Project 1 FYS4150

Vetle A. Vikenes, Johan Mylius Kroken and Nanna Bryne
(Dated: September 13, 2022)

The code can be found on GitHub at https://github.com/Vikenes/FYS4150/tree/main/project1.

INTRODUCTION

We will solve the one-dimensional Poisson equation

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

where the source function, $f(x) = 100e^{-10x}$, is known. We will do this for $x \in [0, 1]$ with boundary conditions $u(0) = u(1) = 0$.

PROBLEM 1

We want to check that

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

is the solution of eq. (1) given our source function, $f(x)$. We first control that eq. (2) satisfies the boundary conditions.

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

We find the double derivative of $u(x)$,

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

and see that $u(x)$ in eq. (2) satisfies equation (1), with our given source function.

PROBLEM 2

We write a program in C++ that defines a vector of linearly spaced values of $x \in [0, 1]$, evaluates the exact solution from eq. (2) at these points and saves the information. We then use Python to read the data and plot the solution, which is shown in figure 1.

PROBLEM 3

For a discrete value of x at a point i, we denote this as x_i, where the corresponding function value is defined as $u(x_i) \equiv u_i$. Similarly, the source function is defined as $f(x_i) \equiv f_i$. We now find the discretized version of the second order derivative of u_i

$$\left.\frac{\mathrm{d}^2 u}{\mathrm{d}x^2}\right|_{x_i} = u_i'' = \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} + \mathcal{O}(h^2) \approx \frac{v_{i-1} - 2v_i + v_{i+1}}{h^2} = v_i''$$

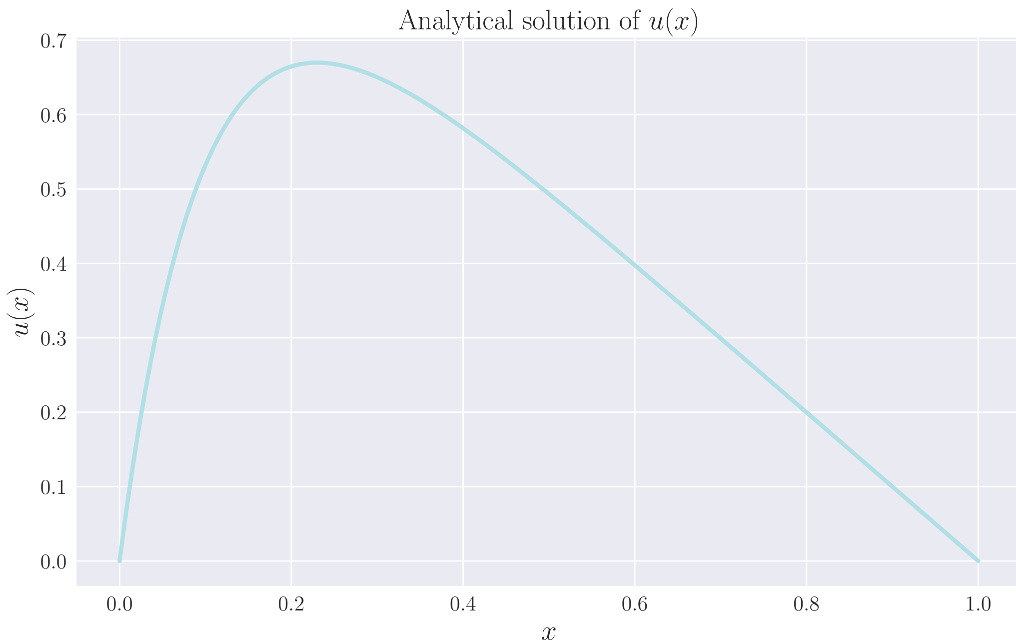

FIG. 1. The analytical solution in eq. (2).

where v_i is the approximated value of u_i obtained by neglecting the $\mathcal{O}(h^2)$ term, and $h = (x_{\max} - x_{\min})/n_{\text{steps}}$ is the step length.

Inserting for v_i into the Poisson equation yields

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

which is a discretized version of the Poisson equation.

