FYS3150 - Project 1

Steffen Brask

10. september 2014

Project 1

The goal of this project is to solve the one-dimensional Poisson equation with Dirichlet bound- ary conditions.

part a)

the equation we are to solve has the form:

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

we can approximate the second derivative of a function with discrete values like this:

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

Where $f_i = f(x_i)$, and x_i is the discrete values of x defined as $x_i = ih$ where $h = 1/(n+1)$.

If we now assume that we can solve this problem as a set of linear equations on the form:

$$\mathbf{A}\mathbf{v} = \tilde{\mathbf{b}}. \tag{3}$$

Where $\mathbf{A}$ is an $n \times n$ tridiagonal matrix, and $\tilde{b}_i = h^2 f_i$. We now take a wild guess on the form of $\mathbf{A}$, because we know where we want to end up, so let $\mathbf{A}$ be:

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 0 & \ldots & \ldots & 0 \\ -1 & 2 & -1 & 0 & \ldots & \ldots \\ 0 & -1 & 2 & -1 & 0 & \ldots \\ & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & & -1 & 2 & -1 \\ 0 & \ldots & & 0 & -1 & 2 \end{pmatrix} \tag{4}$$

Scince $\mathbf{v} = [v_1, v_2, ..., v_n]$, we can see here that:

$$\mathbf{A}\mathbf{v} = \begin{pmatrix} 2v_i & -1v_{i+1} & 0 & \ldots & \ldots & 0 \\ -1v_i & 2v_{i+1} & -1v_{i+2} & 0 & \ldots & \ldots \\ 0 & -1v_{i+1} & 2v_{i+2} & -1v_{i+3} & 0 & \ldots \\ & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & & -1v_{n-2} & 2v_{n-1} & -1v_n \\ 0 & \ldots & & 0 & -1v_{n-1} & 2v_n \end{pmatrix} \tag{5}$$

This gives for all the rows exept the first and last that the equation$\mathbf{Av} = \tilde{\mathbf{b}}$ gields $\tilde{\mathbf{b_i}} = -v_{i-1} + 2v_i - v_{i+1}$, and scince $\tilde{b}_i = h^2 f_i$, we therefore end up with the equation:

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i \tag{6}$$

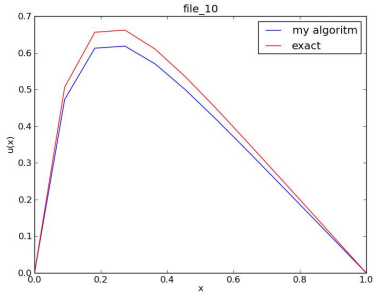
This is the base we are going to build our algorithm on. Be ware though that the first and last row is a little different. But this is fine for our case scince we are solving the equation with $u(0) = u(1) = 0$

part b)

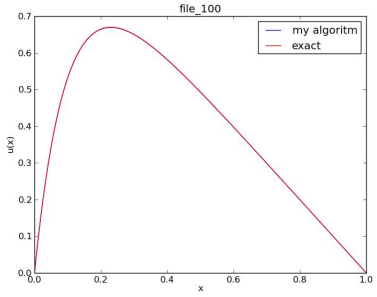
We now need to outline the algorithm. The above tridiagonal system can be written as $a_i v_{i-1} + b_i v_i + c_i v_{i+1} = \tilde{b}_i$. where $a_i = c_i = -1, and b = 2$. This system can be solved in two steps. (i) A forward substitution, and (ii) a backward substitution.

(i)
This step is all about making all the $c_i = 0$. To ensure $c_2 = 0$ we can subtract rowII from rowI times a constant. So: $rowII - x * rowI$ for c_2 this gives an equation $c_2 = b_1 * x \Rightarrow x = \frac{c_2}{b_2}$, with this constant $c_2 = 0$ and we have to apply this to the rest of the row: $b_2' = b_2 - a_1 \frac{c_2}{b_1}$, and $u_2' = u_2 - u_1 \frac{c_2}{b_1}$. We now continue the procedure on rowIII: $x = \frac{c_3}{b_2'} \Rightarrow b_3' = b_3 - a_2 \frac{c_3}{b_2'}$ and $u_3' = u_3 - u_2' \frac{c_3}{b_2'}$
And we now see the system, for each turn we ned to calculate the constant x, to get b_i', which gives us u_i'.

