Project 1

Øyvind Bø Kaasa
(Dated: September 10, 2025)

https://github.com/yvn-1/REPO/tree/fe2d837e6027ef012a142b2767dd684870ae1cb8/Skole/FYS4150/Project1

PROBLEM 1

Inserting expression (2) into equation (1) from the problem text:

$$-\frac{d^2u}{dx^2} = -\frac{d^2\left(1-(1-e^{-10})x - e^{-10x}\right)}{dx^2} = 0 + 0 + 100e^{-10x} = 100e^{-10x} = f(x)$$

We also check the boundary conditions:

$$u(0) = 1 - (1-e^{-10})\cdot 0 - e^{-10\cdot 0} = 1 - 0 - 1 = 0$$

$$u(1) = 1 - (1-e^{-10})\cdot 1 - e^{-10\cdot 1} = 1 - 1 + e^{-10} - e^{-10} = 0$$

Both the insertion and boundary conditions checks out, meaning $u(x)$ is a solution to equation (1).

PROBLEM 2

See Figure 1 for the plot, and for the C++ and Python code see the files named problem2.py and problem2.cpp on the above GitHub.

PROBLEM 3

We will use the discretization given by $x_{i+1} = x_i + h$. Expressing our unknown function $u(x)$ as a Taylor series and using the discretization we have:

$$u(x_{i+1}) = u(x_i + h) = \sum_{n=0}^{\infty} \frac{1}{n!} \cdot u^{(n)}(x_i) \cdot h^n = u(x_i) + u'(x_i)\cdot h + \frac{1}{2}\cdot u''(x_i)\cdot h^2 + \mathcal{O}(h^3) \tag{1}$$

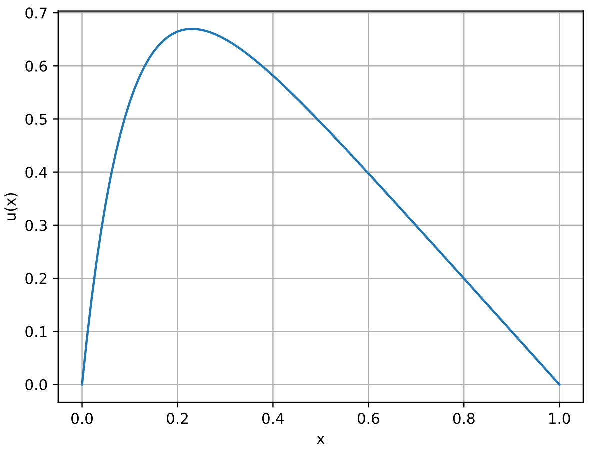

FIG. 1. Plot of x-values against u(x) for $x \in [0, 1]$

Doing the same for $u(x_{i-1})$ gives:

$$u(x_{i-1}) = u(x_i - h) = u(x_i) - u'(x_i) \cdot h + \frac{1}{2} \cdot u''(x_i) \cdot h^2 + \mathcal{O}(h^3) \tag{2}$$

Adding (1) and (2) gives:

$$u(x_{i+1}) + u(x_{i-1}) = 2 \cdot u(x_i) + u''(x_i) \cdot h^2 + \mathcal{O}(h^4) \tag{3}$$

$$u''(x_i) = \frac{u(x_{i+1}) - 2 \cdot u(x_i) + u(x_{i-1})}{h^2} + \mathcal{O}(h^2) \tag{4}$$

In (3) the $\mathcal{O}(h^3)$ cancel due to oddness, and in (4) we divide by h^2, giving the final error at the order of $\mathcal{O}(h^2)$.

Expression (4) is the known 3-point formula, with truncation error on the order of $\mathcal{O}(h^2)$. Using the discretization $v_i = u(x_i)$, we can express the discretized 1D Poisson equation as:

$$-\frac{v_{i+1} - 2 \cdot v_i + v_{i-1}}{h^2} = f(x_i) \tag{5}$$

with boundary conditions giving $v_0 = 0$ and $v_N = 0$.

PROBLEM 4

Rewriting (5) from problem 3, we have:

$$-1 \cdot v_{i-1} + 2 \cdot v_i - 1 \cdot v_{i+1} = f(x_i) \cdot h^2 \tag{6}$$

We observe that for each i this can be written as:

$$\begin{bmatrix} -1 & 2 & -1 \end{bmatrix} \begin{bmatrix} v_{i-1} \\ v_i \\ v_{i+1} \end{bmatrix} = f(x_i) \cdot h^2 \tag{7}$$

From this we can construct a matrix from all the row vectors, and a column vector for each and all of the v_i

$$\begin{bmatrix} 2 & -1 & 0 & \cdots & \cdots & \cdots & 0 \\ -1 & 2 & -1 & \ddots & & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & \ddots & -1 & 2 & -1 \\ 0 & \cdots & \cdots & \cdots & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \cdots \\ v_{N-1} \end{bmatrix} = \begin{bmatrix} f(x_1)h^2 + v_0 \\ f(x_2)h^2 \\ f(x_3)h^2 \\ \cdots \\ f(x_{N-1})h^2 + v_N \end{bmatrix} \tag{8}$$

where we have gathered the unknown v_i on the left hand side, and included the known v_0 and v_N with the known values of $f(x_i)$ on the right hand side.