PROBLEM 4

The computations performed in equation (3) to obtain a value for f_i can be expressed in terms of vectors. If we have two column vectors, $\mathbf{a}$ and $\tilde{\mathbf{v}}$, defined as $\mathbf{a} = (-1,\, 2,\, -1)$ and $\tilde{\mathbf{v}} = (v_{i-1},\, v_i,\, v_{i+1})$, respectively, equation (3) can be written as

$$\mathbf{a}^T\,\tilde{\mathbf{v}} = \begin{pmatrix} -1 & 2 & -1 \end{pmatrix} \begin{pmatrix} v_{i-1} \\ v_i \\ v_{i+1} \end{pmatrix} = -v_{i+1} + 2v_i - v_{i-1} = h^2 f_i \equiv g_i.$$

We can extend this to all n values of x_i, computing $\mathbf{g} = (g_0, \ldots, g_{n-1})$ by multiplying $\mathbf{v} = (v_0, \ldots, v_{n-1})$ with a tridiagonal matrix A with 2 on its main diagonal and -1 on its subdiagonal and superdiagonal. However, for v_0 and v_{n-1} we are unable to compute the second derivative, since v_{-1} and v_n are not defined. From the boundary conditions we have $v_0 = v_{n-1} = 0$, so v_1, v_{n-2} and all the elements in between can be computed. Omitting the end points, equation (3) can be written as a matrix equation by

$$A\mathbf{v} = \begin{pmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & & -1 \\ & & -1 & 2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-2} \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_{n-2} \end{pmatrix} = \mathbf{g}, \tag{4}$$

where the value of the original differential equation is obtained by $\mathbf{f} = \mathbf{g}/h^2$. For the end-points we actually have $2v_1 - v_2 = g_1 + v_0$ and $-v_{n-3} + 2v_{n-2} = g_{n-2} + v_{n-1}$, but we have ommited these additional terms as they both are zero.

PROBLEM 5

a)

We have two vectors of length m, $\mathbf{v}^*$ and $\mathbf{x}$, representing a complete solution to the discretized Poisson equation and corresponding x values, respectively. With a matrix A being the tridiagonal matrix from problem 4, we can find how n, relates to m. For all elements of $\mathbf{v}^*$ to be a complete solution, the derivative in equation (3) must be applicable, hence $\mathbf{v}^* = (v_1, v_2, ..., v_{n-2})$ lacks the boundaries and is therefore two elements "shorter" than $\mathbf{v}$, i.e. $m = n - 2$.

b)

When solving (4) for $\mathbf{v}$, we get all the elements of $\mathbf{v}^*$. The remaining elements of $\mathbf{v}$ that are missing, are the end points, which is given by the boundary conditions, $v_0 = v_{n-1} = 0$.

PROBLEM 6

a)

We now concern ourselves with the general solution of the matrix equation $A\mathbf{v} = \mathbf{g}$ where A is a *general* $n \times n$ tridiagonal matrix. We thus have the following:

$$A\mathbf{v} = \begin{pmatrix} b_1 & c_1 & & \\ a_2 & \ddots & \ddots & \\ & \ddots & & c_{n-1} \\ & & a_n & b_n \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix} = \mathbf{g}.$$

In order to find a general solution for $\mathbf{v}$ for such a tridiagonal matrix we use the method of Gaussian elimination. In the end, we end up with an algorithm called the Thomas algorithm[1].

We summarize the algorithm as follows:

Algorithm 1 General algorithm

```
b̃_0 = b_0                                   ▷ Define initial values
g̃_0 = g_0
for i = 1, 2, ..., m − 1 do                  ▷ Do forward substitution
    K = a_i / b̃_{i−1}
    b̃_i = b_i − K · c_{i−1}
    g̃_i = g_i − K · g̃_{i−1}

v_{m−1} = g̃_{m−1} / b̃_{m−1}                  ▷ Define initial value
for i = m − 2, m − 3, ..., 0 do              ▷ Do backward substitution
    v_i = (g̃_i − v_{i+1} · c_i) / b̃_i
```

[1] The complete derivation is given in Appendix A

b)

The first loop in Algorithm 1 contains $1+2+2=5$ FLOPs and runs $m-2$ times. In the second loop we perform $1+1+1=3$ FLOPs per iteration. Remembering the operation prior to the second loop gives a total of $5(m-2)+1+3(m-2)=8(m-2)+1$ FLOPs.