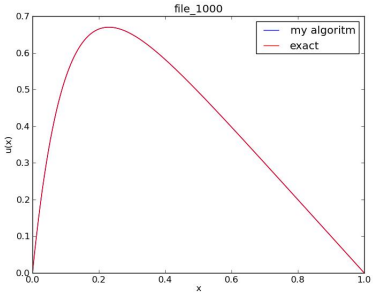
(ii)
The next step is to make the diagonal = 1 (the $b_i = 1$), and all the $a_i = 0$ to do this we use the same logic as step (i) backwards $row(n-1) - x * row(n)$, this gives a constant $x = \frac{a_n}{b_{n+1}'}$, but we want to make $b_{n+1}' = 1$, and because of this the tha constant $x = a_n$ and the whole bacwards substitution simplifies to $u_n'' = u_n' - a_n * u_{n+1}$, and all we need to do now is to force the $n = 0$, and $n = n + 1$ to be 0 and let $n = n + 2$ so we get the endpoints. My Main.cpp file is added at the end of the PDF. i ran it with n= 10, 100, 1000 and here are the plots:



(a) N = 10



(b) N = 100



(c) N = 1000

Figur 1: *plot of my algorithm vs exact analytical solution for different resolutions*

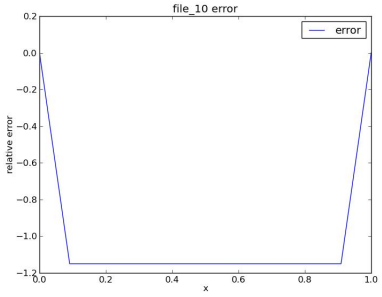
We can see here that we lose precicion for n = 10, but for n = 100, and n = 1000 we get good aproximations.

part c)

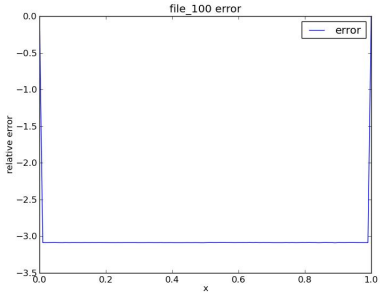
In this part we are going to compute the relative error in the algoritm for different values of n. We compute the relative error with the formula

$$\epsilon_i = log_{10}\left(\left|\frac{v_i - u_i}{u_i}\right|\right), \tag{7}$$

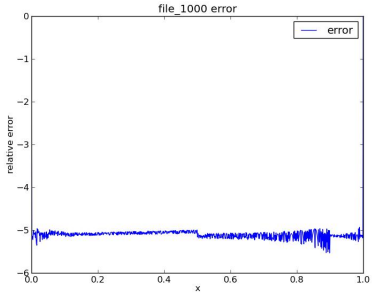
This is easily done by storing the data in a file and plotting in python. (Python script is added at the end). So lets look at some plots!



