Project1 Fys3150

Stian Goplen and Håkon Kristiansen

September 8, 2014

Link to github: https://github.com/hakii27/Fys3150

1 Describing the problem

In this project we want to solve the one-dimensional Poisson equation numerically with Dirichlet boundary conditions, and a given source term, using a forward/backward substitution method. Further we want to compare the computed solution to a given exact solution. Then we proceed by investigating the relative error of the computed solution. Finally we compare the computational efficiency of the forward/backward method with Armadillo's Gaussian elimination and LU decomposition.

1.1 Introduction

To be precise we shall solve the following two-point boundary problem.

$$-u''(x) = f(x), \qquad x \in (0,1) \tag{1}$$

$$u(0) = u(1) = 0 \tag{2}$$

$$f(x) = 100e^{-10x}, \qquad x \in (0,1) \tag{3}$$

with the exact solution given by

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

Let v_i denote the approximated solution to $u(x)$ at the grid points $x_i = ih$ in the interval from $x_0 = 0$ to x_{n+1}, with step length $h = \frac{1}{n+1}$.

Let the second derivative be approximated by

$$u''(x_i) \approx \frac{v_{i-1} - 2v_i + v_{i+1}}{h^2} \tag{5}$$

Then the problem is to solve

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i, \quad i = 1, 2, 3, ..., n \tag{6}$$

with $v_0 = v_{n+1} = 0$ on the boundary.

a) First we check that the exact solution satisfies the problem.

Boundary Conditions

$$\begin{aligned} u(0) &= 1 - 1 = 0 \\ u(1) &= 1 - (1 - e^{-10}) - e^{-10} \\ &= 1 - 1 + e^{-10} - e^{-10} = 0 \end{aligned}$$

For $x \in (0,1)$ we have

$$\begin{aligned} u'(x) &= -(1 - e^{-10}) + 10e^{-10x} \\ u''(x) &= -100e^{-10x} \end{aligned} \tag{7}$$

then

$$-u''(x) = 100e^{-10x} = f(x) \tag{8}$$

Hence $u(x)$ is indeed a solution.

We want to show that the equation given by

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i, \quad i = 1,2,3,...,n \tag{9}$$

can be written as $\mathbf{A}\mathbf{v} = \tilde{\mathbf{b}}$.

Rewrite (9) as:

$$\begin{aligned} -v_{i+1} - v_{i-1} + 2v_i &= h^2 f_i \\ -v_{i+1} - v_{i-1} + 2v_i &= \tilde{b_i} \ , \quad \tilde{b_i} = h^2 f_i \end{aligned} \tag{10}$$

Let $\mathbf{v} = \begin{pmatrix} v_1 \\ . \\ . \\ . \\ v_n \end{pmatrix}$ and $\tilde{\mathbf{b}} = \begin{pmatrix} h^2 f_1 \\ . \\ . \\ . \\ h^2 f_n \end{pmatrix}$ and the coefficients of equation $i, i = 1,2,...,n$ is then given by

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 0 & . & . & . & 0 \\ -1 & 2 & -1 & 0 & . & . & 0 \\ 0 & -1 & 2 & -1 & 0 & . & 0 \\ . & . & . & . & . & . & . \\ 0 & . & . & . & 0 & -1 & 2 \end{pmatrix} \tag{11}$$

and (9) can be written as $\mathbf{A}\mathbf{v} = \tilde{\mathbf{b}}$

b) We want to derive an algorithm for solving the following system of equations:

$$\begin{pmatrix} a_1 & c_1 & 0 & . & . & . & 0 \\ b_1 & a_2 & c_2 & 0 & . & . & 0 \\ 0 & b_2 & a_3 & c_3 & 0 & . & 0 \\ . & . & . & . & . & . & . \\ 0 & . & . & . & b_{n-2} & a_{n-1} & c_{n-1} \\ 0 & . & . & . & 0 & b_{n-1} & a_n \end{pmatrix} \begin{pmatrix} v_1 \\ . \\ . \\ . \\ . \\ v_n \end{pmatrix} = \begin{pmatrix} h^2 f_1 \\ \vdots \\ h^2 f_n \end{pmatrix} \tag{12}$$