PROBLEM 7

a)

We implement the general algorithm, Algorithm 1, in C++ and let A be the matrix from Problem 4, i.e. $a_i = c_i = -1$ and $b_i = 2$. For a given number of discretization steps, n_{steps}, the code saves the solution as pairs (x_i, v_i) to a file.

b)

We compute $\mathbf{v}$ with the general algorithm for $n_{\text{steps}} = 10$, 100 and 1000 and plot the numerical solutions up against the analytic solution in Figure 2. We see that $n_{\text{steps}} = 10$ produces a large deviation from the exact solution, whereas $n_{\text{steps}} = 100$ yields a far smaller deviation. Choosing $n_{\text{steps}} = 1000$ reproduces a good approximation to the exact solution, with no visible deviation. To quantify the validity of these results, we proceed by studying the errors from the different step sizes.

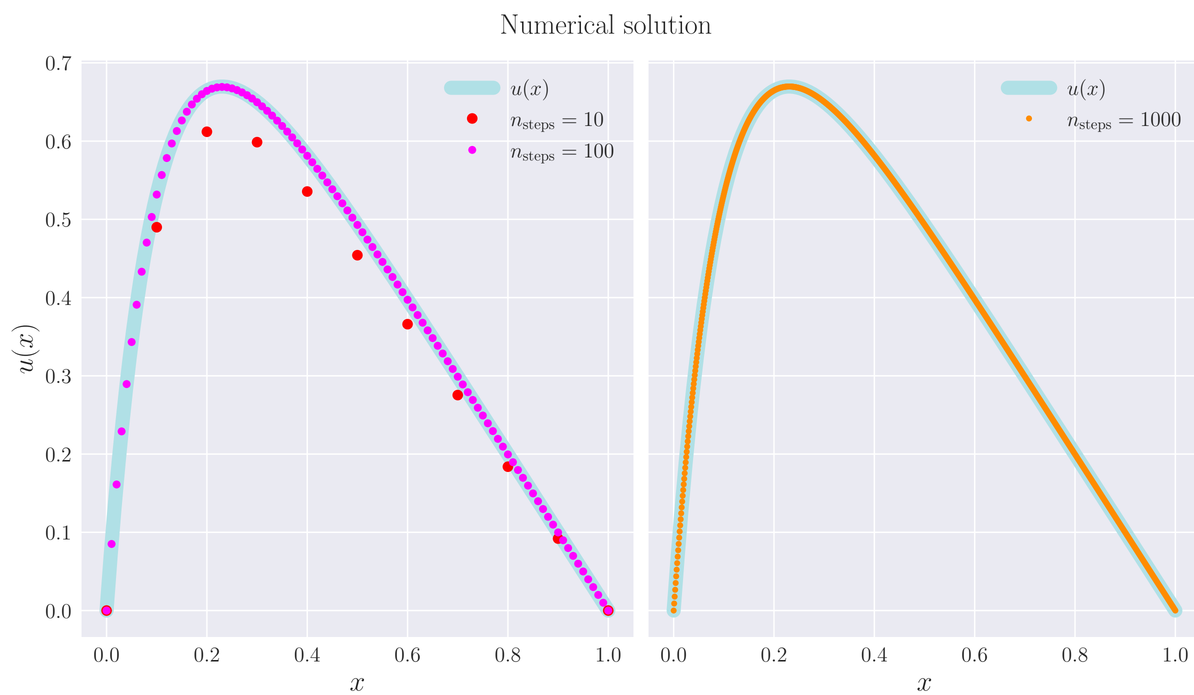

FIG. 2. The numerical solution to the Poisson equation using Algorithm 1 with different choices of n_{steps}, compared the the exact solution.

