\vec{v} = \vec{g}$ for $\vec{v}$

Input: Vector $\vec{g}$, integer number of unknowns n and vectors $\vec{a}, \vec{b}$ and $\vec{c}$ representing the sub-, main- and superdiagonal of $A \in \mathbb{R}^{nxn}$. All vectors are of size $\mathbb{R}^n$. To ensure that indices represent row in A, the diagonal vectors will look like this:

$\vec{a} = [0, a_1, ..., a_{n-1}]$
$\vec{b} = [b_0, b_1, ..., b_{n-1}]$
$\vec{c} = [c_0, ..., c_{n-2},\ 0]$
$\tilde{b}_0 = b_0$; ▷ New notation where $\vec{b}$ and $\vec{g}$ are $\in \mathbb{R}^n$
$\tilde{g}_0 = g_0$;
for $i = 1, ..., n-1$ **do** ▷ Repeating n-1 times
  $\text{tmp} = a_i / \tilde{b}_{i-1}$; ▷ 1 FLOP
  $\tilde{b}_i = b_i - \text{tmp} * c_{i-1}$; ▷ 2 FLOPs
  $\tilde{g}_i = g_i - \text{tmp} * \tilde{g}_{i-1}$; ▷ 2 FLOPs
$v_{n-1} = \tilde{g}_{n-1} / \tilde{b}_{n-1}$ ▷ 1 FLOP
for $j = n-2, ..., 0$ **do** ▷ Repeating n-1 times
  $v_j = (\tilde{g}_j - c_j * v_{j+1}) / \tilde{b}_j$; ▷ 3 FLOPs
return $\vec{v}$

Problem b

The total number of FLOPs in the general algorithm is:

$$(1+2+2)(n-1)+3(n-1)+1 = 5n-5+3n-3+1 = 8n-7 \approx 8n \text{ FLOPs}$$

PROBLEM 7

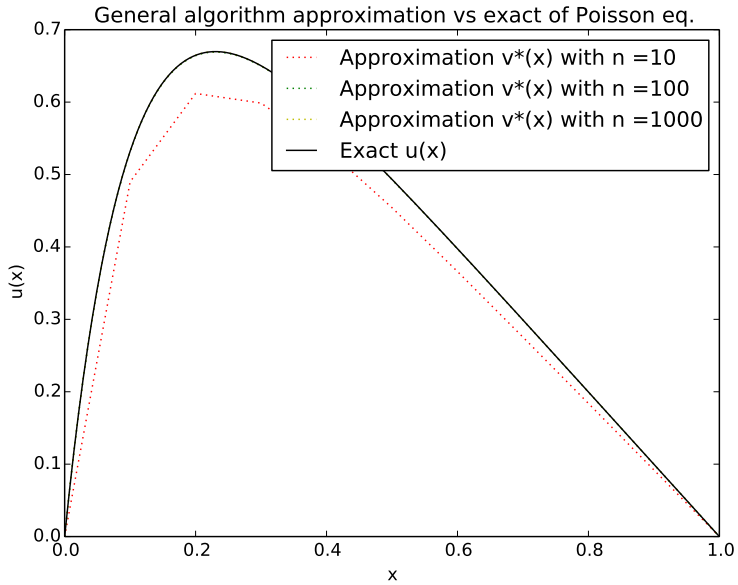

FIG. 2. Functions for the general algorithm for solving u(x), with stepsizes N= 10, 100, 1000, and also the exact u(x)

Figure 2 shows the solutions of u(x) using the general algorithm, compared to the exact solution. This shows that the general algorithm quickly closes up to the exact solution, with higher powers of n.

PROBLEM 8

Problem a

The top part of Figure 3 and Figure 4 shows the graphs for absolute error. Looking at the graph one can see that increasing N decrease the error overall. This changes when N¿=10^6 The reason for the graphs cutting of before the boundary points for the smaller N lines, is because of the step-size, as for N=10 it will start 1/10th of the way into the function.

Problem b

The second part of Figure 3 and Figure 4 shows the relative errors plots. Again we can see the trend of errors decreasing with higher N until N¿=10^6 as for the absolute error..

Problem c

Table I shows the largest values for the relative error, showing the same trend of decreasing error up until N=10^6 where it starts to increase. This relates to the cut-of point for truncation errors sinking and round-off errors rising.

