Project 1

Maziar Kosarifar and Marie Reichelt

Fall 2018

Source files are located in:
https://github.com/maziark/FYS3150-2018.git

1a.

We have the source term $f(x) = 100e^{-10x}$ and the differential equation given by $u(x) = 1 - (1 - e^{10})x - e^{-10x}$. We want to prove that $-u''(x) = f(x)$:

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

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

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

QED.

When rewriting the equation as a linear set of equations we can see that we have two scenarios. Firstly we address the top and bottom row. Then we address the middle rows. For the top and bottom rows we get:

$$f_1 = \frac{-1+2}{h^2} = \frac{1}{h^2}$$
$$f_n = \frac{-1+2}{h^2} = \frac{1}{h^2}$$

We will have

$$\tilde{b_1} = h^2 * \frac{1}{h^2} = 1$$
$$\tilde{b_n} = h^2 * \frac{1}{h^2} = 1$$

For the middle rows using the equation:

$$f_i = \frac{-1 - 1 + 2}{h^2} = 0, i \in (2, n-1)$$

This gives us:

$$\tilde{bi} = h^2 * 0 = 0$$

1b.

As requested, we made three vectors called a, b and c to represent the non-zero values of the tridiagonal matrix. We have used f_i as the value of $f(x_i)$, and similarly u_i is defined as $u(x_i)$.

The algorithm we have used to simplify the tridiagonal matrix is rather simple, only a two step process. This method will have a linear processing time (O(n)).

In the first step we eliminate the coefficient of a_i and update the values of b and f. b' is used as the new value of b, and f' is used as the updated value of f.

Method used for updating b_i:

$$b'_i = b_i - \frac{a_i * c_{i-1}}{b'_{i-1}}$$

Method for updating f_i:

$$f'_i = f_i - \frac{a_i * f'_{i-1}}{b'_{i-1}}$$

We know there is no element for a in the first row, and therefore we have:

$$b'_1 = b_1$$
$$f'_1 = f_1$$

Now out matrix is shaped like:

$$A = \begin{bmatrix} b'_1 & c_1 & 0 & ... & ... & 0 \\ 0 & b'_2 & c_2 & ... & ... & 0 \\ . & ... & b'_3 & ... & ... & . \\ . & ... & ... & ... & ... & . \\ . & ... & ... & ... & ... & . \\ 0 & 0 & 0 & ... & 0 & b'_n \end{bmatrix}$$

And thus, we can now find the value of U_n with a simple division:

$$u_n = \frac{f'_n}{b'_n}$$

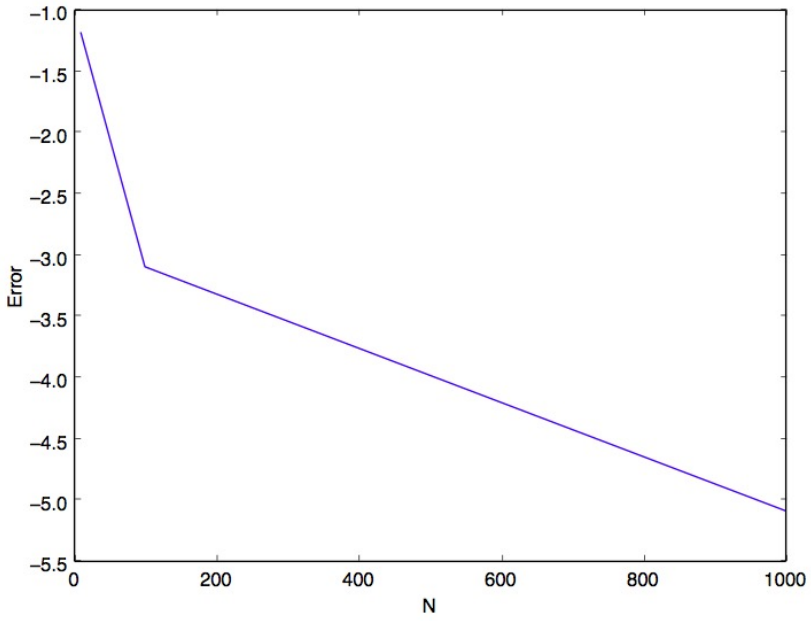
Knowing that value, we can now find the rest using backward substitution:

$$u_i = \frac{f'_i - c_i * u_{i+1}}{f_i}$$

No matter the size of the array, the problem can now be solved using a linear method with 3n floating point operations.