PROBLEM 8

We ommit the end points in this problem, since we have defined $v_i = u_i$ at these points.

a)

For the same n_{steps} used in Figure 2 we find the absolute error $\Delta_i = |u_i - v_i|$, where $u_i = u(x_i)$ from eq. (2). We plot $\log_{10}(\Delta_i(x_i))$ for different n_{steps} in Figure 3. We see that the error reduces by a factor $\sim 10^{-2}$ as we increase the steps size by a factor 10. This is consistent with the error we get from omitting the $\mathcal{O}(h^2)$ terms, since $h^2 \propto n_{\text{steps}}^{-2}$. As seen from figure 1, the analytical solution has its largest gradient at low x values. Near the end-points, we get a small absolute error, since these points are approximated from the exact values at $x = 0$ and $x = 1$. With a lower gradient of $u(x)$ towards $x = 1$, we see that the error desclines as x increases.

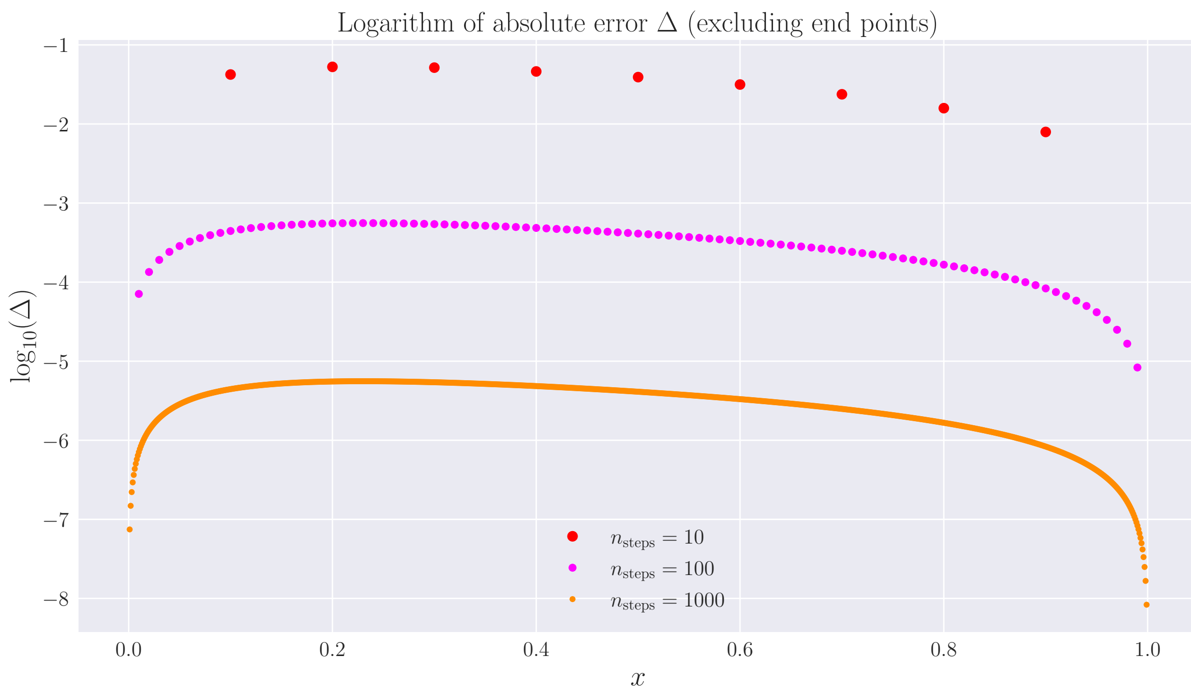

FIG. 3. The logarithm of the absolute error as function of x in the solution computed using Algorithm 1 for different n_{steps}.

b)

We now compute the relative error $\epsilon_i = \frac{\Delta_i}{|u_i|}$, and plot $\log_{10}(\epsilon_i)$, shown in Figure 4. Since we're now scaling the error, the relative error is independent of x, and is determined solely by the value of n_{steps}.

c)

We compute solutions for $n_{\text{steps}} \in [10^1, 10^2, \ldots, 10^7]$ and find the related maximum values, $\max(\epsilon_i)$, for each value of n_{steps}. The results are presented in table I with a corresponding plot shown in Figure 5. In the base-10 logarithm space, we see a linear decrease in relative error as the number of steps reaches 10^5. For $n_{\text{steps}} \geq 10^6$ there is an increase in error, as we're dealing with numbers close to machine precision.

