FYS4150 - Computational Physics

PROJECT 1

SVC

Henrik Sverre Limseth
henrisli@uio.no

Jon Vegard Sparre
jonvsp@uio.no

Anne-Marthe Hovda
annemmho@uio.no

Dato: September 10, 2015

Abstract

Exercise a)

Rewriting of differential equation We'll show that we can write our differential equation $-u''(x) = f(x)$ as the matrix equation $Av = \tilde{b}$. First we use the three point formula for a second derivative to get

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i, \quad \text{for } i = 0, \ldots, n,$$

where we multiply both sides by h^2 such that we get $\tilde{b}_i = h^2 f_i$. Then we simply write the matrix A given in the exercise text and multiply it by v to see what we get

$$\begin{aligned} Av &= \begin{pmatrix} 2 & -1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & -1 \\ 0 & \dots & \dots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} v_1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ v_n \end{pmatrix} \\ &= \begin{pmatrix} 2v_1 & -v_2 & 0 & 0 & \dots & 0 \\ -v_1 & 2v_2 & -v_3 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & -v_n \\ 0 & \dots & \dots & 0 & -v_{n-1} & 2v_n \end{pmatrix}, \end{aligned}$$

where we recognize each row as the three point formula for the i-th component of v, *e.g.* for $i = 2$ we have

$$-v_1 + 2v_2 - v_3 = \tilde{b}_2.$$

Checking the given solution We're given the source term $f(x) = 100e^{-10x}$, and we'll check if the solution $u(x) = 1 - (1 - e^{-10})x - e^{-10x}$ is correct when we have the given $f(x)$ as a source term, by inserting it into to Poisson's equation.

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

Exercise b)

The matrix **A** is rewritten as three column vectors in order to optimize our code. We then get n equations,

$$a_i v_{i-1} + b_i v_i + c_i v_{i+1} = \tilde{b}_i.$$

To solve this we do a forward and then a backward substitution. We do the forward substitution in order to get each v_i

$$a_i v_{i-1} + b_i v_i + c_i v_{i+1} = \tilde{b}_i$$
$$v_i = \frac{\tilde{b}_i - a_i v_{i-1} - c_i v_{i+1}}{b_i}$$

For $i = 0$ and $i = n$ we have the initial conditions, which are $u(0) = u(1) = 0$. This implies that our matrix with all the x-values must have $n + 2$ elements in our code since we have to add the end points manually.

For $i = 1$ we get

$$v_2 = \frac{\tilde{b}_1 - a_1 v_0 - b_1 v_1}{c_1}$$

$$v_2 = \frac{\tilde{b}_2 - a_2 v_1 - c_2 v_2}{b_2}$$

Our algorithm is given below

```
1  Declaring variables
2  ...
3  Setting initial conditions
4  ...
5
6  Make for loops that calculate v_i
7  for( i up to n) {
8          temp[i] = c[i-1]/btemp;
9          btemp = b[i]-a[i]*temp[i];
10         v[i] = (f[i] - a[i]*v[i-1])/btemp;
11
12 Saving results
```

For our calculation we have $8n$ FLOPS, $6n$ in the first for-loop and then $2n$ more in the second for-loop.

Exercise c)

In this part of the project we did an analysis of the relative error between our own numerical solution and the exact solution. We calculated the error as described in the exercise text,

$$\epsilon_i \log_{10}\left(\left|\frac{v_i - u_i}{u_i}\right|\right).$$

In my program it looked like this.

```
1      }
2      cout << "max error: " << max << endl;        // Printing
           max value
3  }
4
5  void save_results( double *x, double *v, double *u, double *
       err, int n){
6      FILE *output_file;
7      output_file = fopen("oppgave_b_n_100000.txt" , "w") ;  //
           Is there a way to produce several output files with
           different names?
```

```
8       fprintf(output_file, "   %s    %s    %s    %s \n", "x", "
            v_numerical", "u", "error");
9       int i;
10      for (i=0; i<=n+1; i++){
11          fprintf(output_file, "%12.5f %12.5f %12.5f %12.5f \n"
                ,
12                  x[i], v[i], u[i], err[i] );
```

Below we see the table with our maximum relative errors. We see that it decreases when n is increasing, remember that the error is written as the logarithm of the relative error. In fig. 1, 2, 3, 4 and 5 we see the errors plotted as a function of x. When $n = 10$ the error is constant equal -1.1797, which must be a coincicence?

n	ϵ_{max}
10	-1.1797
10^2	-3.0880
10^3	-5.0800
10^4	-7.0753
10^5	-8.3667

Table 1: Maximum error for different n when using LU-decomposition with forward and backward substitution. At $n = 10^6$ our program crashes. Also note that the error is decreasing with increasing n.