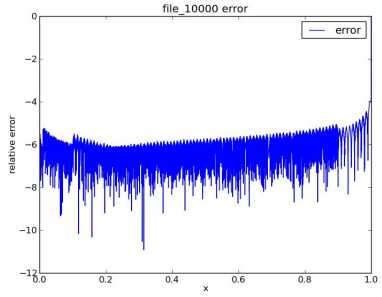
(a) N = 10



(b) N = 100



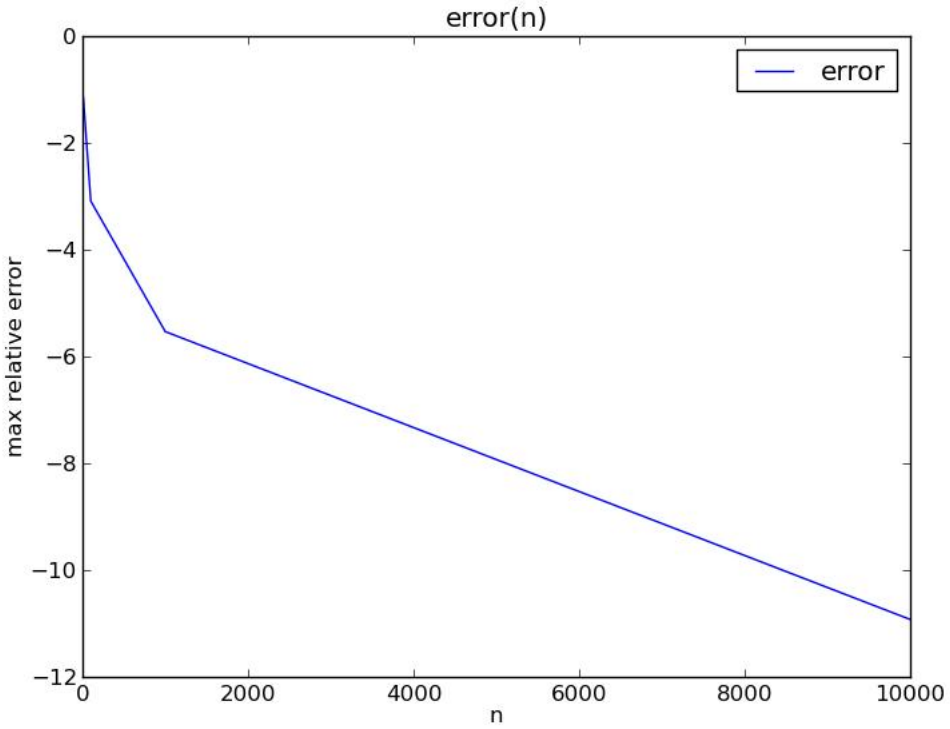
(c) N = 1000



(d) N = 10000

Figur 2: *plot of my relative error for discrete values of x*

I decided to add the plot of the error for each ϵ_i This is because i found them interesting. I would ecpect the error to increase gradually, but that is not the case. It seems to fluctuate for n = 1000, and n = 10000. I can not explain this. I tried to check for n = 100000, but then i got a memory problem.