PROBLEM 9

a)

We specialize the Thomas algorithm to the special case of the tridiagonal, symmetric Toeplitz matrix A with signature $(-1, 2, -1)$. This special algorithm is given in Algorithm 2

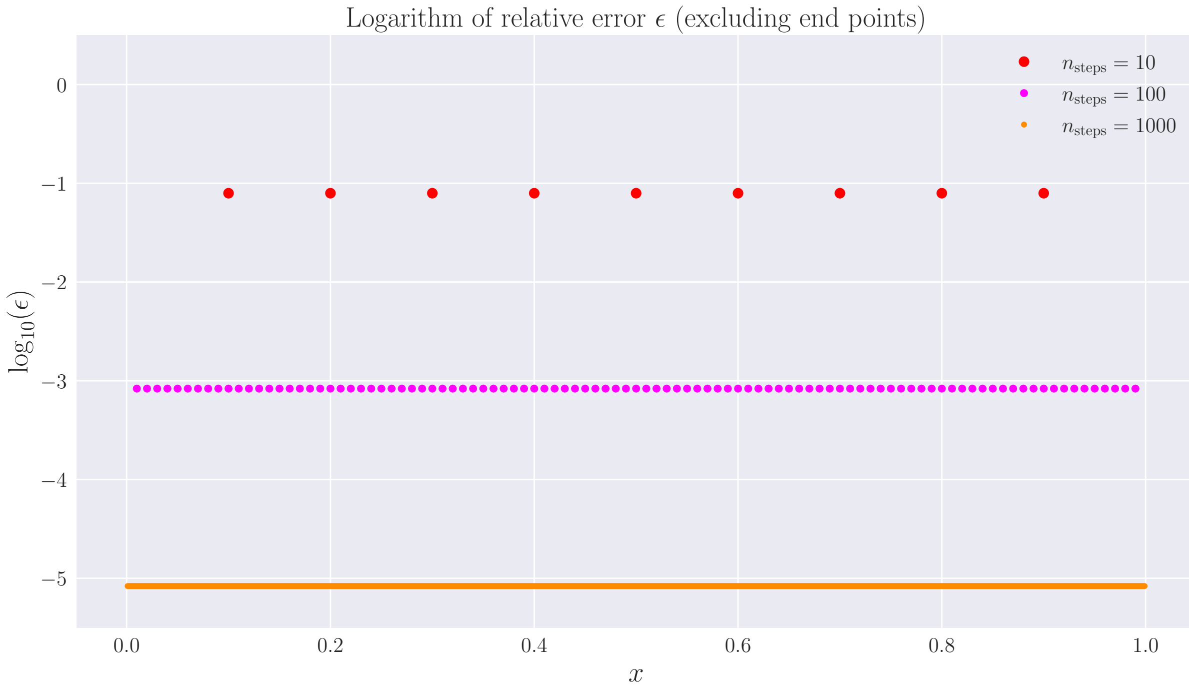

FIG. 4. The logarithm of the relative error as function of x in the solution computed using Algorithm 1 for different n_{steps}.

TABLE I. Maximum relative error per n_{step}

n_{steps}	$\max(\epsilon)$
10^1	7.932641e-02
10^2	8.329168e-04
10^3	8.333292e-06
10^4	8.333006e-08
10^5	1.435597e-09
10^6	8.404123e-07
10^7	2.983801e-06

Algorithm 2 Special algorithm

$\tilde{b}_0 = b_0$
$\tilde{g}_0 = g_0$
for $i = 1, 2, ..., m-1$ **do**
  $\tilde{b}_i = (i+2)/(i+1)$
  $\tilde{g}_i = g_i + \tilde{g}_{i-1}/\tilde{b}_{i-1}$
$v_{m-1} = \tilde{g}_{m-1}/\tilde{b}_{m-1}$
for $i = m-2, m-3, ..., 0$ **do**
  $v_i = (\tilde{g}_i + v_{i+1})/\tilde{b}_i$