Let $g_k = h^2 f_k, \quad k = 1, 2, ..., n$ and

$$\tilde{\mathbf{A}} = \begin{pmatrix} a_1 & c_1 & 0 & . & . & . & 0 \\ b_1 & a_2 & c_2 & 0 & . & . & 0 \\ 0 & b_2 & a_3 & c_3 & 0 & . & 0 \\ . & . & . & . & \ddots & . & . \\ 0 & . & . & . & b_{n-2} & a_{n-1} & c_{n-1} \\ 0 & . & . & . & 0 & b_{n-1} & a_n \end{pmatrix} \tag{13}$$

We row reduce $\tilde{\mathbf{A}}$ eliminating $b_1, b_2, ..., b_{n-1}$ and the elements on the diagonal are then given by:

(i) $\tilde{a}_k = a_k - \dfrac{b_{k-1}c_{k-1}}{\tilde{a}_{k-1}}, \quad k = 2, 3, ..., n$

While the right hand side is given by:

(ii) $\tilde{g}_k = g_k - \dfrac{b_{k-1}\tilde{g}_{k-1}}{\tilde{a}_{k-1}}, \quad k = 2, 3, ..., n$

Note that $\tilde{a}_1 = a_1$ and $\tilde{g}_1 = g_1$ Then we have the following system

$$\begin{pmatrix} \tilde{a}_1 & c_1 & 0 & . & . & . & 0 \\ 0 & \tilde{a}_2 & c_2 & 0 & . & . & 0 \\ 0 & 0 & \tilde{a}_3 & c_3 & 0 & . & 0 \\ . & . & . & . & . & . & . \\ 0 & . & . & . & 0 & a_{n-1} & c_{n-1} \\ 0 & . & . & . & 0 & 0 & a_n \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ . \\ . \\ . \\ v_n \end{pmatrix} = \begin{pmatrix} \tilde{g}_1 \\ \tilde{g}_2 \\ . \\ . \\ . \\ \tilde{g}_n \end{pmatrix} \tag{14}$$

To obtain the solution we backsubstitute Step1:

$$v_n = \frac{\tilde{g}_n}{\tilde{a}_n} \tag{15}$$

Then we have

(iii) $v_k = \dfrac{g_k - c_k v_{k+1}}{\tilde{a}_k}, \quad k = n-1, n-2, ..., 1$

Next we want to find the number of floating point operations. (i), (ii) and (iii) gives us one multiplication, one division and one subtraction in each step. In addition, step1 gives one additional operation Under the assumption that all operations take one unit of time, we have that the number of FLOPS is:

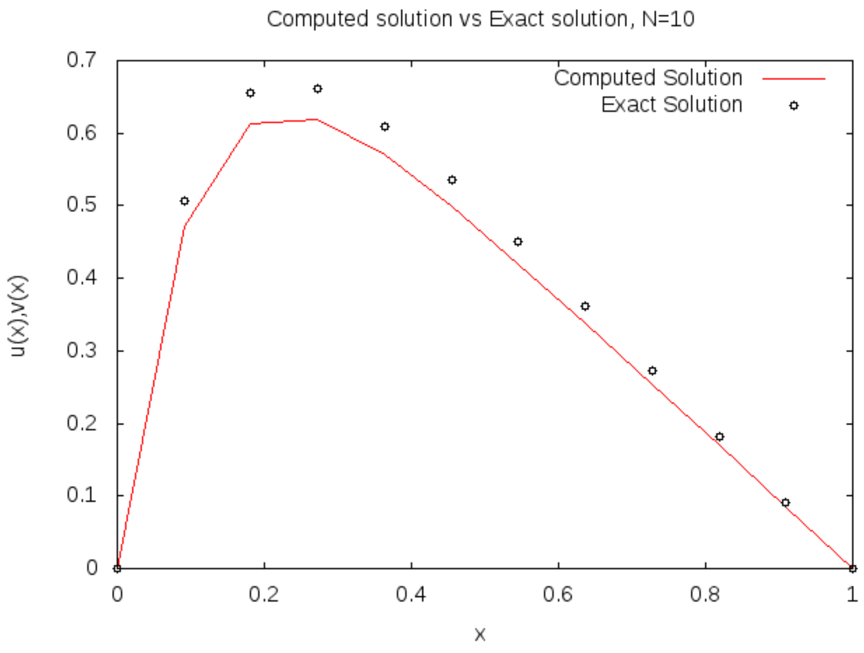
$$3(n-1) + 3(n-1) + 3(n-1) + 1 = 9n - 8 \tag{16}$$

