FYS3150 - Project 1

Einar Skoglund and Emil Kvernevik.
(Dated: September 11, 2023)

The GitHub repository for this project can be found at https://github.uio.no/emiljk/FYS3150.

THE POISSON EQUATION

The Poisson equation

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

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

PROBLEM 1

We check analytically that an exact solution to the Poisson equation is given by

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

First we check the boundary conditions

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

Then we take the negative double derivative

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

PROBLEM 2

A plot of the exact solution (2) is shown in FIG 1.

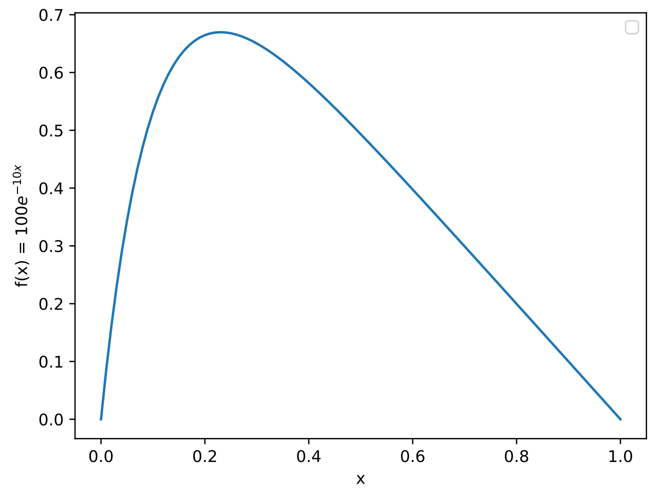

FIG. 1. The exact solution (2) to the Poisson equation.

PROBLEM 3

To derive a discretized version of the Possion equation, we start by adding Taylor expansions for $u(x + h)$ and $u(x - h)$, where h is the step size, to get an equation for the second derivative.

$$\begin{aligned}
u(x+h) &= \sum_{n=0}^{\infty} \frac{1}{n!} u^{(n)}(x) h^n \\
&= u(x) + u'(x)h + \frac{1}{2}u''(x)h^2 + \frac{1}{6}(x)h^3 + \mathcal{O}(h^4)
\end{aligned}$$

$$\begin{aligned}
u(x-h) &= \sum_{n=0}^{\infty} \frac{1}{n!} u^{(n)}(x) (-h)^n \\
&= u(x) - u'(x)h + \frac{1}{2}u''(x)h^2 - \frac{1}{6}(x)h^3 + \mathcal{O}(h^4)
\end{aligned}$$

We then add these two terms.

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

Then we rearrange for $u''(x)$.

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

We then change the notation $u(x) \to u_i$ since we are using discretized points, and not continuous ones. The discretized Poisson equation can then be written as

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

We approximate this equation by leaving out the truncation error $\mathcal{O}(h^2)$ and change the notation $v_i \approx u_i$. Further, arranging the v-terms yields

$$-v_{i-1} + 2v_i - v_{i+1} = h^2 f_i. \tag{4}$$

PROBLEM 4

We want to rewrite our discretized equation as a matrix equation. If we use 4 steps, we get three unknowns v_1, v_2 and v_3, and three equations:

$$\begin{array}{lllll} (i=1) & -v_0 + 2v_1 & -v_2 & = & h^2 f_1 \\ (i=2) & \quad - v_1 & +2v_2 - v_3 & = & h^2 f_2 \\ (i=3) & & -v_2 + 2v_3 \quad -v_4 & = & h^2 f_3. \end{array}$$

Since we know the boundary points v_0 and v_4 we move these to the right hand side, and use the notation g_i.

$$\begin{array}{lllllll} (i=1) & +2v_1 & -v_2 & = & h^2 f_1 + v_0 & = & g_1 \\ (i=2) & -v_1 & +2v_2 - v_3 & = & h^2 f_2 & = & g_2 \\ (i=3) & & -v_2 + 2v_3 & = & h^2 f_3 + v_4 & = & g_3 \end{array}$$

We then set the right hand side as the elements of a matrix $\mathbf{A}$ with a subdiagonal, main diagonal and superdiagonal with the signature (-1,2,-1). We can rewrite the set of equations above as a matrix equation $\mathbf{A}\vec{v} = \vec{g}$

$$\begin{bmatrix} 2 & 1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix} \tag{5}$$

PROBLEM 5

a)

Given that the vector $\vec{v}^*$ of length m represents the complete solution of the Poisson equation, then we know that the $n \times n$ matrix $\mathbf{A}$ represents the coefficients of the n unknowns in the set of n equations, without the boundary points v_0^* and v_{n+1}^*.

