Project 1

Laila Egbeocha Andersland
(Dated: 13. september 2021)

https://github.com/laila503/fys3150

PROBLEM 1

We have the one-dimentional Poisson equation:

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

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

We solve this by inserting the function for $f(x)$ and integrate on both side of the equation:

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

$$\frac{du}{dx} = \underbrace{-\int 100e^{-10x}dx}_{(*)} + c_1$$

We use substistusion: $u = -10x \rightarrow du/dx = -10 \rightarrow dx = -du/10$, so that (*) becomes:

$$-100\int e^u(-\frac{du}{10}) = 10\int e^u du = 10e^u + c_2 = 10e^{-10x} + c_2$$

We put this back into the equation:

$$\frac{du}{dx} = 10e^{-10x} + c_1 + c_2$$

We can integrate this equation again:

$$\int \frac{du}{dx}dx = \int \underbrace{10e^{-10x}}_{**} + c_1 + c_2 dx$$

At (**) we can use the same substitution as in (*), and we get:

$$u(x) = -e^{-10x} + c_1x + c_2x + c_3 + c_4$$

$$u(x) = -e^{-10x} + x\underbrace{(c_1 + c_2)}_{\equiv \mathbf{b_1}} + \underbrace{c_3 + c_4}_{\equiv \mathbf{b_2}}$$

$$u(x) = -e^{-10x} + b_1x + b_2 \tag{2}$$

With $u(0) = 0$ eq.(2) becomes:

$$-e^{-10x\cdot 0} + b_1 \cdot 0 + b_2 = 0$$

$$-e^0 + b_2 = 0$$

$$-1 + b_2 = 0$$

$$b_2 = 1 \tag{3}$$

With $u(1) = 0$ eq.(2) becomes:

$$-e^{-10\cdot 1} + b_1 \cdot 1 + b_2 = 0$$

$$-e^{-10} + b_1 + b_2 = 0$$

$$b_1 = e^{-10} - b_2$$

Use eq. (3):

$$b_1 = e^{-10} - 1$$

We can now put this back in eq. (2):

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

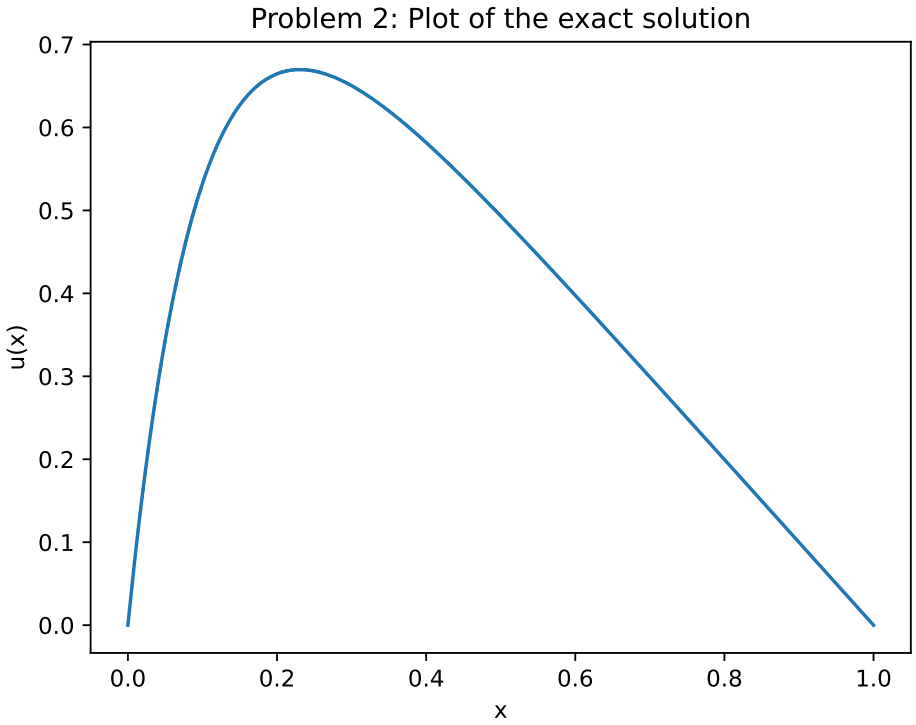
Rearrange to get it on the form represented in the problem:

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

PROBLEM 2

a

See code in *main.cpp* and *plot.py*.