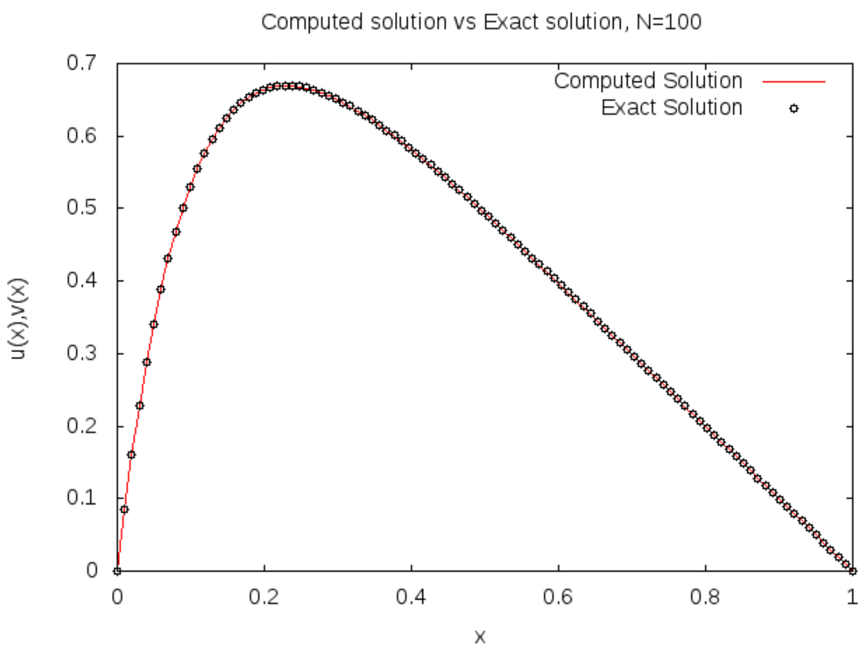
Gaussian elimination takes $\simeq \frac{2}{3}n^3$ of FLOPS, and LU takes $\simeq n^2$ number of FLOPS for solving the system when the matrix is already decomposed. For large n, the number of FLOPS for our algorithm is $\simeq 9n$. Compared to the other two, this is a major improvement. If we exploit the fact that

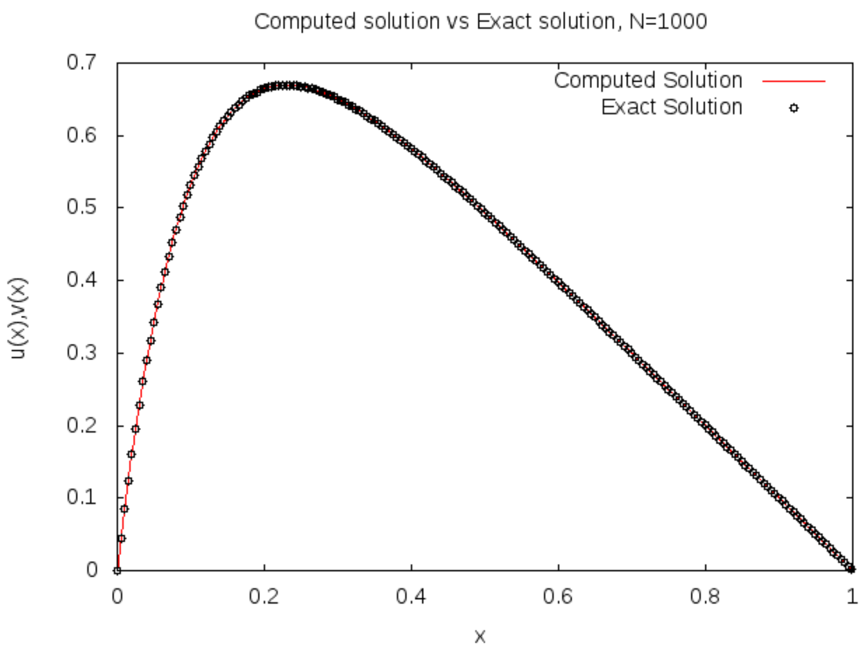
$$b_k = c_k = -1, \quad k = 1, 2, ..., n-1$$

the time complexity is reduced to $\simeq 6n$ for large n in our algoritm.