Table 1: Runtime and error

10^N	Runtime (Matrix)	Error
10	0.000022	-1.179698
100	0.000029	-3.088037
1000	0.000233	-5.080052



With increasing n the h gets smaller which results in decrease in error.

1c.

Following the solution of part b, and knowing that in this particular task the values of a, b and c are set. We have a $a = c = -1$ and $b = 2$. Now we can simplify further:

$$b'_i = 2 - \frac{-1 * -1}{b'_{i-1}}$$

Which can be rewritten as:

$$b'_i = 2 - \frac{-1 * -1}{b'_{i-1}} = \frac{i+1}{i}$$

This can be proven by mathematical induction assuming that the base case is n = 1:

$$b_n = \frac{n+1}{n} = \frac{1+1}{1} = 2$$

which is true.

Assuming the equation is true for the case n = k, we have:

$$b_k = \frac{k+1}{k}$$

For n = k + 1 we have:

$$b_{k+1} = 2 - \frac{1}{b_k} = 2 - \frac{k}{k+1} = \frac{2(k+1)-k}{k+1} = \frac{2k+2-k}{k+1} = \frac{k+2}{k+1}$$

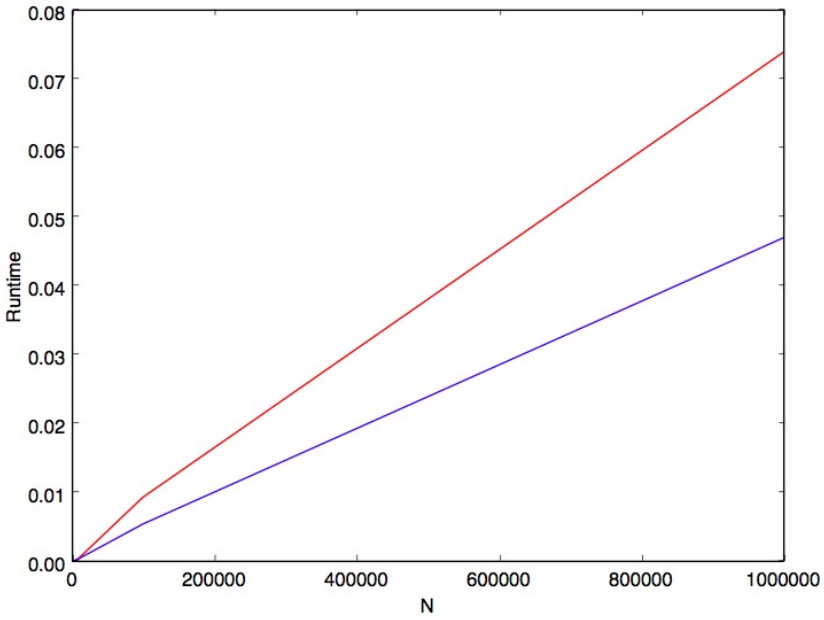
This shows that we can simplify $b'_i = \frac{i+1}{i}$.

Simplifying the equation for finding the value of f'_i:

$$f'_i = f_i - \frac{a_i * f'_{i-1}}{b'_{i-1}} = f_i - \frac{-1 * f'_{i-1}}{\frac{i}{i-1}} = f_i + \frac{(i-1)f'_{i-1}}{i}$$

