Project 1

Olav K. Arnesen
(Dated: September 12, 2022)

https://github.com/olavkar/fys3150/tree/main/project1

PROBLEM 1

To check whether $u(x) = 1 - (1 - e^{-10})x - e^{-100x}$ is a solution or not we insert it into the Poisson equation:

$$\begin{aligned} f(x) =& -\frac{d^2 u(x)}{dx^2} \\ =& -\frac{d^2}{dx^2}\left[1 - \left(1 - e^{-10}\right)x - e^{-10x}\right] \\ =& -\frac{d}{dx}\left[1 - e^{-10} + 10e^{-10x}\right] \\ =& 100e^{-10x} \end{aligned} \tag{1}$$

which is the expected solution.

PROBLEM 2

Using $n = 1000$ steps between $x = 0$ and $x = 1$ we got Fig. 1.

PROBLEM 3

The second derivative of $u(x_i) = u_i$, where $i = 1, 2 \ldots n$, is given numerically by

$$u_i'' = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + O(h^2) \tag{2}$$

where $h = \frac{x_n - x_1}{n}$ and $O(h^2)$ are all terms of order h^2 or higher. Inserting the Poisson equation and removing higher order terms we get the equation

$$-v_{i+1} + 2v_i - v_{i-1} = h^2 f_i \tag{3}$$

where $v_i \approx u_i$ and $f(x_i) = f_i$. We can rename $h^2 f_i$ to g_i to get a discretized Poission equation

$$-v_{i+1} + 2v_i - v_{i-1} = g_i \tag{4}$$

PROBLEM 4

We can write out the discretized Poisson equation for all i:

$$\begin{aligned} i = 1 :&\ 2v_1 - v_2 \ + 0 \quad + 0 \ \cdots + 0 \qquad + 0 \quad = g_1 \\ i = 2 :&-v_1 + 2v_2 - v_3 \ + 0 \ \cdots + 0 \qquad + 0 \quad = g_2 \\ i = 3 :&\ \ 0 \ - v_2 \ + 2v_3 - v_3 \cdots + 0 \qquad + 0 \quad = g_3 \\ &\vdots \\ i = n :&\ \ 0 \ + 0 \quad + 0 \quad + 0 \ \cdots - v_{n-1} + 2v_n = g_n \end{aligned}$$

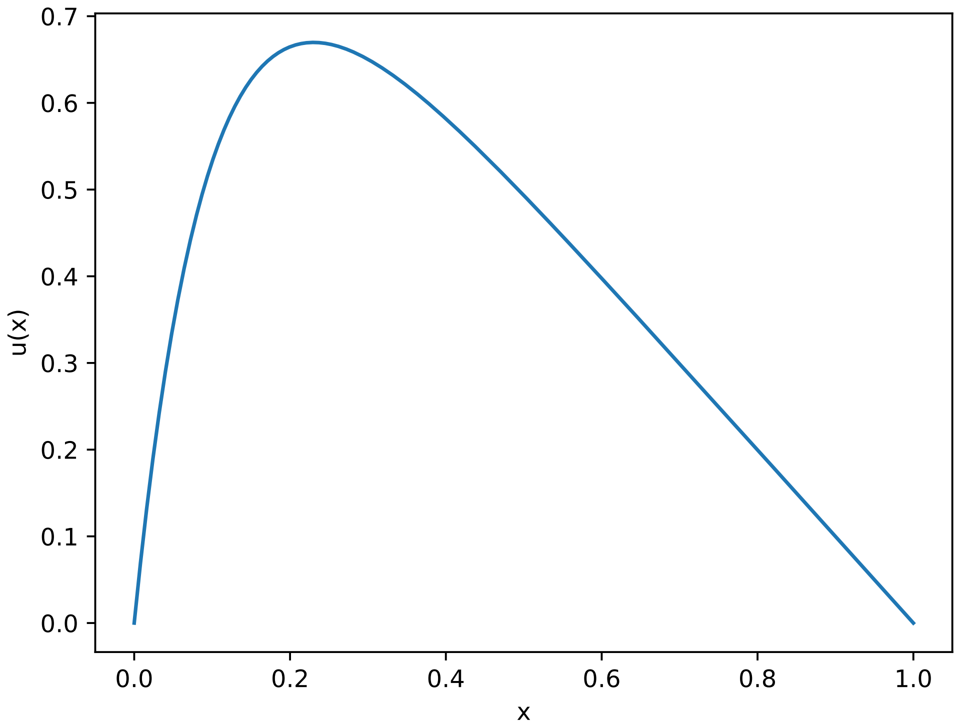

FIG. 1: The function $u(x)$ from $x = 0$ to $x = 1$.