Solving the system for $n = 10, 100, 1000$, we get the following results:

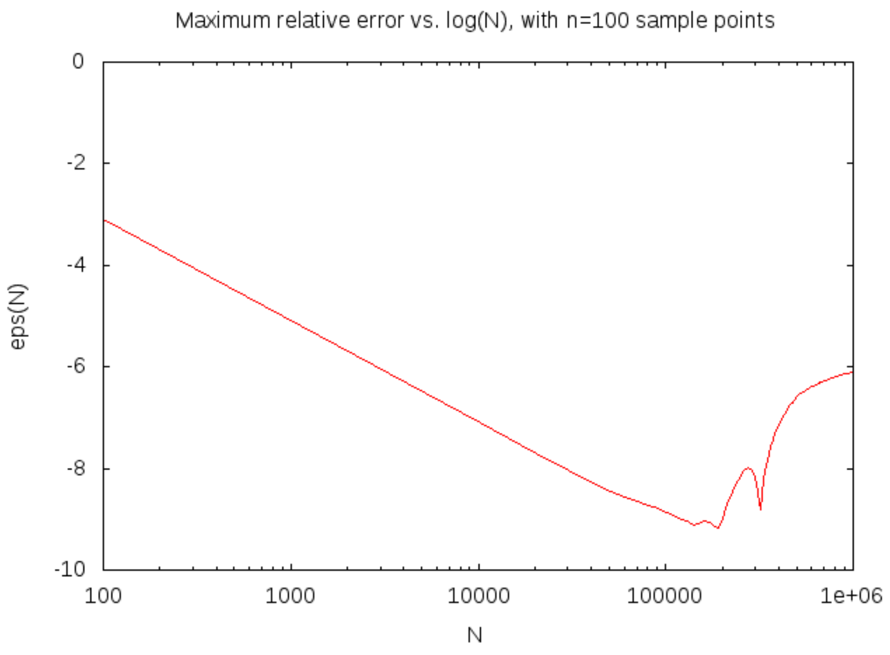






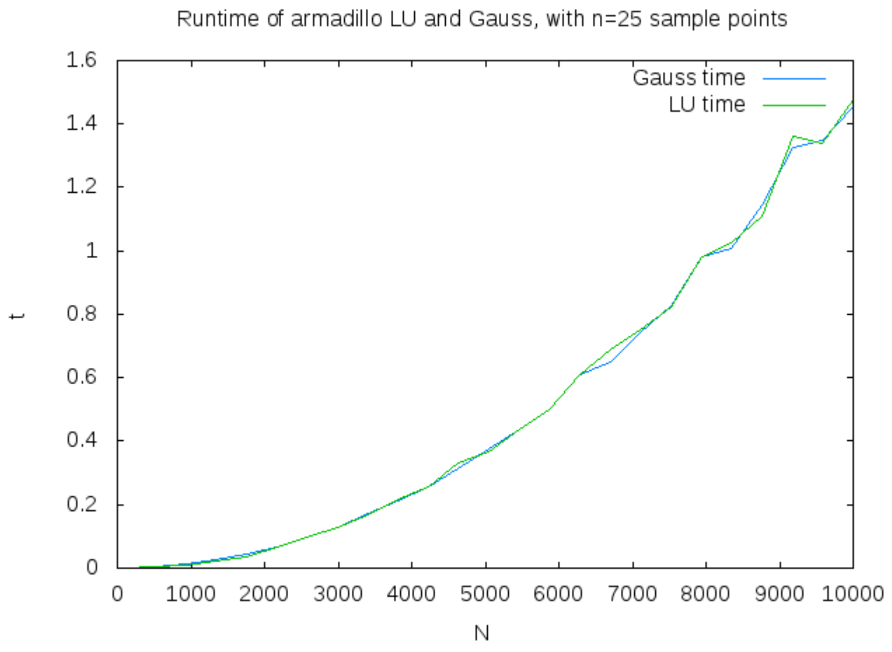
For $n = 10$ we see that the computed solution is not a good approximation to the exact solution, but for $n = 100$ and $n = 1000$ its hard to seperate the two solutions by eye. Note that for $n = 1000$, we have only plotted every 5th point of the exact solution in order to distinguish between the computed solution and the exact solution.