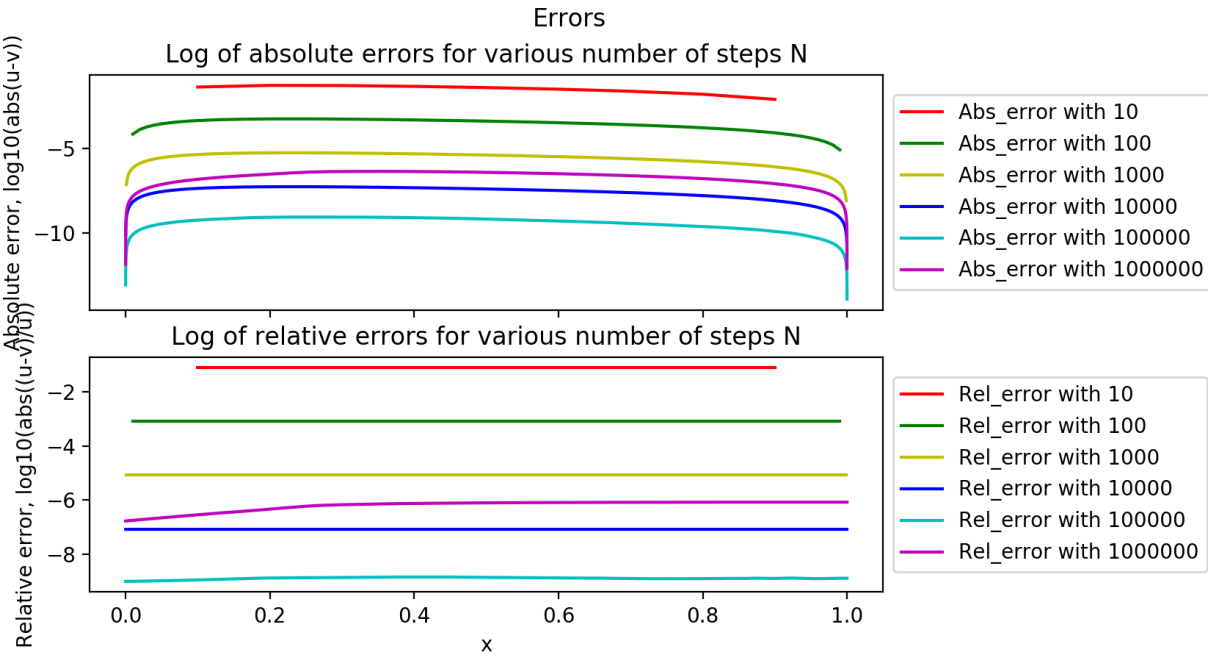

FIG. 3. Plots showing the absolute- and relative errors for N=10 - 10^6 over x

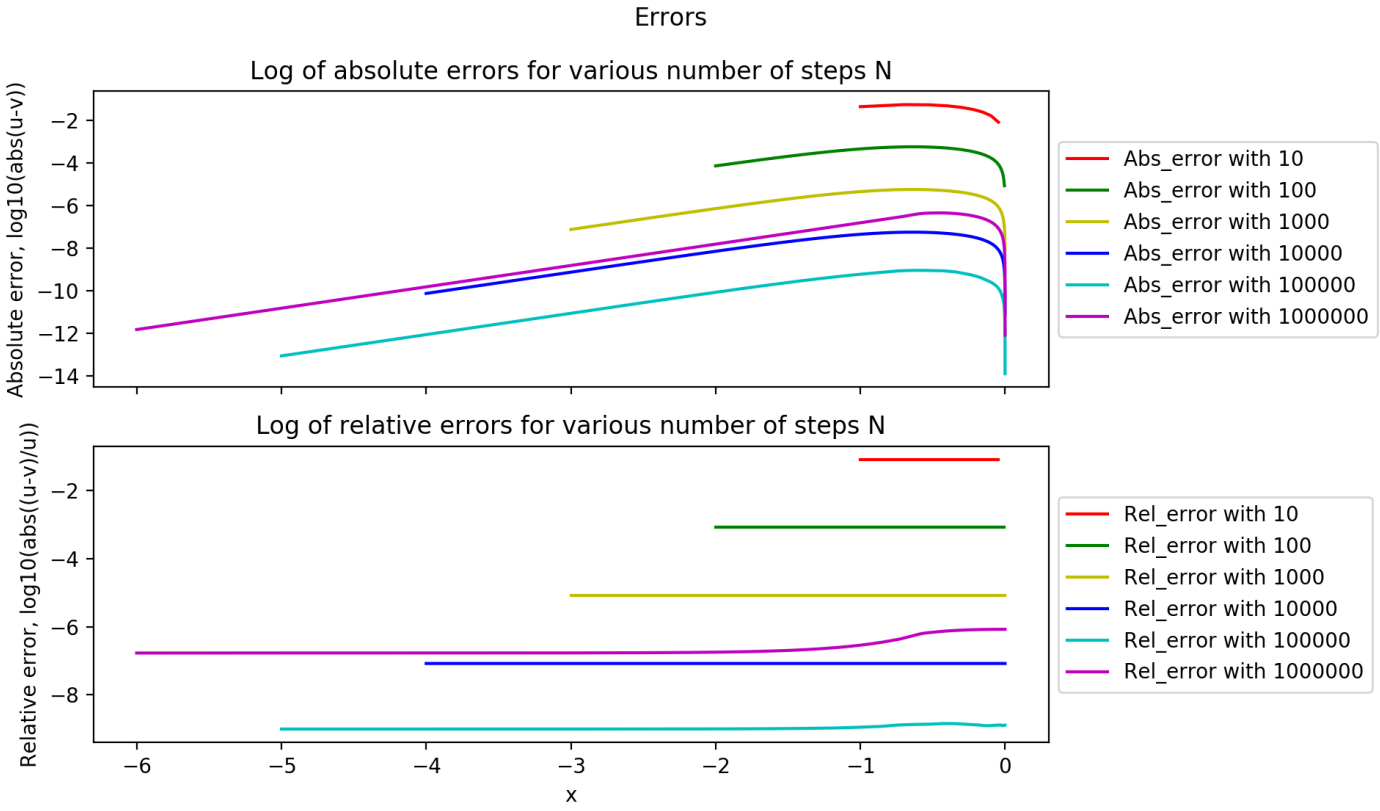

FIG. 4. Plots showing the absolute- and relative errors for N=10 - 10^6 over log_{10}(x)

N	$\max(\epsilon_i)$
$N = 10$	$7.9326405792 * 10^{-4}$
$N = 10^2$	$8.3291683197 * 10^{-4}$
$N = 10^3$	$8.3332920165 * 10^{-6}$
$N = 10^4$	$8.3330064213 * 10^{-8}$
$N = 10^5$	$1.4355972481 * 10^{-9}$
$N = 10^6$	$8.4041230864 * 10^{-7}$
$N = 10^7$	$2.9838006618 * 10^{-6}$

TABLE I. Showing the maximum relative error for each choice of N

PROBLEM 9

The algorithm with the special case of traditional matrix **A** of the signature $(-1, 2, -1)$:

Problem a

Algorithm 2 Special algorithm

Want to solve matrix equation $A\vec{v} = \vec{g}$ for $\vec{v}$