Here we have simply substituted for a_i, b_i and c_i in Algorithm 1 and recognized that $(a_i/\tilde{b}_{i-1}) \cdot c_{i-1} = 1/2$ for the first iteration, which yields the formula $\tilde{b}_i = (i+2)/(i+1)$. Similar considerations have been made to simplify the expressions for $\tilde{g}_i$ and v_i.

b)

Algorithm 2 performs 1 and 2 FLOPs for computing $\tilde{b}_i$ and $\tilde{g}_i$, respectively, yielding $3(m-2)$ FLOPs in the first loop. 1 FLOP is performed before the second loop, and $2 \cdot (m-2)$ FLOPs are performed in the second loop, yielding

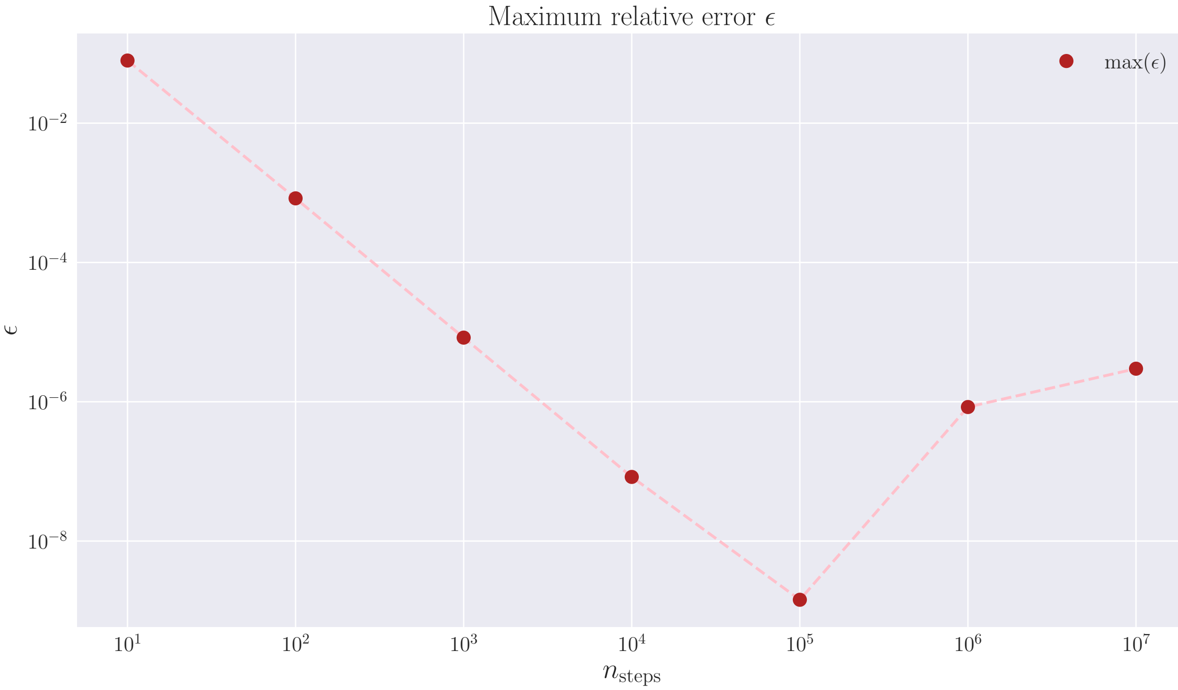

FIG. 5. The maximum relative error plotted over the number of steps used in Algorithm 1.

TABLE II. The mean μ and standard deviation σ of time spent to execute the general and special Thomas algorithms

n_{steps}	$\mu_G[s]$	σ_G [s]	μ_S [s]	σ_S [s]
10^1	0.000002	6.410620e-08	0.000001	6.419160e-07
10^2	0.000007	5.700580e-07	0.000005	4.185220e-08
10^3	0.000057	4.404390e-07	0.000043	6.829020e-07
10^4	0.000559	1.158680e-05	0.000426	4.409930e-06
10^5	0.006123	1.860330e-04	0.004404	1.504870e-04
10^6	0.094134	3.354690e-04	0.062452	3.423120e-04

a total of $5(m-2)+1$ FLOPs in the special Thomas algorithm.

c)

We implement Algorithm 2 in our C++ script.