Figur 1. Plot of the exact solution, u(x).

PROBLEM 3

To discretize the Poisson equation means to transfere the contious function into a discretized version. This is of course valuable when working with scientific programming, since a nummerical representation of fuctions will always be discrete.

To derive a discretized version of eq. (1), we introduce the notation for the numerical representation of the variable and functions; $x \to x_i$, $u(x) \to u_i$, and $f(x) \to f_i$ where $i = 0, 1, 2, ..., n$. We use the approximation of the second derivative [1]:

$$f'' = \frac{f_h - 2f_0 + f_{-h}}{h^2} + \mathcal{O}(h^2)$$

We applied this to our function and replace the differential operator, so that we get the discertized version of the second derivative:

$$u_i'' = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \mathcal{O}(h^2)$$

We put this in eq.(1):

$$-\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \mathcal{O}(h^2) = f_i$$

The error part, $\mathcal{O}(h^2)$, makes this function an exact representation, but since we can't find this error, we will have an approximation of the function which we call v_i instead of u_i. We multiply everything with h^2 and get:

$$-v_{i-1} + 2v_i - v_{i+1} = h^2 f_i \tag{5}$$

PROBLEM 4

To show that eq.(5) can be written as a matrix equation on the form:

$$\mathbf{A}\vec{v} = \vec{g} \tag{6}$$

We check eq.(5) for $n = 5$ steps, 6 points: $v_0, v_1, v_2, v_3, v_4, v_5$, where the boundary points $v_0, v_5 = 0$. After putting inn $i = 1, 2, 3$ and 4, we get an set of equations, and on the rightside of the equation, we can rename $h^2 f_i = g_i$. This can be represenseded as:

$$\underbrace{\begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix}}_{\vec{v}} = \underbrace{\begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \end{pmatrix}}_{\vec{g}} \tag{7}$$

We can see that the matix $\mathbf{A}$ contains the number 2 on the maindiagonal, and -1 on the sub -and super-diagonal, and 0 on the rest. In other words: $\mathbf{A}$ is a tridiagonal matrix with the subdiagonal, main diagonal and superdiagonal specified by the signature $(-1, 2, -1)$. This matrix equation holds for n abitrary number of steps.

PROBLEM 5

We will now let $\vec{v}^*$ vector represent the complete solution of the discretisized Poisson equation.

a

Since we excluded the boundary points when we derived eq.(7) , the complete solution, $\vec{v}^*$, would include the first and the n'th element, so that $m = n + 2$.

b

When we solve eq.(7) for v we find the solution for $i = 1, 2, \ldots, n-1$ of the complete solution (excluding $i = 0$ and $i = n$).

PROBLEM 6

a

We can solve the general tridiogonal case of $\mathbf{A}\vec{v} = \vec{g}$, by renaming the values, so that the maindiagonal: $(2, 2, 2, 2) = (b_1, b_2, b_3, b_4)$, superdiagonal: $(-1, -1, -1) = (c_1, c_2, c_3)$, and the subdiagonal is: $(-1, -1, -1) = (a_2, a_3, a_4)$:

$$\begin{array}{c|cccc|c} R_1 & b_1 & c_1 & 0 & 0 & g_1 \\ R_2 & a_2 & b_2 & c_2 & 0 & g_2 \\ R_3 & 0 & a_3 & b_3 & c_3 & g_3 \\ R_4 & 0 & 0 & a_4 & b_4 & g_4 \end{array}$$

To solve this we use Gaussian elimination, where we first use forward substituation and elimination to get the matrix on an upper triangular form. After some row reduction and renaming of variables, we get:

$$\begin{array}{cccc|c} \widetilde{b_1} & c_1 & 0 & 0 & \widetilde{g_1} \\ 0 & \widetilde{b_2} & c_2 & 0 & \widetilde{g_2} \\ 0 & 0 & \widetilde{b_3} & c_3 & \widetilde{g_3} \\ 0 & 0 & 0 & b_4 & \widetilde{g_4} \end{array}$$

