Project 1

Sunniva Kiste Bergan
(Dated: September 13, 2022)

https://github.com/sunnikbe/comfys/tree/main/Project1

THE ONE-DIMENSIONAL POISSON EQUATION

The one-dimensional Poisson equation can be written as

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

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

PROBLEM 1

Checking analytically that eq.1 is given by

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

We have

$$\frac{du}{dx} = e^{-10} + 10e^{-10x} - 1,$$
$$\frac{d^2u}{dx^2} = -100e^{-10x},$$

which corresponds to eq. 1 and our source term:

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

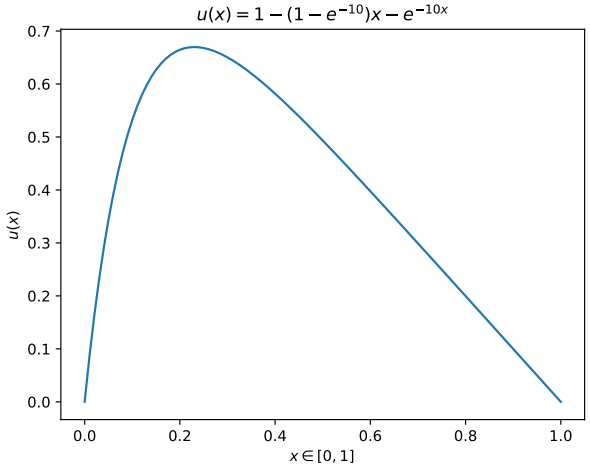

FIG. 1. Plot showing eq. 2 in the given x interval.

PROBLEM 2

A program (p2.cpp) computing eq. 2 and writing the x values and the computed u(x) values to a data file is found in the github repo. Plot of eq. 2 is shown in figure 1.

PROBLEM 3

Deriving a discretized version of the Poisson equation (eq. 1). We will use the finite difference method by using the central difference type. The approximation to $u''(x)$ is represented by $v''(x)$.

We have that

$$v_i' \approx \frac{v_{i+\frac{1}{2}} - v_{i-\frac{1}{2}}}{h}, \tag{3}$$

by the definition of the derivative, where h is the timestep between indices. To find the double derivative, we will use the same method as in eq. 3, with the whole expression from eq. 3.

This yields

$$v_i'' = \frac{\left(\frac{v_{i+1}-v_i}{h} - \frac{v_i - v_{i-1}}{h}\right)}{h} = \frac{v_{i+1} - 2v_i + v_{i-1}}{h^2}.$$

Eq. 1 can now be written as

$$-\left[\frac{v_{i+1} - 2v_i + v_{i-1}}{h^2}\right] = f_i, \tag{4}$$

where f_i represents $f(x_i)$.

PROBLEM 4

Eq. 4 can now be rewritten as a matrix equation

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

Rearranging eq. 4

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

The least amount of steps required to represent the discretized equation as a matrix equation is five. This is due to our two known boundary points and the fact that eq. 4 has three steps within the expression. Five steps, i.e. six points, where two of them are known.

$$\begin{array}{lccccc}
(i=1) & -v_0 + 2v_1 & -v_2 & & & = h^2 f_1 \\
(i=2) & -v_1 & +2v_2 - v_3 & & & = h^2 f_2 \\
(i=3) & & -v_2 + 2v_3 & -v_4 & & = h^2 f_3 \\
(i=4) & & -v_3 & +2v_4 - v_5 & & = h^2 f_4
\end{array}$$

This gives us

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}, \vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}$$

and

$$\vec{g} = \begin{bmatrix} h^2 f_1 + v_0 \\ h^2 f_2 \\ h^2 f_3 \\ h^2 f_4 + v_5 \end{bmatrix}.$$

PROBLEM 5

(a) Using the same reasoning as in problem 4, m should be $n+2$ long. We have the two boundary conditions that are known, and thus is not a part of the solution to the matrix equation.

(b) This means that when we solve eq. 5 for $\vec{v}$, we will find all but the first and last item in $\vec{v}*$, which are the initial conditions.

PROBLEM 6

(a) Equation 5 can be solved by using Gaussian elimination. Specifically, when used on an equation with a tridiagonal matrix such as $\mathbf{A}$, this turns into the Thomas algorithm. The diagonal vectors of $\mathbf{A}$ is $\vec{a}$,$\vec{b}$ and $\vec{c}$. The algorithm has two steps; forward substitution followed by backward substitution.

- Forward substitution algorithm:

$$\tilde{b}_1 = b_1$$
$$\tilde{b}_i = b_i - \frac{a_i}{\tilde{b}_{i-1}} c_{i-1}$$

$$\tilde{g}_1 = g_1$$
$$\tilde{g}_i = g_i - \frac{a_i}{\tilde{b}_{i-1}} \tilde{g}_{i-1},$$

for $i = 2, 3, ..., n$.