We therefore have the relation $n = m - 2$.

b)

When solving Eq. 4 for $\vec{v}$ we get the part of $\vec{v}^*$ without the boundary points v_0^* and v_{m-1}^*.

PROBLEM 6

a)

We will use the Thomas algorithm to solve the matrix equation $\mathbf{A}\vec{v} = \vec{g}$ where $\mathbf{A}$ is a general triangular matrix with vectors $\vec{a}$, $\vec{b}$ and $\vec{c}$ representing the subdiagonal, main diagonal and superdiagonal.

The Thomas algorithm uses a forward and backward substitution shown in algorithm 1. We will refer to this as the general algorithm.

Algorithm 1 General algorithm

procedure FORWARD SUBSTITUTION(n, a, b, c, g)
- $\tilde{b}_1 \leftarrow b_1$
- $\tilde{g}_1 \leftarrow g_1$
- **for** $i = 2, 3 \ldots n$ **do**
  - $\tilde{b}_i \leftarrow b_i - \frac{ai-1}{\tilde{b}_{i-1}} c_{i-1}$ ▷ $3(n-1)$ FLOPS
  - $\tilde{g}_i \leftarrow g_i - \frac{ai-1}{\tilde{b}_{i-1}} \tilde{g}_{i-1}$ ▷ $3(n-1)$ FLOPS

procedure BACK SUBSTITUTION$(v, \tilde{b}, \tilde{g}, c)$
- $v_n = \frac{\tilde{g}_n}{\tilde{b}_n}$ ▷ 1 FLOP
- **for** $i = n-1, n-2 \ldots 1$ **do**
  - $v_i \leftarrow \frac{\tilde{g}_i - c_i v_{i+1}}{\tilde{b}_i}$ ▷ $3(n-1)$ FLOPS

b)

We calculate the number of floating point operations (FLOPs) in this algorithm by adding the FLOPs for each calculating listed to the left in algorithm 1.

$$\text{FLOPs} = 9(n-1) + 1 = 9n - 8 \tag{6}$$

For large n this is roughly $9n$ FLOPs.

PROBLEM 7

With the general algorithm, we can calculate the solution $\vec{v}$ in equation 5 for the number of steps $n_{steps} = 10^1, 10^2, 10^3$ and 10^4. A comparison of these solutions and the exact solution $u(x)$ is shown in FIG. 2. As we see from this figure,

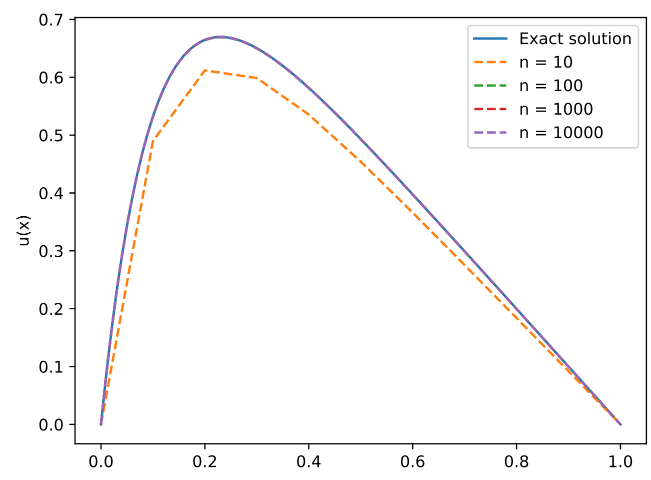

FIG. 2. Plot comparing the solutions of the general algorithm for different time steps n compared to the exact solution.

PROBLEM 8

a)

We calculate the logarithm of absolute error

$$\log_{10}(\Delta_i) = \log_{10}(|u_i - v_i|)$$

for different $n_{steps} = 10, 100, 1000$ and 10^7. This is shown in FIG. 3.