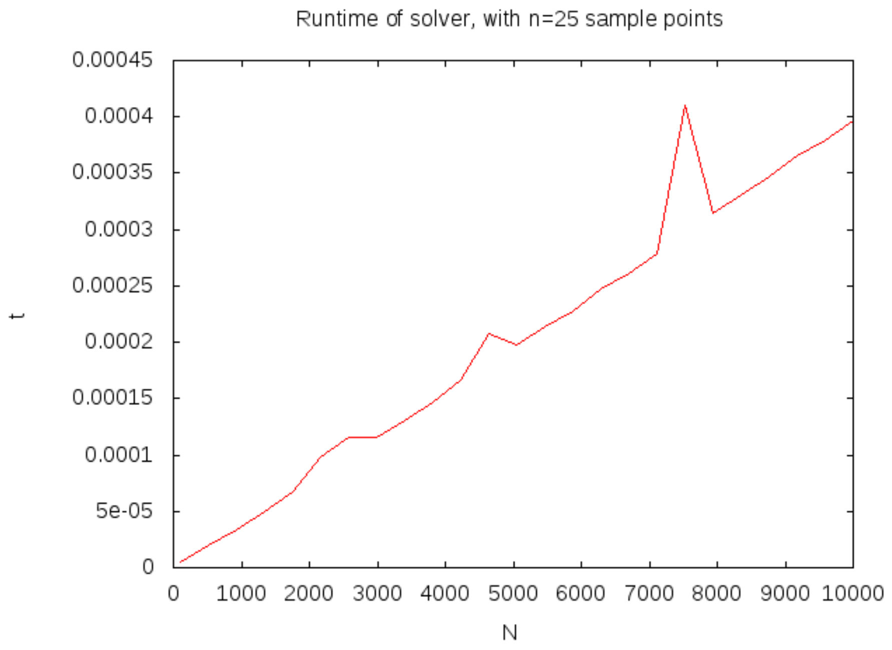
(c)



We see from the plot that if we increase N by a factor of 10 the error goes down by a factor of 100, all the way up to $N \simeq 2 \cdot 10^5$. This is what we expect since the error should go as $O(h^2)$ with $h = \frac{1}{N+1}$. As we increase N above this threshold we notice that the error no longer decreases, but instead increases. This is due to the fact that when n gets larger the step size decreases. If h gets small, we will subtract two very similar numbers. This leads to round off errors.

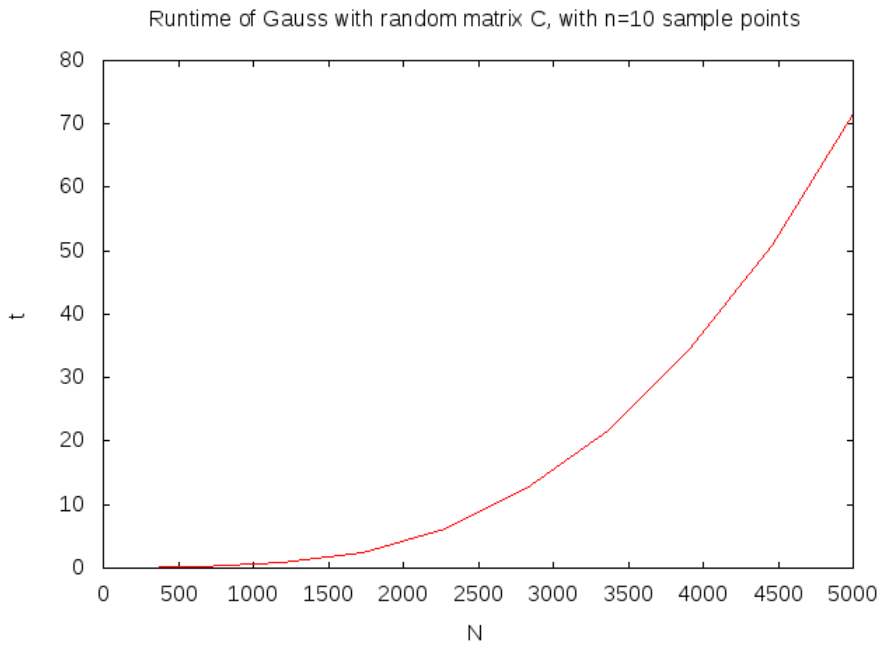
(d)





Above we have plotted the execution time for Armadillo's Gaussian elimination, the LU decomposition and our own algorithm. For the LU decomposition we have only measured the time for the part that solves the system. We see that the execution time for the LU and Gaussian goes as $O(N^2)$ which is what we expect for the LU, but not for Gaussian elimination. We see that the execution time for our algorithm is approximately linear in N, which is what we deduced in b).

Below we have included a plot where we have solved a random linear system using Armadillo's solve, and we see that the execution time goes as $O(N^3)$. Hence it seems like Armadillo exploits that $\tilde{\mathbf{A}}$ is tridiagonal to increase the efficiency.


Runtime of Gauss with random matrix C, with n=10 sample points
t
N