PROBLEM 10

We write a code that for $n_{\text{steps}} \in [10^1, 10^2, \ldots, 10^6]$ computes Algorithm 1 and Algorithm 2 500 times, and finds the average duration of the two algorithms for each n_{steps}. In addition, we find the root mean square error of the measurements. In Figure 6 we visualise the results that are listed in table II.

In figure 6 we see that the special algorithm is quicker than the general one. The only noticeable feature is the standard deviation of the general algorithm for $n_{\text{steps}} = 10$. However, the standard deviations at small step sizes are on the order of micro seconds, where our result is relatively sensitive to hardware specifics. To see how the time differences compare to the number of FLOPs in the two algorithms, we tabulate μ_S/μ_G for each step, together with the ratio between the number of FLOPs from the general and special algorithm, which are $8(m-2)+1$ and $5(m-2)+1$, respectively. The result is given in table III. The general algorithm performs slightly better than what it would if FLOPs were the only governing factor. This is expected, since memory handling and various other processes affect

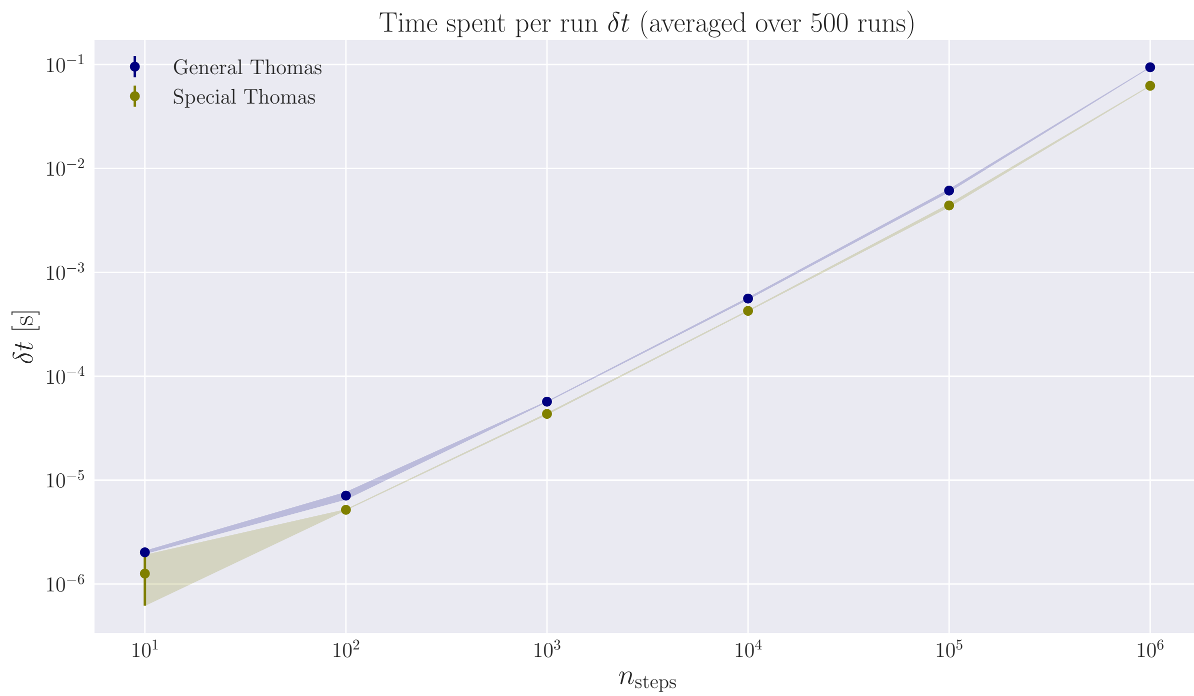

FIG. 6. The plot shows timing results for running the general (blue) and special (green) Thomas algorithms. The average duration (dots), and corresponding RMS error in measurements (bars), of one run are plotted for different choices in number of steps.

n_{steps}	$S_{\text{FLOPs}}/G_{\text{FLOPs}}$	μ_S/μ_G [s]
10^1	0.6308	0.6244
10^2	0.6255	0.7305
10^3	0.6250	0.7624
10^4	0.6250	0.7614
10^5	0.6250	0.7192
10^6	0.6250	0.6634

TABLE III. The ratio of average times and ratio of FLOPs for the two algorithms for different number of steps.

the timing. However, there appear to be a reasonable correspondence between difference in FLOPs and difference in time for the two algorithms.