PROBLEM 5

a) The complete solution will include the known solutions, meaning the given boundary conditions. Of these we are given two, meaning the relationship is given by

$$m - n = 2 \tag{9}$$

b) We find the interior solutions, the solutions between our boundaries.

PROBLEM 6

a) We can solve the equation in a two step process. We will look at an equation with a general NxN tri-diagonal matrix given as

$$\begin{bmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & \cdots & \cdots & a_N & b_N \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_N \end{bmatrix} \tag{10}$$

First we make the matrix upper diagonal. We can eliminate the a_i's by the process $R_{i+1} \to R_{i+1} - \frac{a_{i+1}}{b_i} R_i$, where R_i are the row vectors of the tri-diagonal matrix. Then we can update our b_i's and g_i's.

Algorithm 1 Make tri-diagonal matrix upper diagonal

```
$\tilde{b}_1 = b_1$ and $\tilde{g}_1 = g_1$.
  for $i = 1, ..., N-1$ do
    $R_{i+1} \to R_{i+1} - \frac{a_{i+1}}{\tilde{b}_i} R_i$
    $m_{i+1} = \frac{a_{i+1}}{\tilde{b}_i}$
    $\tilde{b}_{i+1} = b_{i+1} - m_{i+1} c_i$
    $\tilde{g}_{i+1} = g_{i+1} - m_{i+1} \tilde{g}_i$
```

Now if we multiply our last row by $1/\tilde{b}_N$, then we can read out $v_N = \frac{\tilde{g}_N}{\tilde{b}_N}$. Now we can work ourselves back to the top to complete the Gaussian elimination and get our solutions.

Algorithm 2 Do back substitution to get expression for the v_i

```
  $v_N = \frac{\tilde{g}_N}{\tilde{b}_N}$
  for $i = N, ..., 2$ do
    $v_{i-1} = \frac{\tilde{g}_{i-1} - c_{i-1} v_i}{\tilde{b}_{i-1}}$
```

Thats our algorithm to solve the general problem.

b) We see that in Algorithm 1 that we have one FLOP for each m_i, meaning $(N-1)$ FLOPs. The $\tilde{b}_i$ needs $2(N-1)$ FLOPs, same as $\tilde{g}_{i+1}$. The first part of the algorithm takes $5N - 5$ flops.

The other half, Algorithm 2, takes 3 FLOPs each v_i, we have $N-1$ of these, and then the v_N calculation needs 1 FLOP. Algorithm 2 needs $3(N-1) + 1 = 3N - 2$ FLOPs.

Put together the algorithm requires $8N - 7$ FLOPs.

PROBLEM 7

a) The solution is the files named problem7.cpp, problem7.txt

b) See 2 for plot. We see that as the number of steps increase, so does the correspondence with the analytical solution. For code, see problem7b.cpp.

PROBLEM 8

a) See files named problem8a.py and problem8a.cpp, and 3.

b) See file named problem8b.py and problem8b.cpp, and 4.

c) Table of maximum relative error for each number of steps can be found in I. Observe that maximum error rises at 10 000 000 steps. Code is available as problem8c.cpp

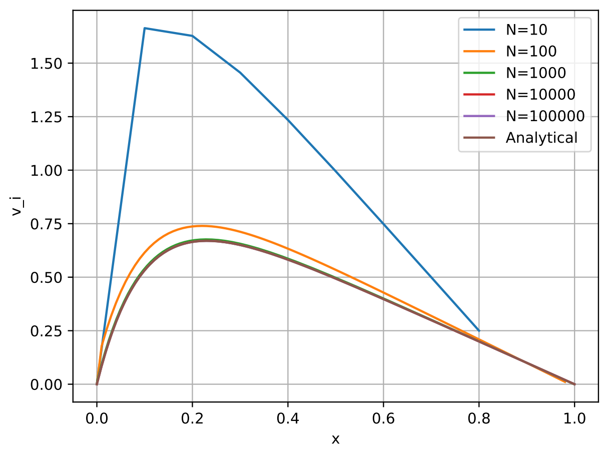

FIG. 2. Comparison between analytical solution and numerical solutions using N=10,100,1000,10000,100000. We observe that from and including N=1000, the numerical solutions are close to the analytical.