- Backward substitution algorithm:

$$v_n = \frac{\tilde{g}_n}{\tilde{b}_n}$$

$$v_i = \frac{\tilde{g}_i - c_i v_{i+1}}{\tilde{b}_i}$$

for $i = (n-1), (n-2), ..., 1$.

(b) For the forward substitution we have $(n-1)$ repetitions and 3 FLOPs for both $\tilde{b}_i$ and $\tilde{g}_i$. This gives

$$3(n-1) \cdot 2 = 6(n-1)$$

FLOPs. For The backward substitution we have the same number of operations for v_i, 3 FLOPs for v_i and one FLOP for v_n, which gives a total of

$$3(n-1) + 1 = 3n - 2$$

FLOPs.
Put together for the whole Thomas algorithm gives a total of

$$9n - 8$$

FLOPs.

PROBLEM 7

(a) A program (p7.cpp) using the general algorithm to compute eq. 5 and writing the complete solutions for $\vec{v}$ and $\vec{x}$ to a data file is found in the github repo.

(b) The program from (a) for $n_{steps} = 10, 10^2, 10^3, 10^4$, and a comparison to $u(x)$ is found in fig. 2.

PROBLEM 8

(a) A plot showing the logarithm of the absolute error,

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

is found in fig. 3.

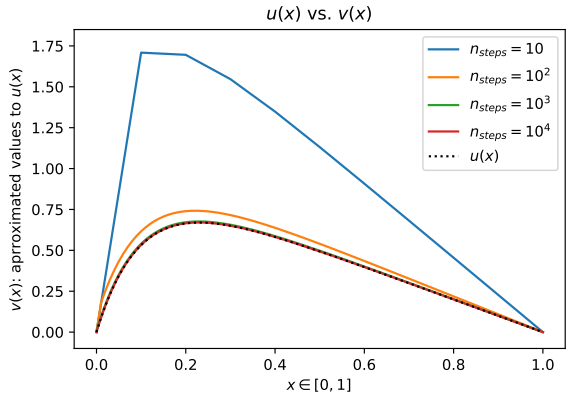

FIG. 2. Plot showing eq. 2 compared to the dicretized version $v(x_i)$ calculated using the Thomas algorithm with different step sizes.

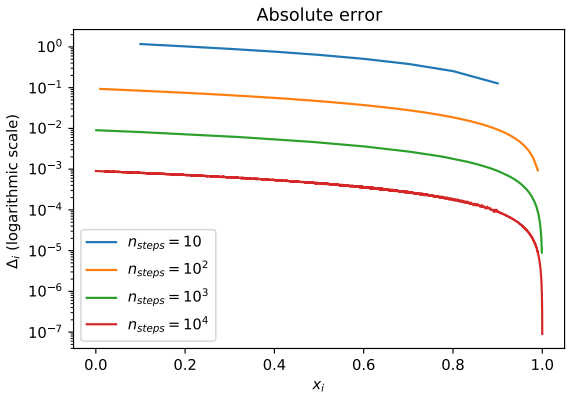

FIG. 3. Plot showing the logarithm of the absolute error, $log_{10}\Delta_i$ for the different numbers of steps.

n_{steps}	$log_{10}(max(\epsilon_i))$
10	$3.45 \cdot 10^{-1}$
10^2	$3.89 \cdot 10^{-2}$
10^3	$3.95 \cdot 10^{-3}$
10^4	$3.96 \cdot 10^{-4}$
10^5	$3.96 \cdot 10^{-5}$
10^6	$4.10 \cdot 10^{-6}$
10^7	$1.34 \cdot 10^{-6}$

TABLE I. Table showing n_{steps} in one column and the corresponding maximum relative error $log_{10}(max(\epsilon_i))$ in the other column.

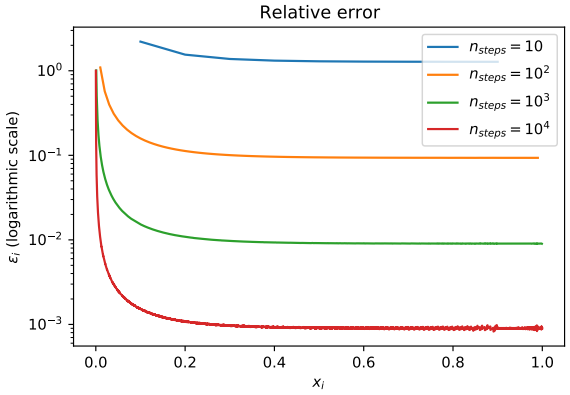

FIG. 4. Plot showing the logarithm of the relative error, $log_{10}(\epsilon_i)$ for the different numbers of steps.

