FYS 3150 Project 1

Amund Bremer
(Dated: September 13, 2022)

Github browser-link to this repository: https://github.com/zmbdr/FYS3150/tree/main/project1

Problem 1

Given

$$u(x) = 1 - (1 - e^{-10})x - e^{-10x},$$

we find that

$$u'(x) = (1 - e^{-10}) - (-10)e^{-10x},$$

and

$$u''(x) = -100e^{-10x}.$$

With the supplied source term $f(c)$ given by

$$f(x) = 100e^{-10x},$$

we get

$$u''(x) = -f(x).$$

The above is valid for $x \in [0, 1]$, and the boundary conditions $u(0) = u(1) = 0$ are satisfied. Hence, $u(x)$ is an exact solution to the given problem.

Problem 2

A simple c++ script to dump $u(x_i)$ for $x = ih$ is supplied in the file `main_2.cpp`. The plot in Figure 1 is generated by the file `plot_2.py`.

Problem 3

It is well established that an approximation to $u''(x)$ is given as follows:

$$u''(x) \approx= \frac{u(x+h) - 2u(x) + u(x-h)}{h^2},$$

where h is an appropriately chosen small step-size. For the problem at hand, we are interested in approximating u'' over the unit interval. We define $x_i = i/N$, discretizing the unit interval into N non-overlapping intervals, and let $i = 0 \ldots N$. Furthermore, we introduce $u(x_i) = u_i$, and $f(x_i) = f_i$.

Thus, from the original Poisson equation, we can write

$$-\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \text{higher order terms} \;=\; f_i, \quad i = 1, \ldots N-1.$$

This leads to the simple discrete approximation v_i:

$$-(v_{i+1} - 2v_i + v_{i-1}) = h^2 f_i, \quad i = 1, \ldots N-1.$$

Here, $v_0 = u_0 = u(0) = 0$, $v_N = u_N = u(1) = 0$, and $h = 1/N$. This is the discrete version of the original Poisson equation, and if this is solvable (it is!), we have found approximation to $u(x)$ on the discrete grid x_i.

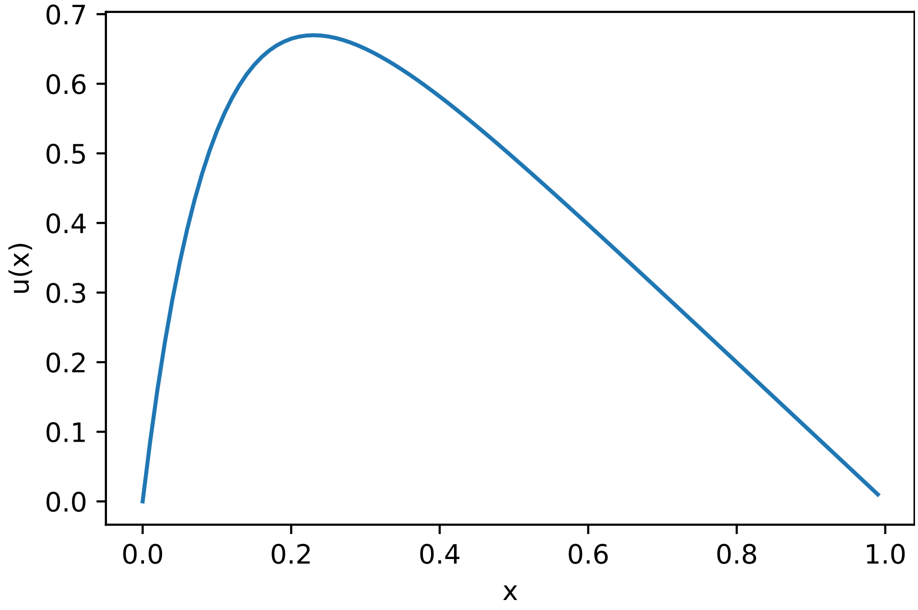

FIG. 1: The exact solution $u(x)$ to the Poisson equation with the given source term over its domain.