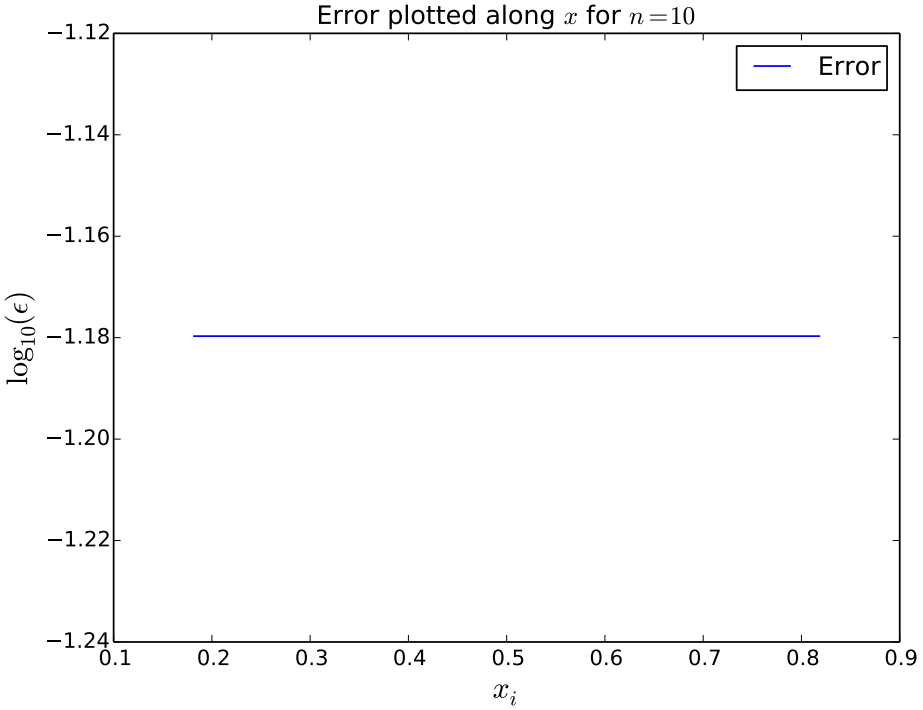

Figure 1: Error plot for $n = 10$

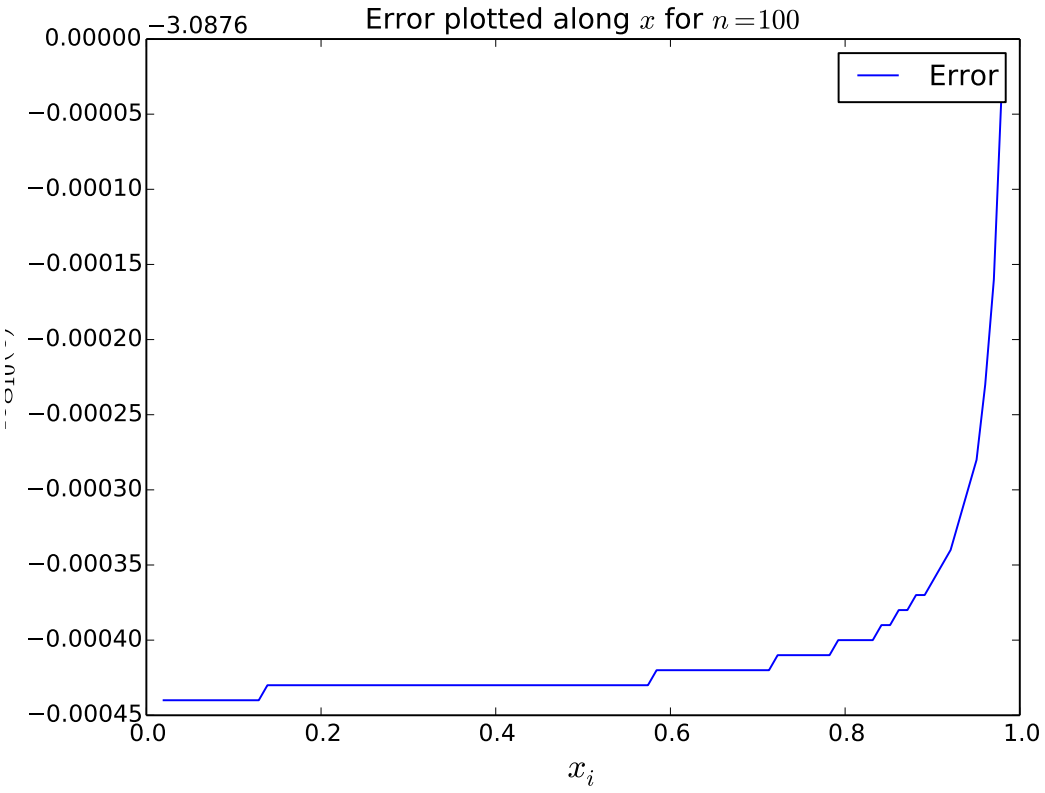

Figure 2: Error plot for $n = 10^2$