Appendix A: Derivation of the Thomas algorithm

In order to derive the Thomas algorithm, we monitor how a series of row operations can achieve the following relation:

$$\begin{pmatrix} A & \mathbf{g} \end{pmatrix} \sim \begin{pmatrix} \mathbb{I}^n & \mathbf{v} \end{pmatrix}$$

which is the same as:

$$\begin{pmatrix} b_1 & c_1 & & & g_1 \\ a_2 & \ddots & \ddots & & g_2 \\ & \ddots & & c_{n-1} & \vdots \\ & & a_n & b_n & g_n \end{pmatrix} \sim \begin{pmatrix} \mathbb{I}^n & \mathbf{v} \end{pmatrix}$$

We have obtained $\mathbb{I}^n$ if all elements in $\mathbf{a}$ are equal to 1, and all elements in $\mathbf{b}$ and $\mathbf{c}$ are equal to 0. We firs remove the $\mathbf{a}$ vector by forward substitution. The first row will remain unchanged, but we need to perform row operations on the remaining rows:

$$\begin{aligned}
\tilde{R}_1 &= R_1 \\
\tilde{R}_2 &= R_2 - \frac{a_2}{\tilde{b}_1}\tilde{R}_1 \\
&\vdots \\
\tilde{R}_n &= R_n - \frac{a_n}{\tilde{b}_{n-1}}\tilde{R}_{n-1}
\end{aligned}$$

After this forward substitution we have that all elements of $\mathbf{a}$ is 0, and:

$$\begin{aligned}
\tilde{b}_1 &= b_1 \\
\tilde{b}_2 &= b_2 - \frac{a_2}{\tilde{b}_1}c_1 \\
&\vdots \\
\tilde{b}_n &= b_n - \frac{a_n}{\tilde{b}_{n-1}}c_{n-1}
\end{aligned}$$

For $\mathbf{g}$ we have likewise:

$$\begin{aligned}
\tilde{g}_1 &= g_1 \\
\tilde{g}_2 &= g_2 - \frac{a_2}{\tilde{b}_1}\tilde{g}_1 \\
\vdots &= \\
\tilde{g}_n &= g_n - \frac{a_n}{\tilde{b}_{n-1}}\tilde{g}_{n-1}
\end{aligned}$$

The next step is to get tid of $\mathbf{c}$ and normalise $\tilde{\mathbf{b}}$ in order to obtain the identity matrix. This is done through backward substitution:

$$\begin{aligned}
\tilde{R}_n^* &= \frac{\tilde{R}_n}{\tilde{b}_n} \\
\tilde{R}_{n-1}^* &= \frac{\tilde{R}_{n-1} - c_{n-1}\tilde{R}_n^*}{\tilde{b}_{n-1}} \\
&\vdots \\
\tilde{R}_1^* &= \frac{\tilde{R}_1 - c_1\tilde{R}_2^*}{\tilde{b}_1}
\end{aligned}$$

We then write in terms of $v_i = \tilde{g}_i^*$:

$$\begin{aligned}
v_n &= \tilde{g}_n^* = \frac{\tilde{g}_n}{\tilde{b}_n} \\
v_{n-1} &= \frac{\tilde{g}_{n-1} - v_n c_{n-1}}{\tilde{b}_{n-1}} \\
v_1 &= \frac{\tilde{g}_1 - v_2 c_1}{\tilde{b}_1}
\end{aligned}$$

To summarize: We define $\tilde{b}_1 = b_1$ and $\tilde{g}_1 = g_1$. Through iteration, the follwing terms, valid for $i \in [2, n]$ become

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

We now obtain an expression for $v_n = \tilde{g}_n^* = \tilde{g}_n / \tilde{b}_n$, and get the remaining elements by backwards iteration

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

for $i \in [n-1,\, 1]$.