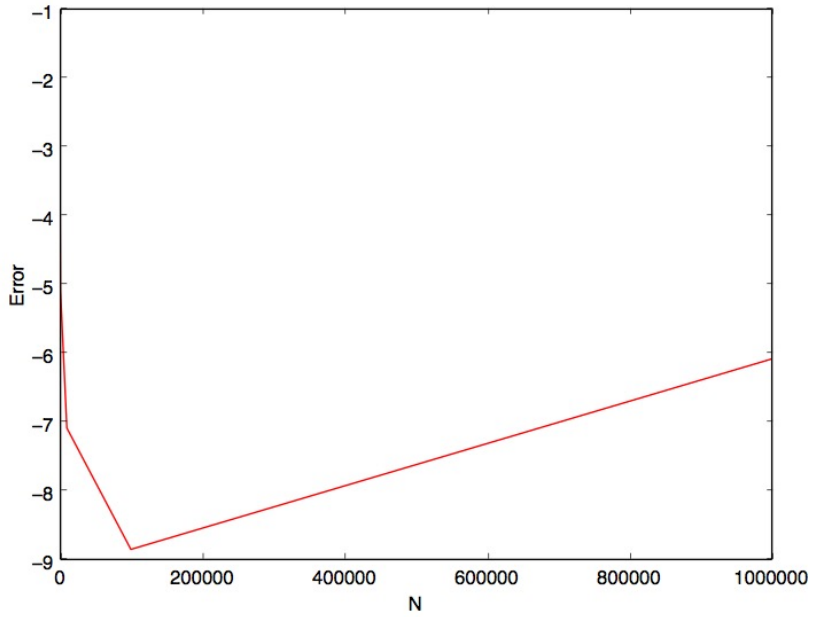
Table 2: Results

10^N	Runtime (Matrix)
1	0.000002
2	0.000008
3	0.000041
4	0.000500
5	0.005527
6	0.047066



With increasing n the runtime increases linearly in both algorithms. It runs faster when we assume that a, b and c have identical values in every row. This is due to loss of precision for round-off error.

1d.



The error decreases to a point before increasing again.

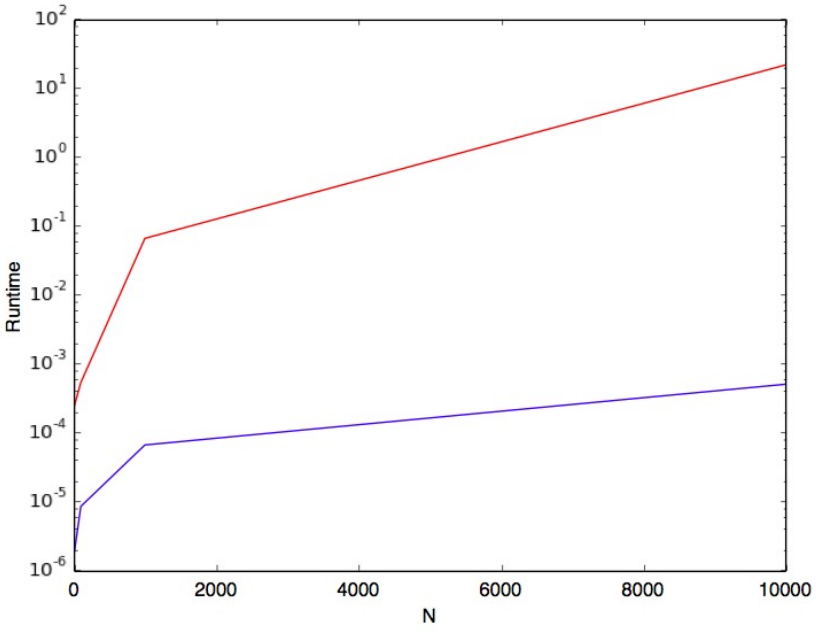
1e.

We cannot run the decomposition of LU for $N = 10^5$ because the algorithm is of $O(n^3)$. This will use too much processing power and also takes up too much memory.

In comparison the algorithm that simplifies the matrix, considering that it is a Tridiagonal matrix, will only grow in $O(n)$, which only requires memory of size n. It can be seen that the relative error is getting proportionately smaller.

Table 3: Comparing results : Optimized V.S. LU

N	Runtime (Matrix)	Error (Matrix)	Runtime (LU)	Error (LU)
10	0.000002	-1.179698	0.000262	-1.179698
100	0.000009	-3.088037	0.000565	-3.088037
1000	0.000070	-5.080052	0.069736	-5.080052
10000	0.000534	-7.079285	22.930153	-7.079285



Red represents the LU decomposition whilst the blue represents the general algorithm.