Project 1 in FYS3150

Simon Halstensen, Carl Fredrik Nordbø Knutsen, Jan Harald Aasen & Didrik Sten Ingebrigtsen

05.09.2021

In this project we are solving the following equation:

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

We also know that:

- $f(x) = 100e^{-10x}$
- $x \in [0, 1]$
- $u(0) = u(1) = 0$

Exercise 1

I will check that

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

is a solution to (1) by differentiating $u(x)$ twice.

$$\frac{d^2u}{dx^2} = \frac{d}{dx}(\frac{du}{dx}) = \frac{d}{dx}(-(1 - e^{-10}) - (-10)e^{-10x})$$

And since the derivative of a constant is 0, we get that:

$$\frac{d^2u}{dx^2} = \frac{d}{dx}(10e^{-10x}) = -100e^{-10x}$$

It immediately follows that

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

This shows that (2) is a solution to equation (1). This solution also satisfies the boundary conditions specified, as:

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

and

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

Exercise 2

The program main.cpp evaluates the exact function $u(x)$ from exercise 1, at points between 0 and 1. It writes the x-values and $u(x)$-values to a .csv-file, named exact_evaluated.csv. The python script read_file_and_plot.py reads the values from the .csv-file, and plots the function (see figure 1).

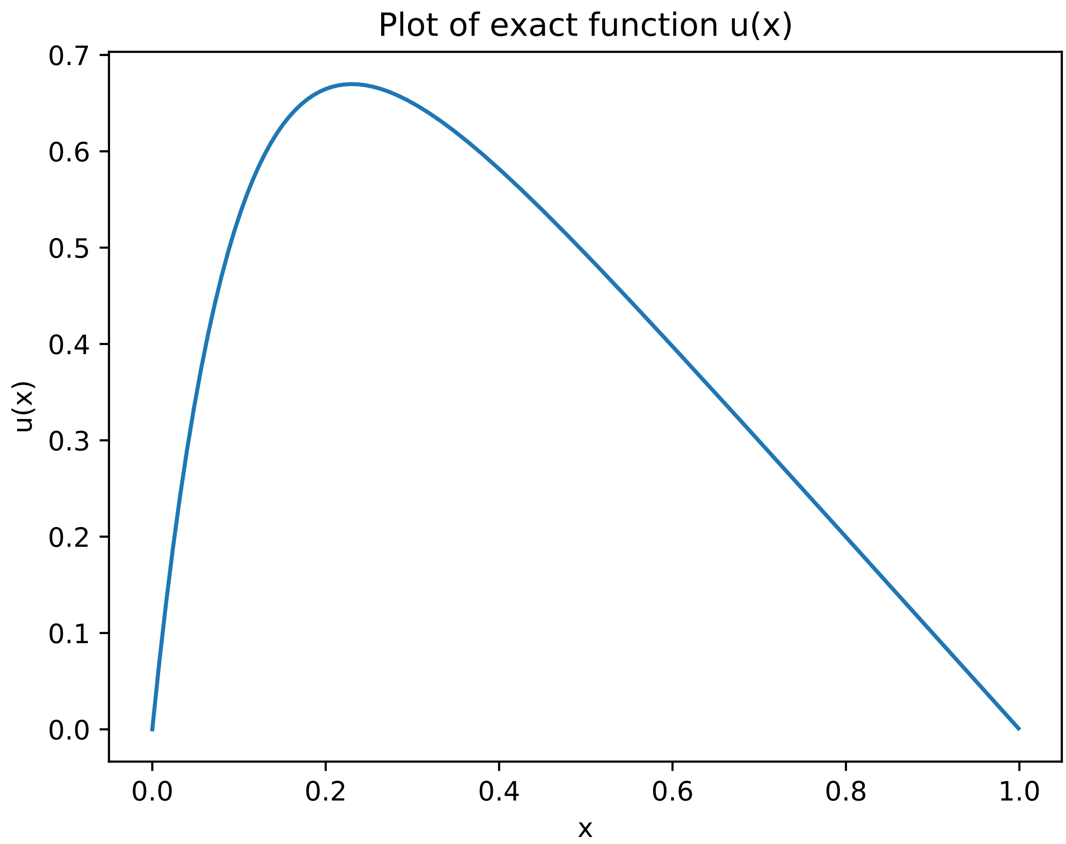

Figure 1: Plot of the exact function $u(x) = 1 - (1 - e^{-10})x - e^{-10x}$

Exercise 3

I will derive a discretized version of equation (1) by finding a discretized approximation of $\dfrac{d^2u}{dx^2} = u''(x)$. Let h be a step size, and let a be a point such that $a \in [h, 1-h]$. Firstly, evaluate the 3rd degree Taylor expansion of u(x) about the point a in the points

$a+h$ and $a-h$.

$$u(a+h) = u(a) + u'(a)\cdot h + \frac{1}{2}u''(a)\cdot h^2 + \frac{1}{6}u'''(a)\cdot h^3 + \mathcal{O}(h^4)$$

$$u(a-h) = u(a) + u'(a)\cdot (-h) + \frac{1}{2}u''(a)\cdot h^2 + \frac{1}{6}u'''(a)\cdot (-h)^3 + \mathcal{O}(h^4)$$

Next, add the two equations, giving the following equality.