Input: Vector $\vec{g}$ and integer number of unknowns n
$\tilde{b}_0 = 2$; ▷ New notation where $\vec{b}$ and $\vec{g}$ are $\in \mathbb{R}^n$
$\tilde{g}_0 = g_0$;
for $i = 1, ..., n-1$ **do** ▷ Repeating n-1 times
  tmp $= 1/\tilde{b}_{i-1}$; ▷ 1 FLOP
  $\tilde{b}_i = 2-$tmp; ▷ 1 FLOPs
  $\tilde{g}_i = g_i +$tmp $* \tilde{g}_{i-1}$; ▷ 2 FLOPs
$v_{n-1} = \tilde{g}_{n-1}/\tilde{b}_{n-1}$ ▷ 1 FLOP
for $j = n-2, ..., 0$ **do** ▷ Repeating n-1 times
  $v_j = (\tilde{g}_j + v_{j+1})/\tilde{b}_j$; ▷ 2 FLOPs
return $\vec{v}$

Problem b

The total number of FLOPs in the special algorithm is:

$$(2 + 1 + 1)(n - 1) + 2(n - 1) = 4n - 4 + 2n - 2 = 6n - 6 \approx 6n \text{ FLOPs}$$

which is less then the 8n for the general algorithm found in Problem 6b.