Where

$$\widetilde{b_1} = b_1$$

$$\widetilde{b_i} = b_i - \frac{a_i}{\widetilde{b_{i-1}}} c_{i-1} \tag{8}$$

$$\widetilde{g_1} = g_1$$

$$\widetilde{g_i} = g_i - \frac{a_i}{\widetilde{b_{i-1}}} \widetilde{g}_{i-1} \tag{9}$$

For $i = 2, 3, 4$. After this we can now do the back substitation, and we get:

$$v_4 = \frac{\widetilde{g_4}}{\widetilde{b_4}}$$

$$v_i = \frac{\widetilde{g}_i - c_i v_{i+1}}{\widetilde{b}_i} \tag{10}$$

For $i = 3, 2, 1$. To generalize this we let i go from 2 to n. This is the *general algorithm* for the general tridiagonal matrix:

Algorithm 1 General algorithm

$\widetilde{b_1} = b_1$
$\widetilde{g_1} = g_1$
for $i = 2, 3, ..., n$ **do**
  $\widetilde{b_i} = b_i - \frac{a_i}{\widetilde{b_{i-1}}} c_{i-1}$ and $\widetilde{g_i} = g_i - \frac{a_i}{\widetilde{b_{i-1}}} g_{i-1}$
end for
$v_n = \frac{\widetilde{g_n}}{\widetilde{b_n}}$
for $i = n-1, n, ..., 2$ **do**
  $v_i = \frac{\widetilde{g_i} - c_i v_{i+1}}{\widetilde{b_i}}$
end for

b

The general algorithm consist of eq.(8), (9) and (10). To calculate the number of FLOPs, we look at each equation.

For eq.(8) the minus between the first and last term is one FLOP. The division in the last term and the multiplication between $\frac{a_i}{\widetilde{b_{b_{i-1}}}}$ and c_{i-1} gives two FLOPS. The initialcondition $\widetilde{b_1} = b_1$ does not require any FLOPS. For $i = 2, 3, \cdots, n$ We get $3(n-1)$ FLOPs.

For eq.(9) we get $2(n-1)$ FLOPs. The division $\frac{a_i}{\widetilde{b_{b_{i-1}}}}$ has already been done.

For eq.(10) we get $3(n-1)$ FLOPs. But from the initial condition $\widetilde{v_4} = \frac{\widetilde{g_1}}{\widetilde{b_4}}$ we get one extra $+1$.

This gives us:

$$\begin{aligned} & 3(n-1) \\ + \ & 2(n-1) \\ + \ & 3(n-1) \\ + \ & 1 \\ = \ & \underline{\underline{8n-7}} \end{aligned}$$

PROBLEM 7

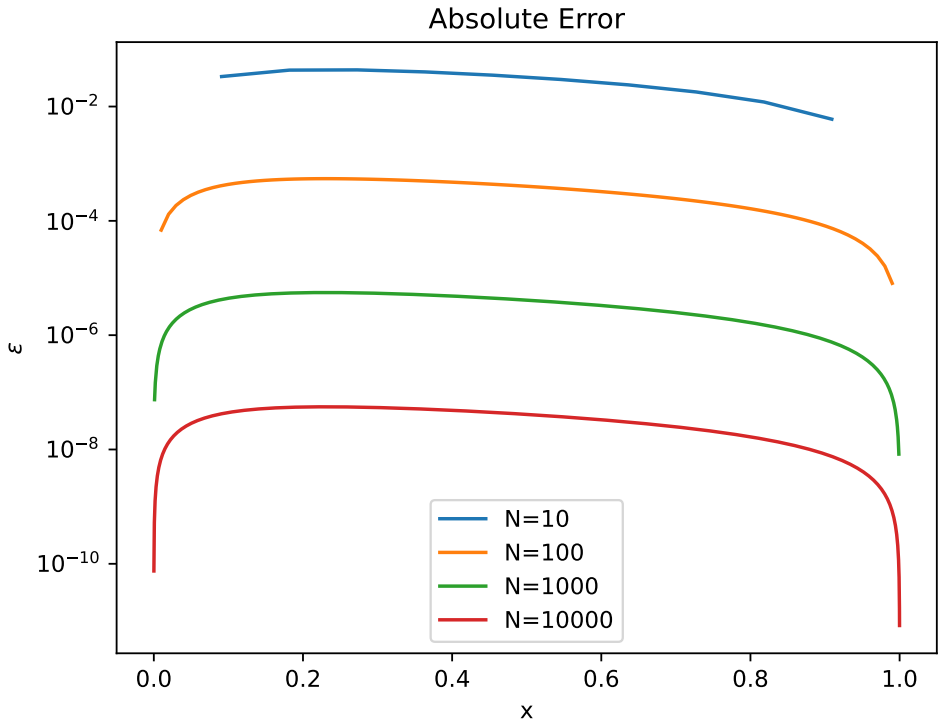
In figure 2 we see the absolute error, and in fig. 3 we see the relative error. Both errors seem to decrease with n, as we expect.