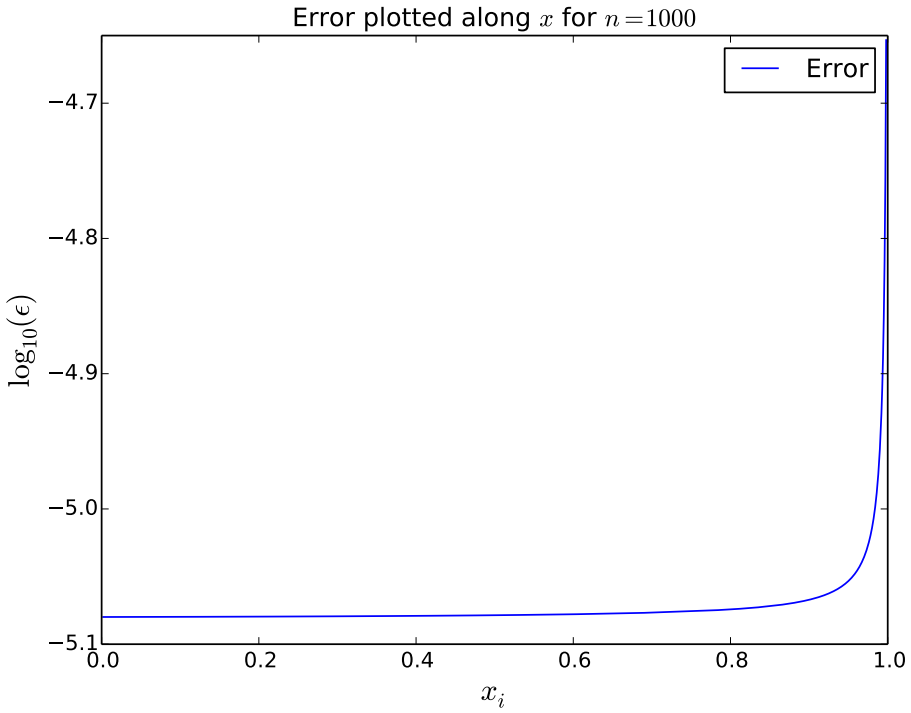

Figure 3: Error plot for $n = 10^3$

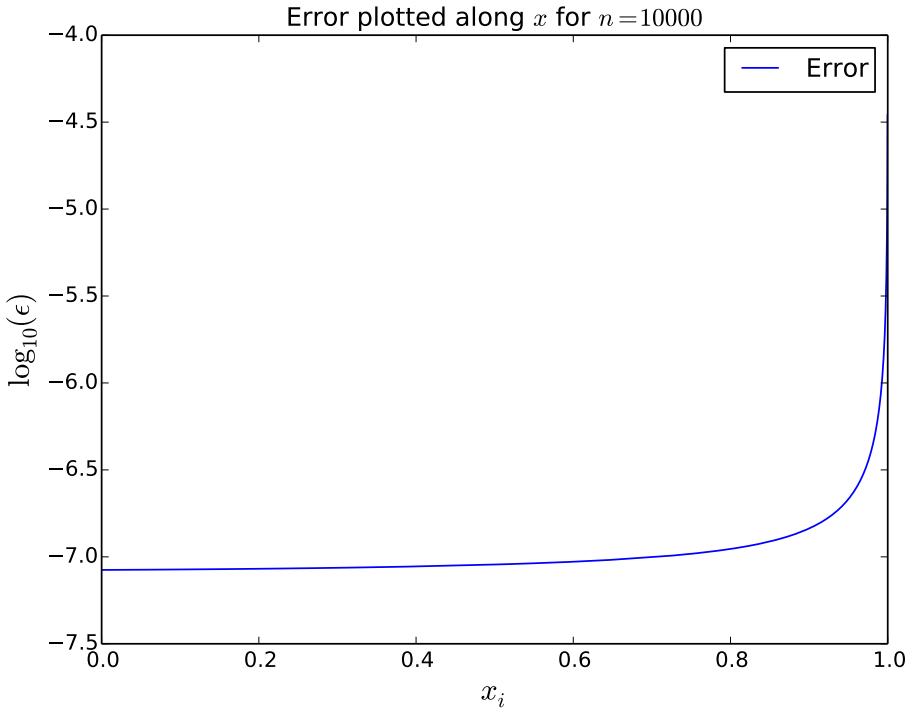

Figure 4: Error plot for $n = 10^4$

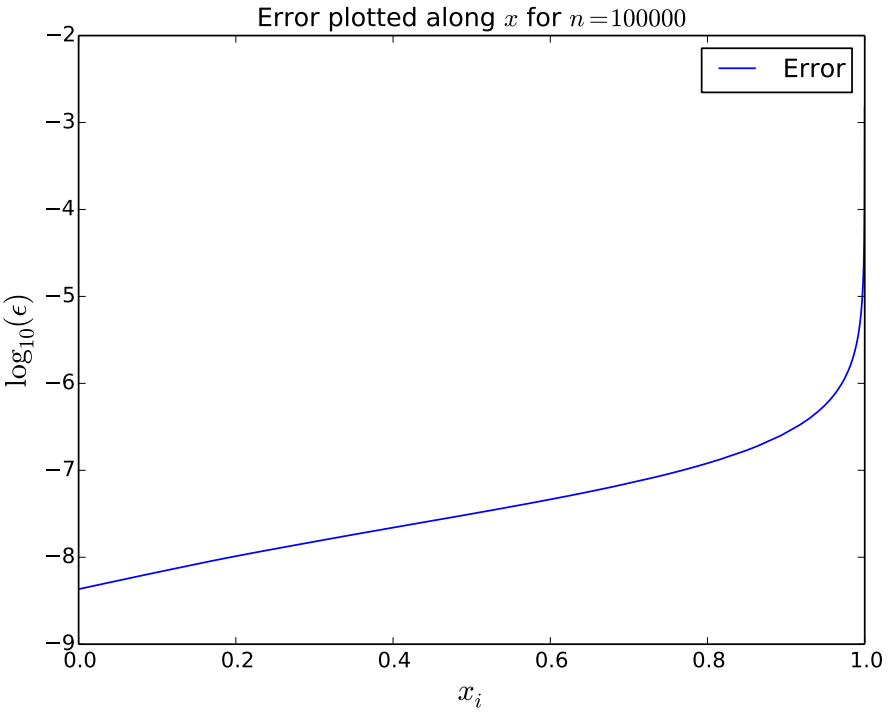

Figure 5: Error plot for $n = 10^5$

Exercise d)

In this last part we're asked to compare our numerical results with the ones we get by using packages like Armadillo.