(b) A plot showing the relative error,

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

is found in fig. 4.

(c) Table showing maximum relative error, $max(\epsilon_i)$, for $n_{steps} = 10, 10^2, ..., 10^7$ is found in table I. The values of the logarithm of the maximum relative error seems to be declining linearly compared to the increase in number of steps. This would of course mean that as the number of steps increase exponentially the maximum relative error decreases exponentially, as they both increment with a factor of 10. A plot of this table was made to see this more clearly. This is found in fig. 5. In the plot we can see that the above is true until $n_{steps} = 10^6$. After this we can see that increasing the number of steps (i.e. decreasing stepsize) by a factor of ten does no longer mean a decrease in the maximum relative error by a factor of ten. Using 10^7 steps therefore will use more computer power than what is gained in precision of the computation.

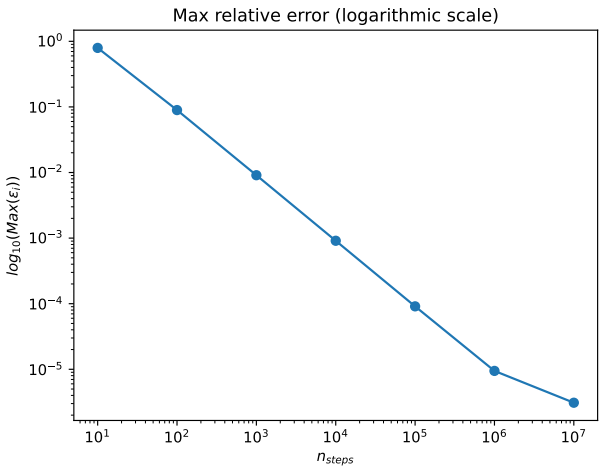

FIG. 5. Plot showing the logarithm of the maximum relative error, $log_{10}(max(\epsilon_i))$ for the different numbers of steps.

PROBLEM 9

(a) The algorithm from Problem 6 specialized for our special case, called the special algorithm:

- Forward substitution algorithm:

$$d = a/b$$

$$\tilde{g}_1 = g_1$$
$$\tilde{g}_i = g_i - d\tilde{g}_{i-1},$$

for $i = 2, 3, ..., n$.

- Backward substitution algorithm:

$$v_n = 0$$

$$v_i = \frac{\tilde{g}_i - cv_{i+1}}{b}$$

for $i = (n-1), (n-2), ..., 1$.

(b) For the forward substitution part we have $2(n-1)+1$ FLOPs, and for the backward substitution part we have $3(n-1)$ FLOPs. In total

$$5n - 4$$

FLOPs for the special algorithm.

(c) Code implementing the special algorithm (p9.cpp) is found in the github repo.

PROBLEM 10

Code examples from Project 1 was used to write two programs (p10.cpp, p10s.cpp) used in comparing the general algorithm (p7.cpp) to the special algorithm (p9.cpp). The results are in table II.

n_{steps}	General alg. [s]	Special alg. [s]
10	$(4.2 \pm 0.4) \cdot 10^{-6}$	$(5.0 \pm 1.0) \cdot 10^{-6}$
10^2	$(9.0 \pm 1.0) \cdot 10^{-6}$	$(6.0 \pm 1.0) \cdot 10^{-6}$
10^3	$(5.0 \pm 1.0) \cdot 10^{-5}$	$(3.2 \pm 0.3) \cdot 10^{-5}$
10^4	$(5.0 \pm 0.7) \cdot 10^{-4}$	$(2.8 \pm 0.1) \cdot 10^{-4}$
10^5	$(5.0 \pm 0.8) \cdot 10^{-3}$	$(3.3 \pm 0.7) \cdot 10^{-3}$
10^6	$(4.9 \pm 0.7) \cdot 10^{-2}$	$(3.2 \pm 0.7) \cdot 10^{-2}$

TABLE II. Table showing $t = \bar{t} \pm \Delta t$ measured for each of the algorithms. The times were measured for n_{steps} from 10 to 10^6.

As the number of steps increase, the special algorithm runs faster than the general algorithm. Which was expected. The number of FLOPs is lower for the special algorithm than for the general algorithm. For a lower number of steps both algorithms run in approximately the same time. This is probably due to the computer and not the algorithm, as the difference gets bigger as n_{steps} increase.
Each of the algorithms were timed five times for each number of steps. The uncertainty of the values in table II are therefore relatively high compared to the mean values themselves.