If we define two vectors $\vec{v} = (v_0, v_1, \dots v_n)$ and $\vec{g} = (g_0, g_1, \dots g_n)$ we can clearly see these equations can be represented by

$$\begin{pmatrix} 2 & -1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 \\ & & \vdots & & & \\ 0 & 0 & 0 & 0 & \dots & 2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ \vdots \\ g_n \end{pmatrix} \tag{5}$$

where the left-hand matrix is a tri-diagonal $n \times n$-matrix, where all elements in the main diagonal is 2 and the elements in the sub- and superdiagonal are -1. If we call the matrix $\mathbf{A}$ we can write the vector equation as

$$\mathbf{A}\vec{v} = \vec{g} \tag{6}$$

PROBLEM 5

a)

$\vec{v}^*$ being a complete solution means it includes the endpoints $x = 0$ and $x = 1$, while $\vec{v}$ does not (though that was not obvious to me without help from a teacher). This means $m = n + 2$.

b)

When we solve for $\vec{v}$ we find the middle part of $\vec{v}^*$, i.e. not the endpoints. We could write $\vec{v}^* = (0, v_1, v_2, \dots, v_n, 0)$.

PROBLEM 6

a)

Algorithm derived in lecture on 2.09.2022.

Algorithm 1 Gaussian elimination

Subdiagonal: $\vec{a} = (a_2, a_3, \ldots, a_n)$
Main diagonal: $\vec{b} = (b_1, b_2, \ldots, b_n)$
Superdiagonal: $\vec{c} = (c_1, c_2, \ldots, c_{n-1})$

$\tilde{g_1} = g_1$ ▷ Forward substitution
$\tilde{b_1} = b_1$
for $i = 2, \ldots, n$ **do**
  $\tilde{g_i} = g_i - \frac{a_i}{b_{i-1}} g_{i-1}$
  $\tilde{b_i} = b_i - \frac{a_i}{b_{i-1}} c_{i-1}$

$v_n = \frac{\tilde{g_n}}{\tilde{b_n}}$ ▷ Back substitution
for $i = n-1, \ldots, 1$ **do**
  $v_i = \frac{\tilde{g_i} - c_i v_{i+1}}{\tilde{b_i}}$

b)

The number of FLOPs from forward substitution is $(n-1) \cdot 2 \cdot 3$, (3 in $\tilde{g_i}$, 3 in $\tilde{b_i}$, $n-1$ times). From back substitution there are $(n-1) \cdot 3 + 1$, (3 in v_i, $(n-1)$ times, plus v_n). The total number of FLOPs is $(n-1) \cdot 2 \cdot 3 + (n-1) \cdot 3 + 1 = 9n - 9 \approx 9n$ for large n.

PROBLEM 7

a)

Using the algorithm for $n = 1000$ non-endpoints and plotting the results gives us Fig. 2.

b)

For $n_{\text{steps}} \in \{10, 100, 1000\}$ we get Fig. 3. Additional figures using higher n_{steps} would be indistinguishable from the current ones.

PROBLEM 8

a) & b)

Fig. 4 shows the absolute and relative errors for $n_{\text{steps}} \in \{10, 100, 1000\}$.

c)

The maximum relative errors are given in table I. We can see that the relative error decreases as we go up to 10^5 steps and increases again with additional steps. Fig. 5 shows that for $n_{\text{steps}} = 10^7$ we get a shape (somewhat) similar to what was predicted during lecture 5, I believe.