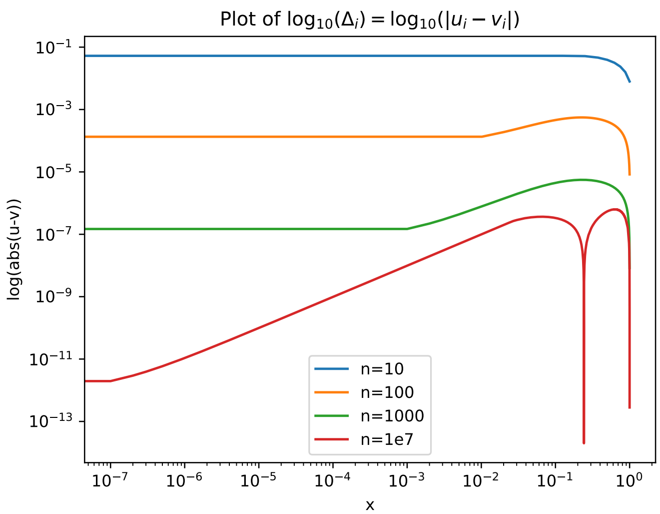

FIG. 3. Comparison of the absolute error for different time steps n using a logarithmic scale.

b)

A plot of the relative error

$$\log_{10}(\epsilon_i) = \log_{10}\left(\left|\frac{u_i - v_i}{u_i}\right|\right)$$

is shown in FIG. 4.

c)

The maximum relative error for each choice of n_{steps} up to $n_{steps} = 10^7$ is shown in table I and FIG. 5. From these we see a steady decrease in maximum relative error towards $n = 10^5$, before the error starts to increase for smaller step sizes. Since the truncation errors shrinks for an increasing number of steps, it is reasonable to assume that this increase in relative error is caused by round-off errors, which increase as the step size becomes smaller. From this, it is possible to argue that $n = 10^5$ is the optimal choice of steps regarding both truncation and round-off errors.

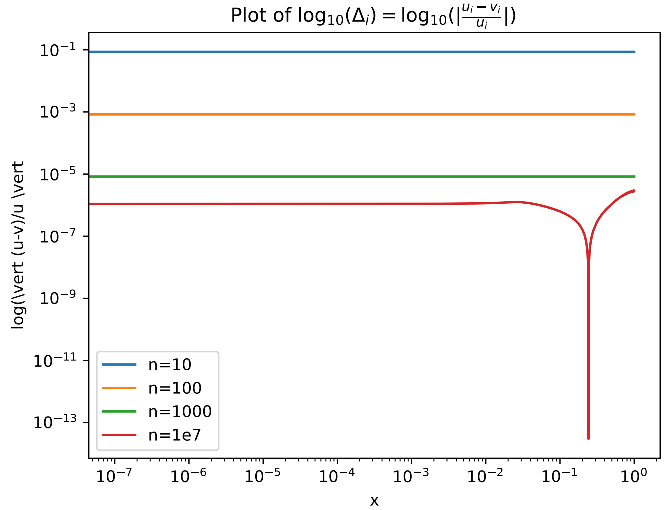

FIG. 4. Comparison of the relative error for different time steps n using a logarithmic scale.

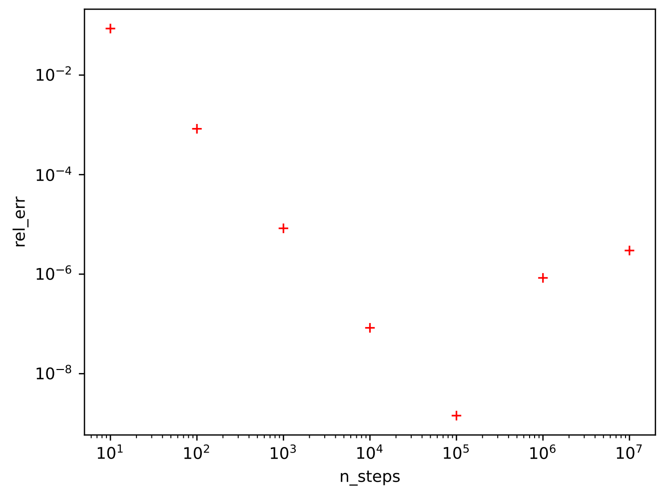

FIG. 5. Comparison of the maximum error for each time step n using a logarithmic scale.

PROBLEM 9

a)

We are now going to specialize our algorithm for the special case where the matrix $\mathbf{A}$ is specified by the signature (-1,2,-1), i.e. $a_i = c_i = -1$ and $b_i = 2$. By replacing a, b and c with these values we reduce the number of FLOPs. The specialized algorithm is shown in 2.