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

The equation can be solved for $u''(a)$

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

Assuming a sufficiently small value for h, we can approximate and discretize with $u(ih) \approx v_i$. Here, $i \in \{0, 1, .., n\}$ (meaning $n = \frac{1}{h}$), and:

$$u''(ih) = \frac{v_{i+1} - 2v_i + v_{i-1}}{h^2}$$

Using equation (1), we can rewrite:

$$h^2 \cdot f(ih) = -v_{i+1} + 2v_i - v_{i-1} \tag{3}$$

Which is a discretized version of equation (1) with the following conditions:

- $v_0 = u(0) = 0$
- $v_n = u(1) = 0$.

Exercise 4

We will show that you can write the discretized equation as a matrix equation:

$$\boldsymbol{A}\vec{v} = \vec{g}$$

We have eq. (3) from exercise 3, with $i = 1, 2, \ldots, n$.

$$\begin{array}{cccccccc}
-v_0 & 2v_1 & -v_2 & & & & & \\
 & -v_1 & 2v_2 & -v_3 & & & & \\
 & & -v_2 & 2v_3 & -v_4 & & & \\
 & & \cdots & \cdots & \cdots & \cdots & \cdots & \\
 & & & & -v_{i-3} & 2v_{n-2} & -v_{n-1} & \\
 & & & & & -v_{n-2} & 2v_{n-1} & -v_n
\end{array}
= h^2
\begin{array}{c}
f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-2} \\ f_{n-1}
\end{array}$$

we know $v_0$and v_n

$$\begin{matrix} 2v_1 & -v_2 & & & & \\ -v_1 & 2v_1 & -v_3 & & & \\ & -v_2 & 2v_1 & -v_4 & & \\ & \dots & \dots & \dots & \dots & \dots \\ & & & -v_{n-3} & 2v_{n-2} & -v_{n-1} \\ & & & & -v_{n-2} & 2v_{n-1} \end{matrix} = h^2 \begin{matrix} f_1 + v_0 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-2} \\ f_{n-1} + v_n \end{matrix} \equiv \begin{matrix} g_1 \\ g_2 \\ g_3 \\ \vdots \\ g_{n-2} \\ g_{n-1} \end{matrix}$$

We can then seperate out $\vec{v}$

$$\begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & \dots & \dots & \dots & \dots & \dots \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-2} \\ v_{n-1} \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ \vdots \\ g_{n-2} \\ g_{n-1} \end{bmatrix}$$

We now have a known A and $\bar{g}$, so we can solve for $\bar{v}$.

Exercise 5

a)

When the matrix equation in exercise 4 is solved, you get approximate solutions for all the inner points x_i on the interval $(0, 1$. That is, you get solutions for v for all points except the first point and the last point. Therefore, if A is a $n \times n$-matrix, the complete solution $\vec{v}^*$ must be of length $m = n + 2$, when you include v_0 and v_{n+1} (note that we use a different notation in this problem, with n being the number of intervals and not the number of points).

b)

The n equations give solutions v_i for all the inner points x_i, as explained in exercise a). We do not solve for v_{n+1} or v_0, as these values are already known (and necessary to compute values of v_i for $i \in \{1, 2, ..., n\}$ with this method).

Exercise 6

a)

In this exercise, we want to formulate the algorithm for solving $Ax = g$ for a general tridiagonal A. This is done in [alg 1].

Algorithm 1 Algorithm for solving $Ax = g$ for a general tridiagonal matrix A. a, b and c represent the sub-, main- and superdiagonal. Solving it means taking in A and g, and returning x.

procedure TRIDIAGONAL SOLVER(a, b, c, g, N)
  $\tilde{b}_0 \leftarrow b_0$
  $\tilde{g}_0 \leftarrow g_0$
  for $i \in (1, N)_{\mathbb{N}}$ **do**
    $\tilde{b}_i \leftarrow b_i - \frac{a_i}{\tilde{b}_{i-1}} c_{i-1}$
    $\tilde{g}_i \leftarrow g_i - \frac{a_i}{\tilde{b}_{i-1}} \tilde{g}_{i-1}$
  end for
  $x_N \leftarrow \frac{\tilde{g}_N}{\tilde{b}_N}$
  for $i \in (N-1, 0)_{\mathbb{N}}$ **do**
    $x_i \leftarrow \frac{\tilde{g}_i - c_i x_{i+1}}{\tilde{b}_i}$
  end for
  return x
end procedure

b)

The number of floating point operations (FLOPs) in the general algorithm in [alg 1] is $2 \cdot 3N = 6N$, where N is the size of the matrix, for forward substitution. For back substitution, we have $3N$ FLOPs. In total, the algorithm has $9N = \mathcal{O}(N)$ FLOPs.

Exercise 7

To be added

Exercise 8

To be added

Exercise 9

To be added

Exercise 10

To be added

Exercise 11

To be added

The number of floating point operations (FLOPS) in the general algorithm in [alg 1] is $2 \cdot 3N = 6N$, where N is the size of the matrix, for forward substitution. For back substitution, we have $3N$ FLOPS. In total, the algorithm has $9N = \mathcal{O}(N)$ FLOPS.