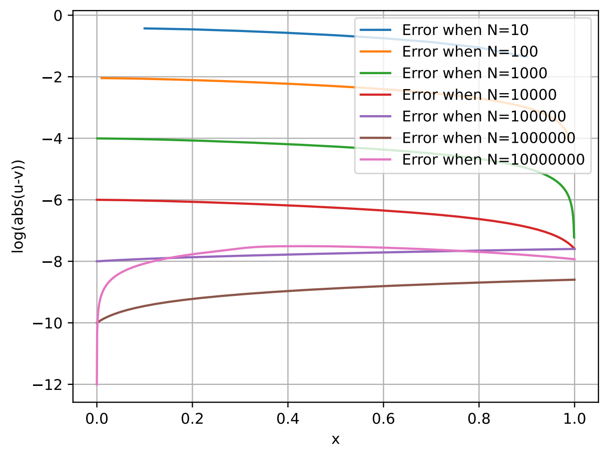

FIG. 3. Log of absolute error, observe the erratic behaviour near the boundary. This might be due to a very low error in that region

PROBLEM 9

a) In our expressions where a and c occur we swap the signs and remove the variable. b) Removing a and c and swapping signs saves us for one operations in each iteration of the two for loops in 1 and 2. This removes 2N FLOPs, meaning our algorithm runs at 6n-7 FLOPs. c) See problem9c.cpp on the GitHub for the code.

Number of steps	Maximum relative error $max(\epsilon_i)$
N=10	$7.015 \cdot 10^{-1}$
N=100	$1.06 \cdot 10^{-1}$
N=1000	$1.106 \cdot 10^{-2}$
N=10000	$1.111 \cdot 10^{-3}$
N=100000	$2.520 \cdot 10^{-3}$
N=1000000	$2.518 \cdot 10^{-3}$
N=10000000	$1.323 \cdot 10^{-1}$

TABLE I. Max relative error for each number of steps.

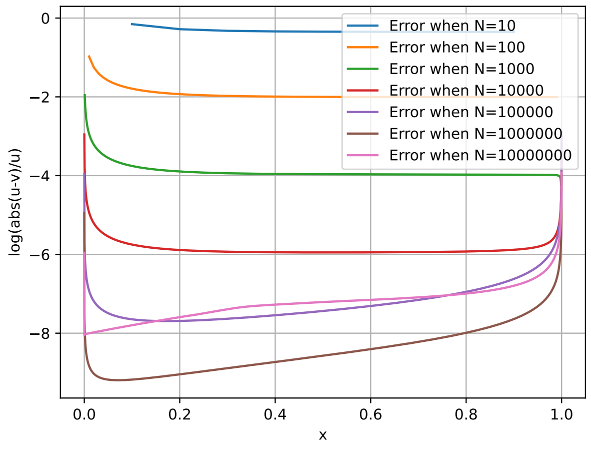

FIG. 4. Log of the absolute of the relative error. We still see the sudden drop then rise in error closer to x=1

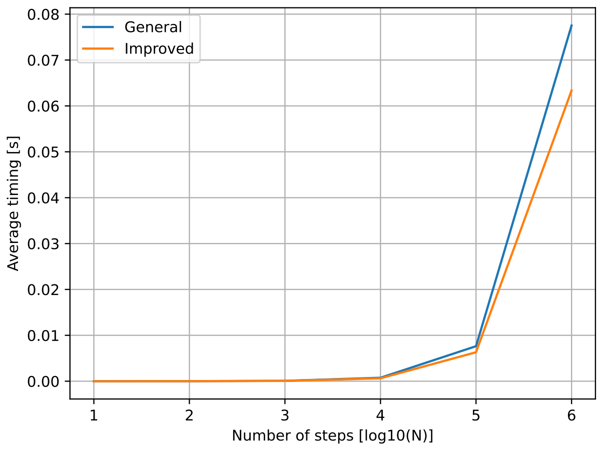

FIG. 5. Average timings of the general and improved algorithms. Each number of steps are ran ten times and averaged. Here its used log10 of N

PROBLEM 10

Code can be found on the GitHub under problem10.py, problem10general.cpp and problem10improved.cpp. As expected, the improved algorithm is faster, and its about 18 percentage faster if we take the ratio. See 6 for timings where N is on logarithmic scale, and 5 for linear. We see an increasing payoff for our improved algorithm.

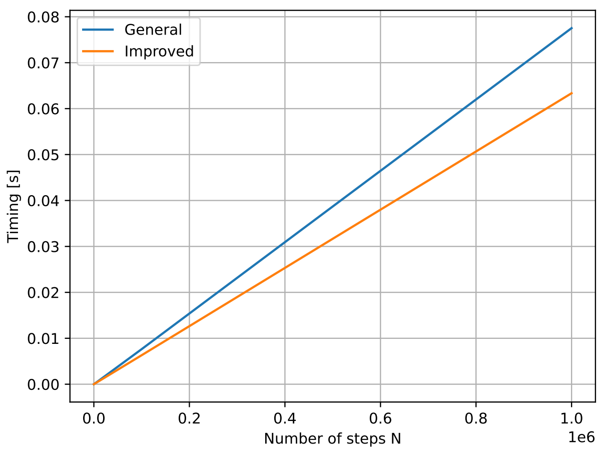

FIG. 6. Comparison of the general and improved algorithm in linear N.