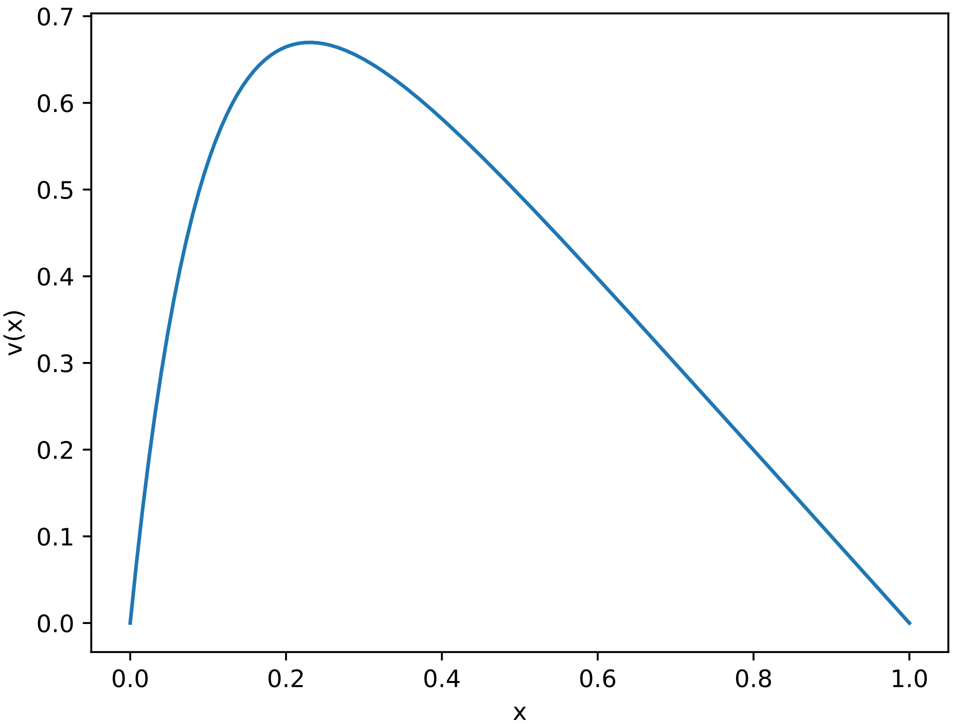

FIG. 2: The function $v(x)$ from $x = 0$ to $x = 1$.

TABLE I: Maximum relative errors

$\log_{10}(n_{\text{steps}})$	Maximum $\log_{10}(\epsilon_i)$
1	-1.25
2	-3.10
3	-5.08
4	-7.08
5	-8.84
6	-6.08
7	-5.53

PROBLEM 9

a)

As all elements in $\vec{a}$, $\vec{b}$ and $\vec{c}$ are the same (-1, 2 and -1 respectively) we can insert these values into algorithm 1. In the forward substitution we get

$$\begin{aligned} \tilde{g_1} &= g_1 \\ \tilde{b_1} &= 2 \\ \tilde{g_i} &= g_i + \frac{1}{2} g_{i-1} \\ \tilde{b_i} &= 2 - \frac{1}{2} = \frac{3}{2} \end{aligned} \tag{7}$$

