FYS4150 - Project 1

Halvor Melkild
(Dated: September 20, 2022)

GitHub repo: https://github.com/halvorme/FYS4150

PROBLEM 1

We are looking at the one-dimensional Poisson equation

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

on the range $x \in [0,1]$. The boudary conditions are $u(0) = u(1) = 0$ and source term is chosen as

$$f(x) = 100\,\mathrm{e}^{-10x}. \tag{2}$$

An exact solution to Equation 1 has the form

$$u(x) = -\mathrm{e}^{-10x} + C_1x + C_2. \tag{3}$$

The boundary condition at $x = 0$ gives that $C_2 = 1$, and the condition at $x = 1$ then gives that $C_1 = \mathrm{e}^{-10} - 1$. It follows that

$$u(x) = 1 - (1 - \mathrm{e}^{-10})x - \mathrm{e}^{-10x} \tag{4}$$

is a solution to Equation 1, satisfying the given boundary conditions.

PROBLEM 2

The plot of the exact solution is seen in Figure 1. The generating code is found in the repository linked at the top of the document.

PROBLEM 3

To make a discretised version of the Poisson equation, we need to discretise the second derivative. We start out with the definition of the derivative

$$u'(x) = \frac{\mathrm{d}u}{\mathrm{d}x} = \lim_{\Delta x\to 0}\frac{u(x+\Delta x) - u(x)}{\Delta x}. \tag{5}$$

If a function is differentiable at a point x it means that the limit has to be the same if you approach x from above or below. It follows that

$$u'(x) = \lim_{\Delta x\to 0}\frac{u(x) - u(x-\Delta x)}{\Delta x} \tag{6}$$

is an equivalent definition. The second derivative can then be defined the following way,

$$\begin{aligned} u''(x) = \frac{\mathrm{d}^2u}{\mathrm{d}x^2} &= \lim_{\Delta x\to 0}\frac{u'(x+\Delta x) - u'(x)}{\Delta x} \\ &= \lim_{\Delta x\to 0}\frac{u(x+\Delta x) - 2u(x) + u(x-\Delta x)}{\Delta x^2}. \end{aligned} \tag{7}$$

When we discretise the x-axis it will no longer be possible to take the limit $\Delta x \to 0$. The smallest possible value is $\Delta x = h$, where h is the stepsize of the discretisation. We will than have to find an approximation for $u''(x)$. By Taylor expanding

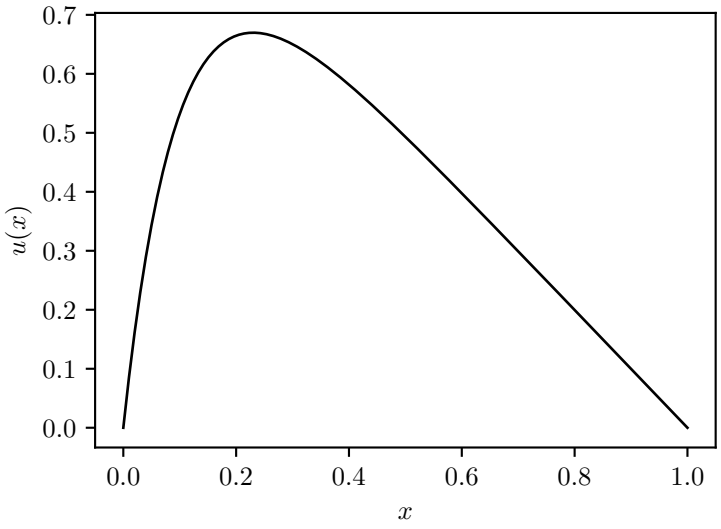

FIG. 1. The plot shows the exact solution to the Poisson equation (Eq. 1), with the source term $f(x) = 100\mathrm{e}^{-10x}$, and boundary conditions $u(0) = u(1) = 0$.