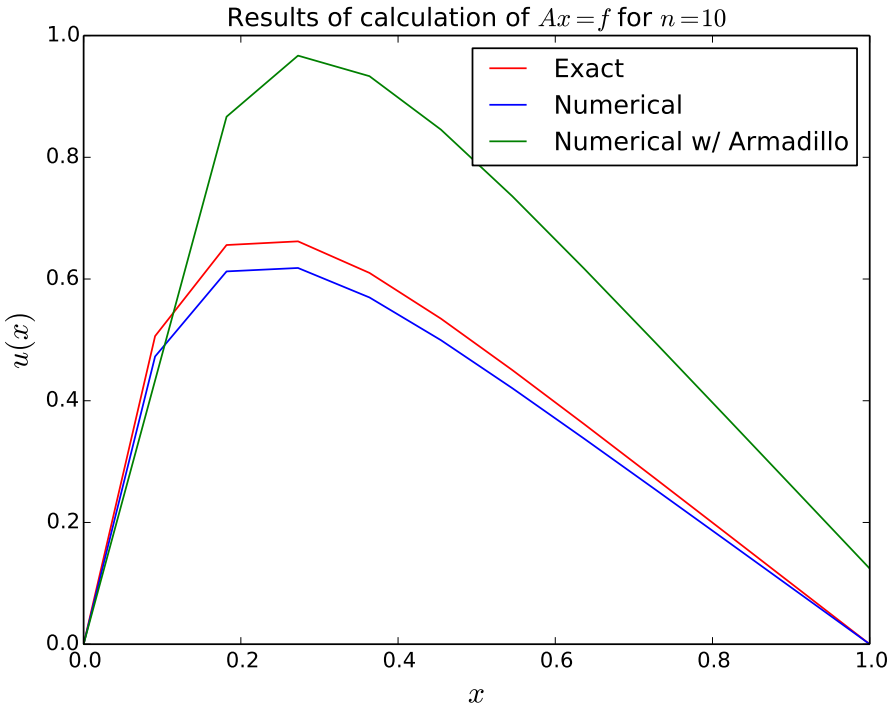

Figure 6: Numerical solution for $n = 10^5$. Armadillo's function 'solve' is pretty far off from the exact and the first numerical solution.

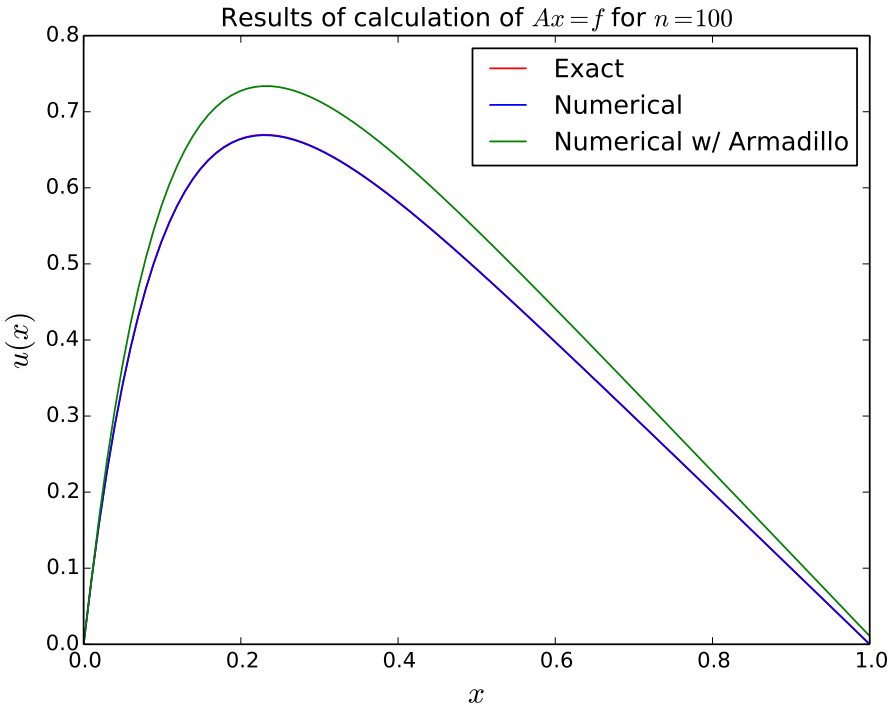

Figure 7: Numerical solution for $n = 10^5$.