To see this better represented we can plot the maximum of the different relative errors for each n. In fig. 4 we see that the error decreases linearly until n 10^5, where it starts to increase. The initial decreases is because the error goes as $1/n^2$ meaning that the log of the error should decrease linearly with a slope of -2. At n 10^5 we hit the machine precision, meaning that the computations becomes less precise, and the errors increase. To the ideal number of steps is around n^5. This number is dependent on the computer used.

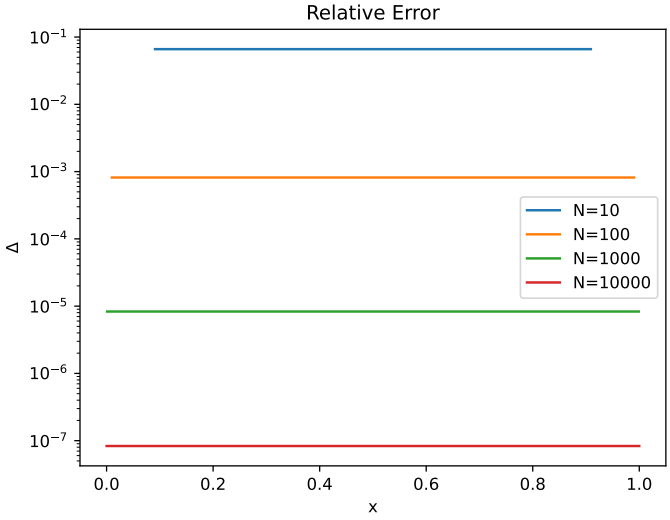
PROBLEM 9

a

We will now specialize the algorithm from problem 6 to a special case of $\mathbf{A}$ specified by the signature $(-1, 2, -1)$.



Figur 2. Plot of the absolute error.

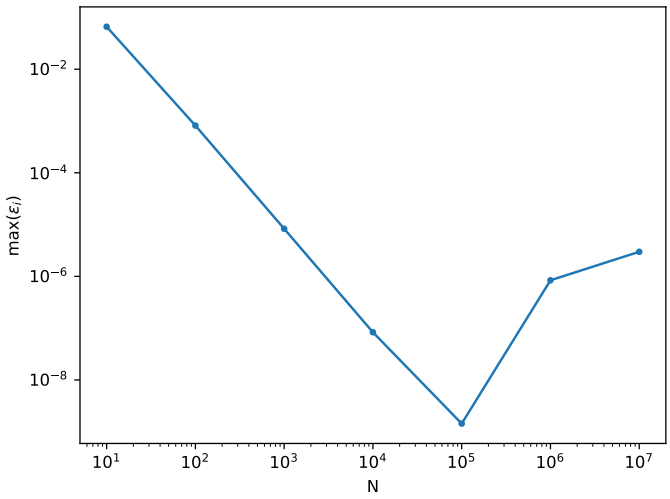


Figur 3. Plot of the relative error.