We would now like to discretise this expression. We start by discretising the x-axis. The line segment $[1, 0]$ is replaced by the set with $N + 1$ elements,

$$\{x_i = ih \mid i \in \{0, 1, \ldots, N\},\ h = \frac{1}{N}\}. \tag{8}$$

The distance between each point is given by h. We notice that $x_i + nh = x_{i+n}$ for any integer n. The function $u(x)$ is now replaced by the set

$$v_i = u(x_i), \tag{9}$$

where we choose denote the solution to the discretised equation as v. The boundary conditions tells us that $v_0 = v_N = 0$. For the source term we will simply use $f_i = f(x_i)$.

When the function $u(x)$ is discretised, we can't take the limit $\Delta x \to 0$ any more. The best we can do is $\Delta x \to h$. We than has to work with an approximation to Equation 7. By Taylor expanding

$$u(x \pm h) = u(x) \pm hu'(x) + \frac{1}{2}h^2u''(x) \pm \frac{1}{6}h^3u^{(3)}(x) + \frac{1}{24}h^4u^{(4)} + \mathcal{O}(h^5) \tag{10}$$

we find that

$$u''(x) = \frac{u(x + h) - 2u(x) + u(x - h)}{h^2} + \mathcal{O}(h^2). \tag{11}$$

Discarding the terms of $\mathcal{O}(h^2)$ and discretising the remainding expression we get

$$v_i'' = \frac{1}{h^2}(v_{i+1} - 2v_i + v_{i-1}). \tag{12}$$

In the end, the discretised Poisson equation (Eq. 1) can be written as

$$-v_{i-1} + 2v_i - v_{i+1} = h^2 f_i, \tag{13}$$

where $i \in \{1, \ldots, N - 1\}$.

PROBLEM 4

The discretised Poisson equation (Eq. 13) gives us a set of $N-1$ equations, with $N-1$ unknowns, as v_0 and v_N are set by the boundary conditions. We can write each of these equation on matrix form

$$\begin{pmatrix} -1 & 2 & -1 \end{pmatrix} \begin{pmatrix} v_{i-1} \\ v_i \\ v_{i+1} \end{pmatrix} = h^2 f_i. \tag{14}$$

For the first and last equation we can use the boundary conditions and simplify them as

$$\begin{aligned} \begin{pmatrix} 2 & -1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} &= h^2 f_1, \\ \begin{pmatrix} -1 & 2 \end{pmatrix} \begin{pmatrix} v_{N-2} \\ v_{N-1} \end{pmatrix} &= h^2 f_{N-1}. \end{aligned} \tag{15}$$

Stacking all these equations in one matrix we get

$$\begin{pmatrix} 2 & -1 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 & 0 \\ 0 & 0 & -1 & 2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & 0 & 0 & \cdots & -1 & 2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ \vdots \\ v_{N-2} \\ v_{N-1} \end{pmatrix} = h^2 \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \vdots \\ f_{N-2} \\ f_{N-1} \end{pmatrix}. \tag{16}$$

By discretising the Poisson equation we could rewrite it as matrix equation with the form

$$A\mathbf{v} = \mathbf{g}, \tag{17}$$

where A is a tridiagonal matrix, $\mathbf{v}$ is a vector of the unknown v_i's and $\mathbf{g}$ encodes the information of the source term, but is scaled by a factor h^2.

PROBLEM 5

The vector $\mathbf{v}$, as defined above, includes only the subset of internal points of v, which are unknown. The complete solution $\mathbf{v}^*$, with length m, also includes the two endpoints, v_0 and v_N, which are determined by the boundary conditions. That means

$$m = n + 2 = N + 1. \tag{18}$$

PROBLEM 6

Problem a

We want an algorithm to solve the general matrix equation

$$A\mathbf{v} = \mathbf{g}, \tag{19}$$

where A is a $n \times n$ tridiagonal matrix. We denote the subdiagonal, main diagonal and superdiagonal as $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$, respectively. The vectors $\mathbf{v}$, $\mathbf{g}$ and $\mathbf{b}$ has length n, and the off-diagonal vectors $\mathbf{a}$ and $\mathbf{c}$ has length $n-1$.