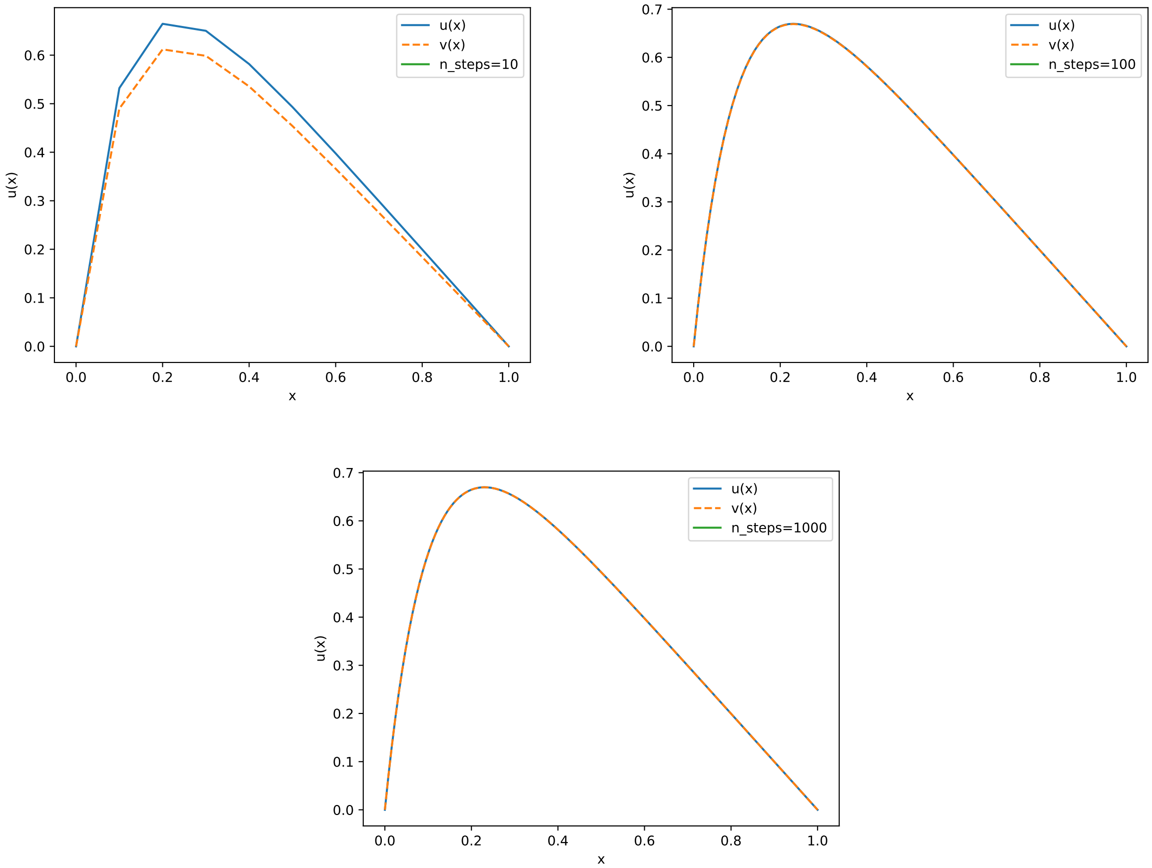

FIG. 3: $u(x)$ and $v(x)$ as functions of x using $n_{\text{steps}} \in \{10, 100, 1000\}$ steps.

In the back substitution we get

$$
\begin{aligned}
v_n &= \frac{2}{3}\tilde{g_n} = \frac{2}{3}(g_n + \frac{1}{2}g_{n-1}) \\
v_i &= \frac{2}{3}(\tilde{g_i} + v_{i+1}) = \frac{2}{3}(g_i + \frac{1}{2}g_{i-1} + v_{i+1}) \\
v_1 &= \frac{1}{2}(\tilde{g_1} + v_2) = \frac{1}{2}(g_1 + v_2)
\end{aligned}
\tag{8}
$$

where we needed to specify v_1 due to $\tilde{b_1} \neq \tilde{b_i}$. Written formally we get algorithm 2.

Algorithm 2 Special algorithm

We know all elements g_i.
$v_n = \frac{2}{3}(g_n + \frac{1}{2}g_{n-1})$
for $i = n-1, \ldots, 2$ **do**
    $v_i = \frac{2}{3}(g_i + \frac{1}{2}g_{i-1} + v_{i+1})$
$v_1 = \frac{1}{2}(g_1 + v_2)$

b)

The number of FLOPs in the special algorithm (not counting $\frac{1}{2}$ and $\frac{2}{3}$ as they are numbers) is $3 + 2 + (n-2) \cdot 4 = 4n - 3 \approx 4n$; 3 from v_n, 2 from v_1 and 4 from v_i $n-2$ times.