From eq.(8), eq.(9) and eq.(10)we get:

$$\widetilde{b}_1 = b_1$$

$$\widetilde{b}_i = 2 - \frac{1}{\widetilde{b}_{i-1}} \tag{11}$$



Figur 4. Plot of maximum of the relative error for n from 10 to 10^7.

$$\widetilde{g}_1 = g_1$$

$$\widetilde{g}_i = g_i + \frac{1}{\widetilde{b}_{i-1}}\widetilde{g}_{i-1} \tag{12}$$

$$v_n = \frac{1}{2}\widetilde{g}_n$$

$$v_i = \frac{\widetilde{g}_i + v_{i+1}}{\widetilde{b}_{i-1}} \tag{13}$$

Algorithm 2 Special algorithm

$\widetilde{b_1} = b_1$
$\widetilde{g_1} = g_1$
for $i = 2, 3, ..., n$ **do**
  $\widetilde{b_i} = 2 - \frac{1}{\widetilde{b_{i-1}}}$ and $\widetilde{g_i} = g_i - \frac{1}{2}g_{i-1}$
end for
$v_n = \frac{\widetilde{g}_n}{\widetilde{b}_n}$
for $i = n-1, n, ..., 2$ **do**
  $v_i = \frac{\widetilde{g}_i + v_{i+1}}{\widetilde{b}_{i-1}}$
end for

b

Setting $\widetilde{b}_1 = b_1$ demands 0 FLOPs, while the loop eq.(11) requires $(n-1)$ FLOPS for every division ($\frac{1}{\widetilde{b}_{i-1}}$) and $(n-1)$ FLOPs for every subtraction between the first and second term, giving $2(n-1)$ FLOPs.

In eq.(12) we remember that $g_i = f_i h^2$, so that we have the same amount of FLOPs required for g_n as f_n. The number of FLOPs needed for $\frac{1}{2}\widetilde{g}_{i-1}$ is $(n-1)$ and $g_i - \frac{1}{2}\widetilde{g}_{i-1}$ is another $(n-1)$, giving $2(n-1)$ FLOPs.

From eq.(13) we also get $2(n+1)$ FLOPs. In total we then get:

$$\begin{aligned} & 2(n-1) \\ + \ & 2(n-1) \\ + \ & 2(n-1) \\ = \ & \underline{\underline{6(n-1)}} \end{aligned}$$

Since we saw that the general case demanded $(8n-7)$ FLOPs in problem 6, we observe that we need $(8n-7)-(6n-6)=2n-1$ less FLOPs for this special case.

c

The implementation of the special algorithm can be found in *special.cpp* in the github repository.

PROBLEM 10

Using `<chrono>` to time the general algorithm in our code:

```
Time: 4.3e-06s, for N= 10
Time: 3.2e-06s, for N= 10
Time: 3.2e-06s, for N= 10
Time: 3.1e-06s, for N= 10
Time: 3.2e-06s, for N= 10
```

I'll run the timing of the algorithm for 5 times for each N and calculate the mean general time, which is what is included in the Timing Table.

Timing Table			
N	General time $[s]$	Special time $[s]$	Special time / General time
10	3.980×10^{-6}	2.800×10^{-6}	0.704
100	2.210×10^{-5}	1.252×10^{-5}	0.567
1000	1.134×10^{-4}	7.244×10^{-5}	0.639
10000	8.404×10^{-4}	8.118×10^{-4}	0.966
100000	9.345×10^{-3}	7.723×10^{-3}	0.826
1000000	8.952×10^{-2}	7.273×10^{-2}	0.812

As we see from the last row (Special/General), the special algorithm uses approximately $57-97\%$ of the time used in on the general algorithm. In problem 9 we saw that the special case needed 1/4 less FLOPs needed for the general case. This is in the ballpark of the theoretical expectation, but to draw a more accurate conclusion, more runs for each N is required.

PROBLEM 11

From [2] we see that LU decomposition goes as $\frac{2}{3}n^3$. This means that for larger n, this will be slower than our algorithm. For $n=10^5$ we can expect that the code becomes too slow to run on our personal laptop.

---

[1] Morten Hjorth-Jensen. *Computational Physics*. 2015.
[2] R. Mark Prosser. Lu factorization. https://student.cs.uwaterloo.ca/~cs370/notes/LU.pdf. [Online; accessed 13-September-2021].