To solve Equation 19 we simply us Gaussian elimination. The algorithm is separated in forward and backward substitution. In the forward substitution, the subdiagonal $\mathbf{a}$ is eliminated, while $\mathbf{b}$ and $\mathbf{g}$ are modified. In the backward substitution, we solve each row for v_i, one after the other. The exact steps are presented in Algorithm 1.

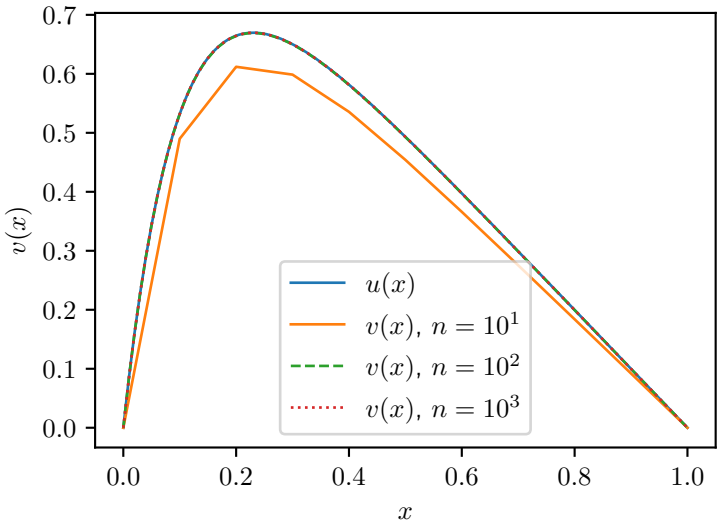

FIG. 2. The plot shows the exact solution $u(x)$ to the Poisson equation and the numerical approximation $v(x)$ for three different stepsizes.

Algorithm 1 Gaussian elimination of a tridiagonal matrix

Initialise **a**, **b**, **c** and **g** with their given values
Initialise new vector **v** of length n

▷ Forward substitution
for $i = 1, 2, \ldots, n-1$ **do** ▷ $n-1$ iterations
  $d = a_i / b_i$ ▷ 1 FLOP
  $b_{i+1} = b_{i+1} - d\, c_i$ ▷ 2 FLOPs
  $g_{i+1} = g_{i+1} - d\, g_i$ ▷ 2 FLOPs
▷ Backward substitution
$v_n = g_n / b_n$ ▷ 1 FLOP
for $i = 1, 2, \ldots, n-1$ **do** ▷ $n-1$ iterations
  $v_{n-i} = (g_{n-i} - c_{n-i} v_{n-i+1}) / b_{n-i}$ ▷ 3 FLOPs

Problem b

In the right coloumn of Algorithm 1 we have listed the number of floating-point operations (FLOPs) needed for each step. We see that there are $5(n-1)$ FLOPs needed for the forward substitution and $3(n-1)+1$ for backwards substitution. When we sum these, we find that the algorithm needs

$$(8n - 7) \text{ FLOPs}. \tag{20}$$

We note that the number of FLOPs depends linearly on n.

PROBLEM 7

The alogrithm is implemented in the function `genAlgo`, found in the file `algo.cpp` in the repo. The resulting plot is found in Figure 2.