PROBLEM 10

The timing differences of the general and special algorithm for different choices of number of steps, N, are given in Table II

As N increases it is clear to see that the significance of the executing times of the general and special algorithm increase as well.

PROBLEM 11

If we were to solve the matrix equation using general LU decomposition instead of Gaussian elimination as we've done in this project, then it would have taken significantly more time and memory to run the code. This is due to the decomposition of matrix **A** into lower and upper matrices **L** and **U** with complexity $O(n^3)$, and then solving the equation with complexity $O(n^2)$. We omitted the decomposition step by using algorithms based on Gaussian elimination fairly cheaply, with complexity $O(n)$ based on out calculations of FLOPs.

Timing results (in seconds)											
N=10		N=10^2		N=10^3		N=10^4		N=10^5		N=10^6	
General	Special	General	Special	General	Special	General	Special	General	Special	General	Special
$3.0 * 10^{-6}$	$3.0 * 10^{-6}$	$2.0 * 10^{-5}$	$1.2 * 10^{-5}$	$1.1 * 10^{-4}$	$1.4 * 10^{-4}$	$3.2 * 10^{-3}$	$8.5 * 10^{-3}$	$1.5 * 10^{-2}$	$7.2 * 10^{-3}$	$1.2 * 10^{-1}$	$8.3 * 10^{-2}$
$1.4 * 10^{-5}$	$6.0 * 10^{-6}$	$1.4 * 10^{-5}$	$1.0 * 10^{-5}$	$1.1 * 10^{-4}$	$1.2 * 10^{-4}$	$1.1 * 10^{-3}$	$1.1 * 10^{-3}$	$1.1 * 10^{-2}$	$8.0 * 10^{-3}$	$1.4 * 10^{-1}$	$9.0 * 10^{-2}$
$7.0 * 10^{-6}$	$3.0 * 10^{-6}$	$1.6 * 10^{-5}$	$8.0 * 10^{-6}$	$1.1 * 10^{-4}$	$1.5 * 10^{-4}$	$1.1 * 10^{-3}$	$7.2 * 10^{-3}$	$1.1 * 10^{-2}$	$1.2 * 10^{-2}$	$1.1 * 10^{-1}$	$8.8 * 10^{-2}$
$5.0 * 10^{-6}$	$3.0 * 10^{-6}$	$1.9 * 10^{-5}$	$2.2 * 10^{-5}$	$1.2 * 10^{-4}$	$9.2 * 10^{-5}$	$1.1 * 10^{-3}$	$1.1 * 10^{-3}$	$1.5 * 10^{-2}$	$7.3 * 10^{-3}$	$1.1 * 10^{-1}$	$9.1 * 10^{-2}$
$1.1 * 10^{-5}$	$4.0 * 10^{-6}$	$1.6 * 10^{-5}$	$3.2 * 10^{-5}$	$1.1 * 10^{-4}$	$7.8 * 10^{-5}$	$1.5 * 10^{-3}$	$1.1 * 10^{-3}$	$1.3 * 10^{-2}$	$8.1 * 10^{-3}$	$1.3 * 10^{-1}$	$7.2 * 10^{-2}$
$3.0 * 10^{-6}$	$3.0 * 10^{-6}$	$2.8 * 10^{-5}$	$2.8 * 10^{-5}$	$1.1 * 10^{-4}$	$4.0 * 10^{-4}$	$1.1 * 10^{-3}$	$7.3 * 10^{-4}$	$1.5 * 10^{-2}$	$7.1 * 10^{-3}$	$1.1 * 10^{-1}$	$7.0 * 10^{-2}$

TABLE II. Result om timing tests for general and special algorithm, with number of points, N, ranging between 10 and 10^6

LU decomposition seem to be more useful when we have multiple equations with the same matrix $\mathbf{A}$. Our $\mathbf{A}$ represent taking the negative of the second derivative, so it could be used for equations on the same form as 1 but with different source terms. Therefore, since our $f(x)$ is constant, we conclude that our algorithms were more efficient than it would have been solving with LU decomposition.