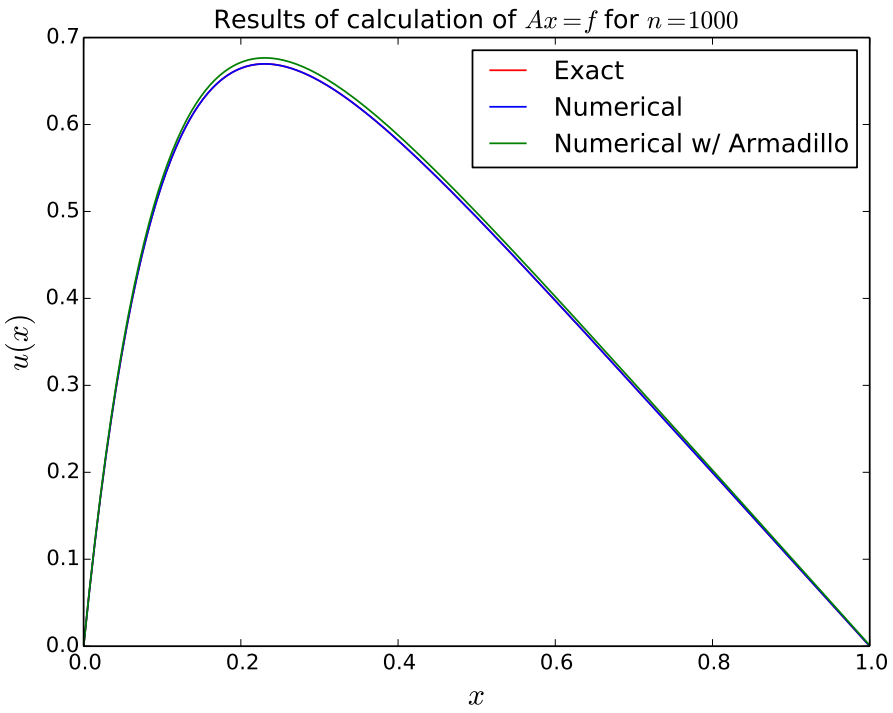

Figure 8: Numerical solution for $n = 10^5$.

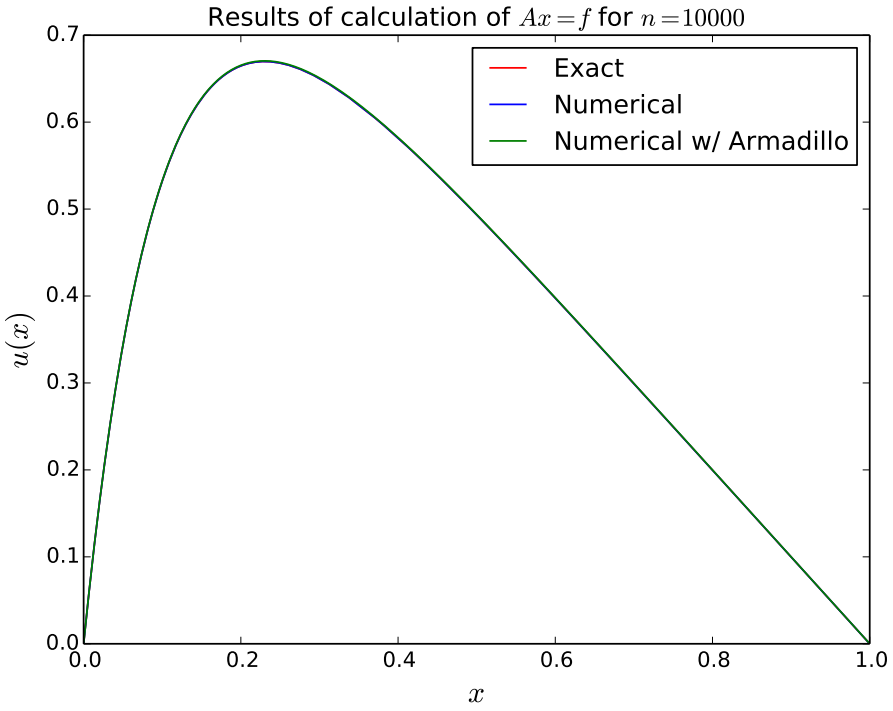

Figure 9: Numerical solution for $n = 10^5$.

n	$t_{\text{Numerical}}$	$t_{\text{Armadillo}}$	$t_{\text{LU decomp.}}$
10	0.000002 s	0.000075 s	0.000123 s
10^2	0.000004 s	0.000701 s	0.002931 s
10^3	0.000027 s	0.068193 s	0.459989 s
10^4	0.000312 s	40.02037 s	297.6299

Table 2: Time spent by the different methods used to solve our problem. Something weird happens with the time-function when we set $n = 10^4$ cause it didn't take five minutes to run the program on my own computer.