Problem 4

The discrete equation above can be written as follows for $i = 2, \ldots, N-2$:

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

However, we need to take care for the cases $i = 1$ and $i = N-1$:

$$\begin{bmatrix} 2 & -1 \end{bmatrix} \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} - v_{i-1} = h^2 f_i, \quad \text{for } i = 1,$$

and,

$$\begin{bmatrix} -1 & 2 \end{bmatrix} \begin{bmatrix} v_{i-1} \\ v_i \end{bmatrix} - v_{N+1} = h^2 f_i, \quad \text{for } i = N-1.$$

Stacking these three equations in matrix form, we get

$$\begin{bmatrix} 2 & -1 & \ldots & 0 & 0 \\ -1 & 2 & -1 & \ldots & 0 \\ 0 & -1 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & \ddots & -1 & 2 \end{bmatrix}_{(N-1)\times(N-1)} \begin{bmatrix} v_1 \\ \vdots \\ \vdots \\ \vdots \\ v_{N-1} \end{bmatrix}_{(N-1)\times 1} - \begin{bmatrix} v_0 \\ 0 \\ \vdots \\ 0 \\ v_N \end{bmatrix}_{(N-1)\times 1} = h^2 \begin{bmatrix} f_1 \\ \vdots \\ \vdots \\ \vdots \\ f_{N-1} \end{bmatrix}_{(N-1)\times 1}.$$

This is what we want; we get

$$\mathbf{A}\vec{v} = \vec{g}, \tag{1}$$

where

$$g_i = \begin{cases} h^2 f_i + v_0 & \text{for } i = 1 \\ h^2 f_i & \text{for } i = 2 \ldots N-2 \\ h^2 f_i + v_N & \text{for } i = N-1 \end{cases} \tag{2}$$

Problem 5

A. Relation n to m

Now, we let $\vec{v^*}$ be a complete solution of the discretized Poisson equation. If we employ N intervals, the full solution need to be specified for $N+1$ values. Hence, if $\vec{v^*}$ has length m, it is clear that $m = N+1$. As described under Problem 4, the matrix $\mathbf{A}$ has dimensions $(N-1) \times N-1$, which is $n \times n$. Thus, $n = m+2$.

B. Relation $\vec{v^*}$ to $\vec{v}$

From the discussion above, it appears that solving 1 gives the *interior solution* of the discrete Poisson equation, i.e. not including the boundary conditions.

Problem 6

OK, now $\mathbf{A}$ is an $n \times n$ tridiagonal matrix, and we are interested in solving $\mathbf{A}\vec{v} = \vec{g}$ for $\vec{v}$.

a) General algorithm

A general algorithm follows from standard Gaussian elimination forward for eliminating the subdiagonal a, and then solving backwards to find v. This is summarized here for algorithm 1.

Algorithm 1 Thomas algorithm (Gaussian elimination applied to tridiagonal matrices)

Require: $a = [a_2, \dots a_n]\,, b = [b_1, \dots b_n]\,, c = [c_1, \dots c_{n-1}]\,, g = [g_1, \dots g_n]$

Allocate memory $\tilde{b} = \left[\tilde{b}_1, \dots \tilde{b}_n\right], \tilde{g} = [\tilde{g}_1, \dots \tilde{g}_n]\,, v = [v_1, \dots v_n]$

Initialize new diagonal: $\tilde{b}_1 = b_1$

for $i = 2, ..., n$ **do** ▷ Repeated $n-1$

  $\tilde{b}_i = b_i - \frac{a_i}{\tilde{b}_{i-1}} c_i$ ▷ FLOP: 4

  $\tilde{g}_i = g_i - \frac{a_i}{\tilde{b}_{i-1}} \tilde{g}_i$ ▷ FLOP: 4

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

for $i = n-1, ..., 1$ **do** ▷ Repeated $n-1$

  $v_i = \frac{\tilde{g}_i - c_i v_{i+1}}{\tilde{b}_i}$ ▷ FLOP: 4

return v

b) Number of FLOPs

From the algorithm above, we see that the number of FLOPS equals $(n-1) \times (4+4) + 1 + (n-1) \times 4 = 12 \times (n-1) + 1$. In other words, solving a triangular system of n linear equations is requires $\approx 12n$ FLOPs.