PROBLEM 8

The plots of the absolute error

$$\Delta_i = |u_i - v_i| \tag{21}$$

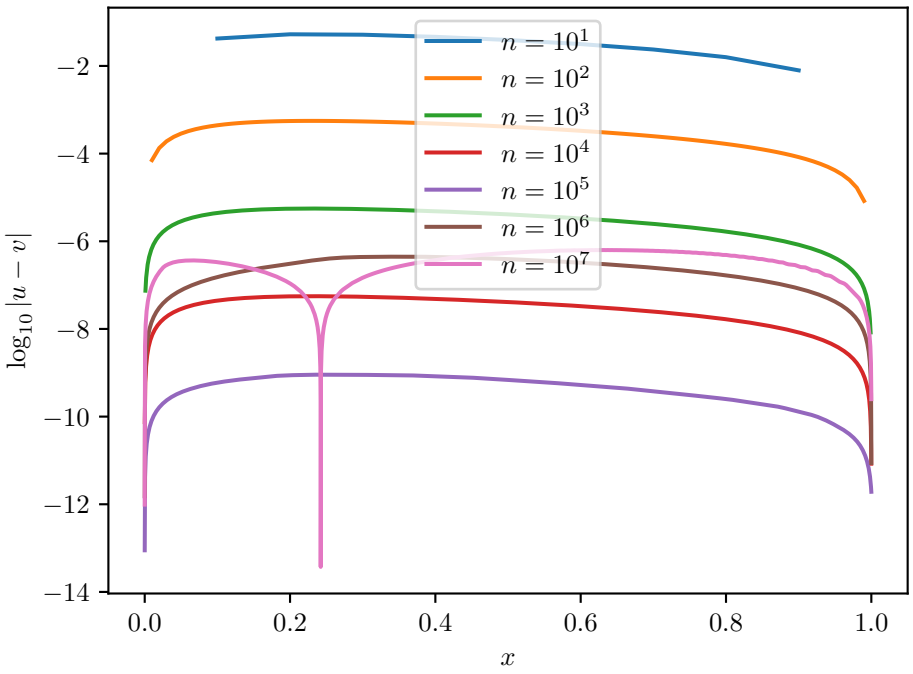

FIG. 3. The plot shows the absolute error of the numerical approximation $v(x)$ for different numbers of steps.

TABLE I. The table shows the maximal relative error in $v(x_i)$, $\max(\epsilon_i)$, for different number of steps, n_{steps}, used in the numerical approximation.

n_{steps}	$\max(\epsilon_i)$
10	$3.3 \cdot 10^{-1}$
10^2	$4.6 \cdot 10^{-2}$
10^3	$6.2 \cdot 10^{-3}$
10^4	$8.5 \cdot 10^{-4}$
10^5	$1.2 \cdot 10^{-3}$
10^6	$6.4 \cdot 10^{-3}$
10^7	$7.4 \cdot 10^{-2}$

and relative error

$$\epsilon_i = \left| \frac{u_i - v_i}{v_i} \right| \tag{22}$$

of the numerical approximation $v(x)$ is shown in Figures 3 and 4, respectively. We notice that the errors first decrease with the stepsize $h \sim 1/n_{\text{steps}}$, until it is around $10^{-4} - 10^{-5}$. When we reduce the stepsize further, we notice that the errors increase again. This is evident when we compare the maximal relative error, $\max(\epsilon_i)$, for different number of steps in Table I. The initial improvement in precision comes from the fact that the discretised algorithm gets more precise for smaller stepsizes, as shown in Equation 11. The problem we face for smaller stepsizes comes from the fact that the computer cannot deal with numbers of arbitrary precision. In the last step of the algorithm, we add two numbers, v_i and g_i, which are of widely different magnitude (g_i is of order h^2). So as h gets smaller, the presicion of g_i is reduced, when added to v_i.