When I'm setting $n = 10^6$ Armadillo won't even make matrices for me, it tells me to use some other compilator, this may be because the matrices are too big for the "normal" Armadillo-library to handle when it comes to memory capacity.

```
1  #include <iostream>
2  #include <time.h>
3  #include "armadillo"
4  #include "arma_solve.h"
5
6  using namespace std;
7  using namespace arma;
8
9  ////////////////////////////////////////////////
10 // Solving the same problem with Armadillo
11 ////////////////////////////////////////////////
12
13 mat arma_solve(int n, double h, mat x_mat, mat x2){
14     int i;
15
16     clock_t start , finish ; // declare start and final time
           start = clock () ;
17
18     mat A = zeros<mat>(n+1,n+1);
19     mat f2 = zeros<mat>(n+1,1);
20
21     for (i=1; i<=n; i++){
22         f2(i) = h*h*100*exp(-10*x_mat(i));
23     }
24
25     // Filling matrix A
26     A.diag() += 2.;
27     A.diag(1) += -1.;
28     A.diag(-1) += -1.;
29
30     //cout << A << endl;
31
32     start = clock();
33
34     x2 = solve(A, f2);
35
36     finish = clock();
37
38     printf ("Elapsed time Armadillo: %5.9f seconds.\n", ( (
           float( finish - start )) / CLOCKS_PER_SEC ));
39
```

```
40      // Shifting the array one element
41      x2.reshape(n+3,1);
42      for (i=n; i>=0; i--){
43          x2(i+1) = x2(i);
44      }
45
46      x2(0) = 0.;
47
48      return x2;
49  }
```

```
1   #include <iostream>
2   #include <time.h>
3   #include "armadillo"
4   #include "lu_decomposition.h"
5
6   using namespace std;
7   using namespace arma;
8
9   ////////////////////////////////////////////////
10  // Solving the same problem by LU-decomposition
11  ////////////////////////////////////////////////
12
13  mat lu_decomposition(int n, double h, mat x_mat, mat x3){
14      int i;
15      mat w, y, L_inv, U_inv;
16
17      clock_t start , finish ; // declare start and final time
            start = clock () ;
18
19      // Initializing matrices for the calculations
20      mat A = zeros<mat>(n+1,n+1);
21
22      // Filling matrix A
23      A.diag() += 2.;
24      A.diag(1) += -1.;
25      A.diag(-1) += -1.;
26
27      mat f2 = zeros<mat>(n+1,1);
28
29      for (i=1; i<=n; i++){
30          f2(i) = h*h*100*exp(-10*x_mat(i));
31      }
32
33      w = f2;
34
35      mat L,U,P;
36
37      // Doing the LU-decomposition and finding x
38      // as described in the lecture notes.
39
40      start = clock();
41
42      lu(L,U,P,A);
43
44      L_inv = inv(L);
```

```
45
46      y = L_inv*w;
47
48      U_inv = inv(U);
49
50      x3 = U_inv * y;
51
52      finish = clock();
53
54      // Shifting the array one element
55      x3.reshape(n+3,1);
56      for (i=n; i>=0; i--){
57          x3(i+1) = x3(i);
58      }
59
60      x3(0) = 0.;
61
62      printf ("Elapsed time LU-decomposition: %5.9f seconds.\n"
            , ( (float)( finish - start ) / CLOCKS_PER_SEC ));
63
64      cout << "LU_decomp. success" << endl;
65      return x3;
66  }
```

Wikipedia says that LU-decomp. requires $(2/3)n^3$ FLOPS.
Armadillo won't compile when I'm trying to make $(10^6 \times 10^6)$-matrices.