TABLE I. The maximum relative error $\max(\epsilon_i)$ compared for the exact solution for different time steps n_{steps} using the general algorithm.

n_{steps}	$\max(\epsilon_i)$
10^1	$8.6 \cdot 10^{-2}$
10^2	$8.3 \cdot 10^{-4}$
10^3	$8.3 \cdot 10^{-6}$
10^4	$8.3 \cdot 10^{-8}$
10^5	$1.4 \cdot 10^{-9}$
10^6	$8.4 \cdot 10^{-7}$
10^7	$3.0 \cdot 10^{-6}$

Algorithm 2 Special algorithm

procedure FORWARD SUBSTITUTION(n, a, b, c, g)
  $\tilde{b}_1 \leftarrow b_1$
  $\tilde{g}_1 \leftarrow g_1$
  for $i = 2, 3 \dots n$ **do**
    $\tilde{b}_i \leftarrow 2 - \frac{1}{\tilde{b}_{i-1}}$ ▷ $2(n-1)$ FLOPS
    $\tilde{g}_i \leftarrow g_i + \frac{\tilde{g}_{i-1}}{\tilde{b}_{i-1}}$ ▷ $2(n-1)$ FLOPS
procedure BACK SUBSTITUTION($v, \tilde{b}, \tilde{g}, c$)
  $v_n = \frac{\tilde{g}_n}{\tilde{b}_n}$ ▷ 1 FLOP
  for $i = n-1, n-2 \dots 1$ **do**
    $v_i \leftarrow \frac{\tilde{g}_i + v_{i+1}}{\tilde{b}_i}$ ▷ $3(n-1)$ FLOPS

b)

This specialized algorithm has $7n - 6$ FLOPs, which is roughly $7n$ for large n, i.e. $2n$ less than the general algorithm.

Using this algorithm gives the results in FIG. 6.

PROBLEM 10

We now run timing test for n_{steps} up to 10^6 for both the general and special algorithm. We do this three times for each algorithm, with the average values listed in table II. From this we see that the special algorithm is a lot quicker than the general algorithm, using below 2% of the time for $n_{steps} \geq 10^4$. This increase in efficiency comes from reducing the number of FLOPs, and by not iterating through the vectors $\vec{a}, \vec{b}$ and $\vec{c}$ in the matrix $\mathbf{A}$.

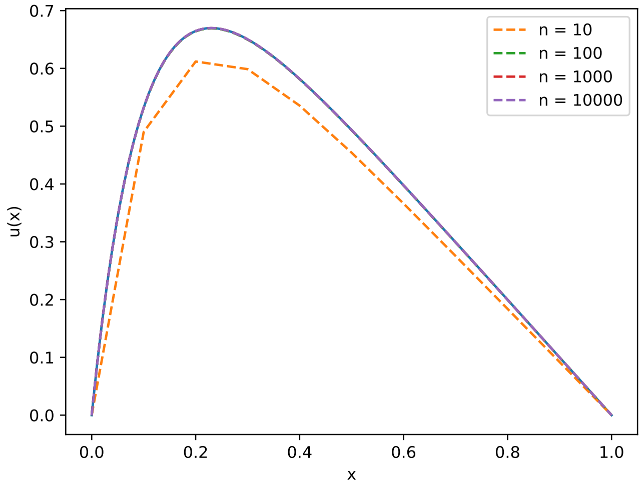

FIG. 6. Comparison of the absolute error for different time steps n using a logarithmic scale.

TABLE II. The average timing results comparing the general and special algorithm for different time steps n.

n_{steps}	$t_{general}$ (s)	$t_{special}$ (s)	$t_{special}/t_{general}$ (%)
10^1	$2.9 \cdot 10^{-3}$	$1.2 \cdot 10^{-3}$	41
10^2	$7.1 \cdot 10^{-3}$	$1.0 \cdot 10^{-4}$	14
10^3	$4.7 \cdot 10^{-3}$	$1.6 \cdot 10^{-4}$	3.4
10^4	$4.5 \cdot 10^{-2}$	$8.6 \cdot 10^{-4}$	1.9
10^5	$4.4 \cdot 10^{-1}$	$7.9 \cdot 10^{-3}$	1.8
10^6	$4.7 \cdot 10^{1}$	$7.8 \cdot 10^{-2}$	1.7