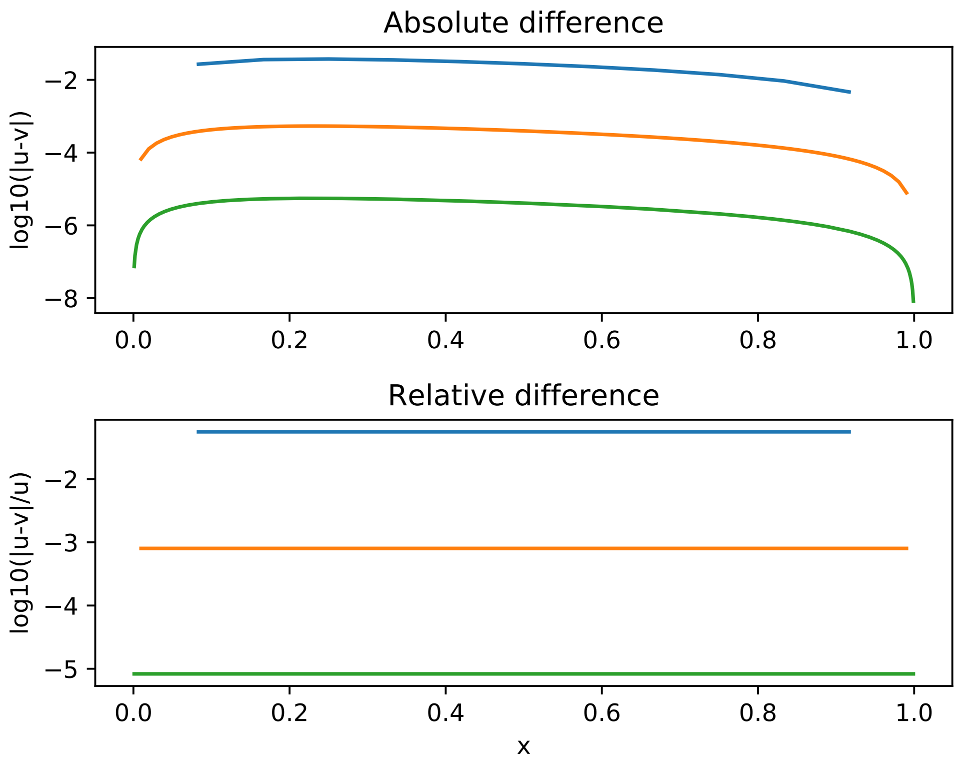

FIG. 4: $\log_{10}(|u_i - v_i|)$ and $\log_{10}(|\frac{u_i - v_i}{u_i}|)$ as functions of x_i. Blue corresponds to $n_{\text{steps}} = 10$, orange to 100, and green to 1000 steps.

TABLE II: Average times for algorithms

$\log_{10}(n_{\text{steps}})$	Gen. Alg. [s]	Spc. Alg. [s]	Ratio
1	$1.63 \cdot 10^{-6}$	$1.17 \cdot 10^{-6}$	1.39
2	$7.10 \cdot 10^{-6}$	$3.59 \cdot 10^{-6}$	1.98
3	$5.40 \cdot 10^{-5}$	$1.98 \cdot 10^{-5}$	2.73
4	$2.15 \cdot 10^{-4}$	$1.04 \cdot 10^{-4}$	2.07
5	$2.30 \cdot 10^{-3}$	$8.99 \cdot 10^{-4}$	2.56
6	$2.37 \cdot 10^{-2}$	$1.17 \cdot 10^{-2}$	2.03

c)

Just look at the code. It works.

PROBLEM 10

Note: General algorithm was slightly optimized by only doing the FLOP $\frac{a_i}{b_{i-1}}$ once per i instead of twice, giving a total of $8n$ FLOPs instead of $9n$.

The average time for each algorithm for the different n_{steps} are listed in table II. We can see from the ratio of the times that the general algorithm is somewhere between 2 to 3 times slower than the special algorithm. The expected ratio would be to compare the number of FLOPs, which is $\frac{8n}{4n} = 2$.

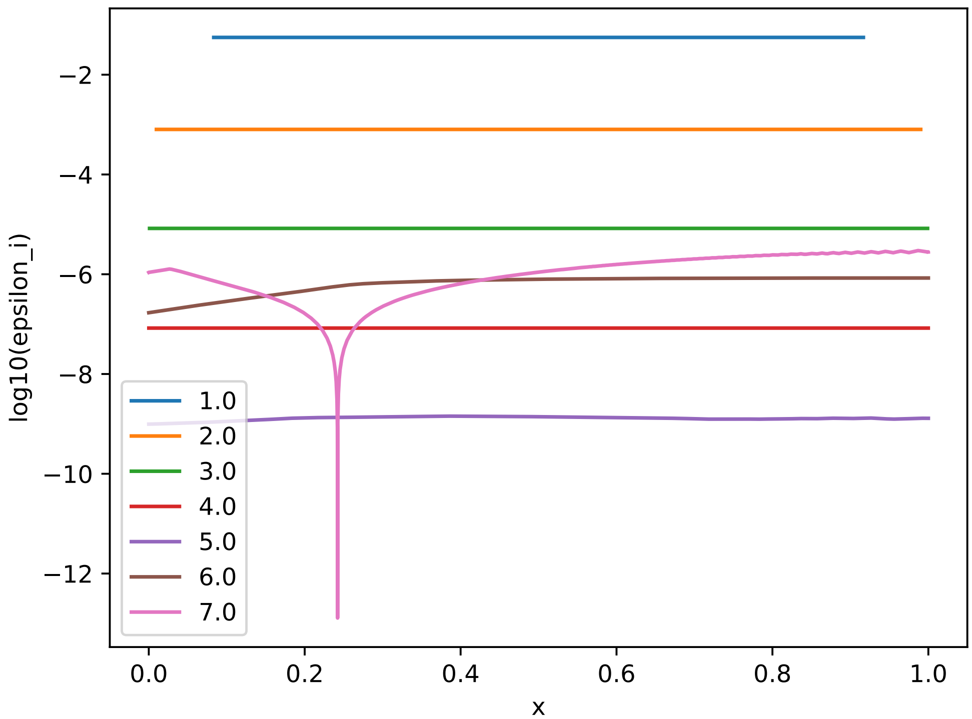

FIG. 5: Relative error for $\log_{10}(n_{\text{steps}}) \in \{1, 2, 3, 4, 5, 6, 7\}$.