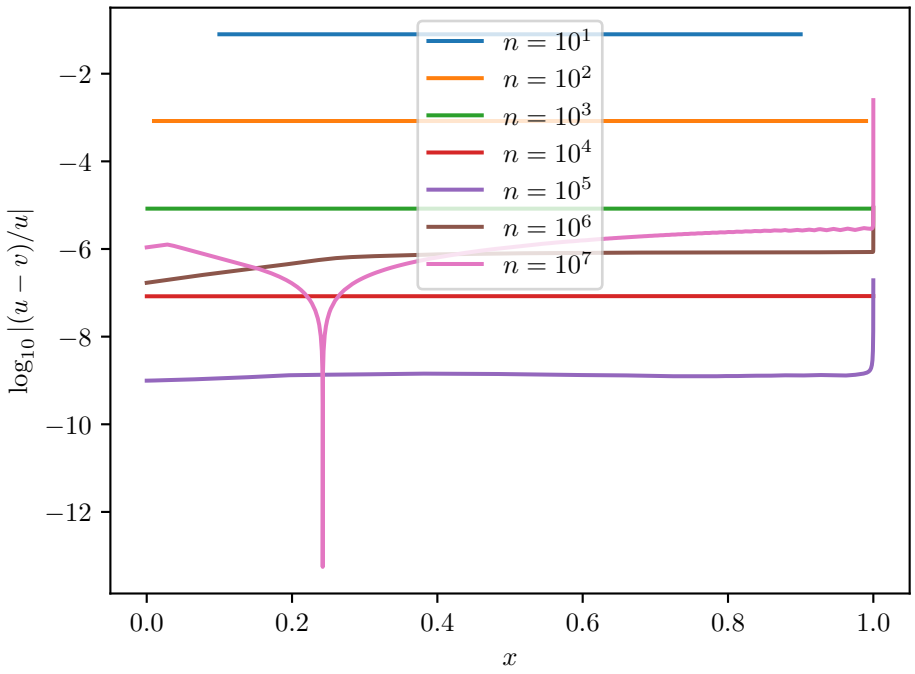

FIG. 4. The plot shows the relative error of the numerical approximation $v(x)$ for different numbers of steps.

PROBLEM 9

The algorithm presented in Problem 6 can be specialised for the purpose of the discretised Poisson equation (Eq. 16). As we know that $a_i = c_i = -1$ and $b_i = 2$, Algorithm 1 can be simplified a bit. The result is shown in Algorithm 2. The specialised algorithm has

$$(6n - 5) \text{ FLOPs}. \tag{23}$$

It is implemented as the function `specAlgo` in `algo.cpp`.

Algorithm 2 Gaussian elimination of Poisson matrix

Initialise **g** with its given values
Initialise new vectors **v** and **b** of length n

▷ Forward substitution

$b_1 = 2$
for $i = 1, 2, \ldots, n-1$ **do** ▷ $(n-1)$ iterations
  $b_{i+1} = 2 - 1/b_i$ ▷ 2 FLOPs
  $g_{i+1} = g_{i+1} + g_i/b_i$ ▷ 2 FLOPs

▷ Backward substitution

$v_n = g_n/b_n$ ▷ 1 FLOP
for $i = 1, 2, \ldots, n-1$ **do** ▷ $(n-1)$ iterations
  $v_{n-i} = (g_{n-i} + v_{n-i+1})/b_{n-i}$ ▷ 2 FLOPs

PROBLEM 10

Table II shows the time used for 100 runs of the algorithms. From the number of FLOPs for each of the two algorithms (Eq. 20 anf 23) we would expect an relative factor

$$t_{\text{gen}}/t_{\text{spec}} \sim \frac{4}{3} \tag{24}$$

TABLE II. The table shows the time used, in seconds, for 100 runs of the general and special algorithms, for different numbers of steps. The last coloumn shown the time factor between the two algorithms.

n_{steps}	t_{gen} (s)	t_{spec} (s)	$t_{\text{gen}}/t_{\text{spec}}$
10	$4.62 \cdot 10^{-4}$	$2.54 \cdot 10^{-4}$	1.82
10^2	$2.44 \cdot 10^{-3}$	$1.91 \cdot 10^{-3}$	1.28
10^3	$2.12 \cdot 10^{-2}$	$1.77 \cdot 10^{-2}$	1.20
10^4	$2.27 \cdot 10^{-1}$	$1.20 \cdot 10^{-1}$	1.15
10^5	1.93	1.57	1.23
10^6	28.7	21.4	1.34

between the two algorithms. From our measurements we see that the relative time factor fits quite well with the expectation.