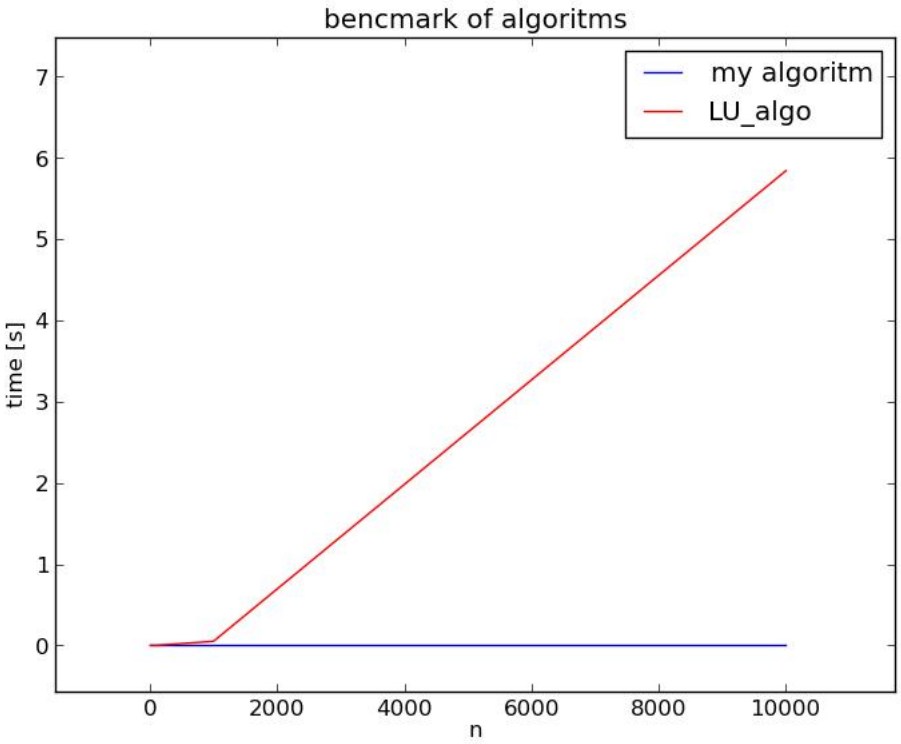
(a) N = 10

Figur 3: *plot of max relative error for n = 10, 100, 1000, 10000*

In this plot we se that the relative error gets smaller end smaller for higher values of n. this is what we would expect as we get higher numerical precision.

part d)

In this section we are going to compare the algorithm to a standard algoritm by using armadillo's lu() and solve() functions. I am first going to make a plot of the time used to compute with my algoritm vs the time used to compute by LU decomposition and solving in armadillo.



(a) N = 10

Figur 4: *plot benchmark times for the two different methods of solving*

The numbers i got for mu algorithm where all 0.0 i tried to figurre out how to get a higher precision in the timer with no luck. But the point still stands here. We can see that my algorithm solves in basically the same time for all values of n. While the Armadillo version seems to shoot in the "sky" exponentially. This is expected though scince the Armadillo algorithm typically uses n^2+n^3 flops while mine uses $8(n-1)$ flops and therefore is much more effective.

Attachments

Attachment 1: C++ main program

```
#include <iostream>
#include <armadillo>
#include <fstream>
#include <cmath>
#include <cstdlib>
#include "time.h"

using namespace std;
using namespace arma;

int main(int argc, char* argv[])
    {
    ofstream myfile;
    int n = atoi(argv[1]);
    //char *filename = new char[10000];
    //printf(filename, "file_%d.dat", n);
    //myfile.open(filename);
    myfile.open("file_100000.dat");
    clock_t start, finish; //declare start and final time
    start = clock();
    double a = -1;
    double b = 2;
    double c = -1;
    vec temp = zeros(n+2);
    vec f = zeros(n+2);
    vec u = zeros(n+2);
    double dx = 1.0/(n+1);
    vec btemp = zeros(n+2);

    for(int i=0 ; i < n+1 ; i++) {
        f(i) = dx*dx*100.0*exp(-10*dx*i);
    }
    btemp(1) = b;
    u[1] = f[1];

    for(int i=2 ; i <= n ; i++) {
```

```
    temp[i] = c/btemp[i-1];
    btemp[i] = b-a*temp[i];
    u[i] = f[i] - u[i-1]*temp[i];
}
u[n] = u[n] / btemp[n];

for(int i=n-1 ; i >= 1 ; i--) {
    u[i] = (u[i] - u[i+1]*a)/btemp[i];
}

finish = clock();
double totaltime = ( (finish - start)/((double)CLOCKS_PER_SEC) );

//cout << totaltime << endl;
{
clock_t start, finish; //declare start and final time
start = clock();

mat A = zeros<mat>(n,n);
A.diag(0).fill(2);
A.diag(1).fill(-1);
A.diag(-1).fill(-1);

mat L, U;
lu(L, U, A);
vec f_without_end_points = f.subvec(1, n);

vec y = solve(L,f_without_end_points);
vec x = solve(U,y);

finish = clock();
double totaltime_LU = ( (finish - start)/((double)CLOCKS_PER_SEC)


for (int i=0; i<=n+1; i++) { // writes file with 3 columns, data f
    myfile << u[i] << ' ' << i*dx << ' ' << x[i] << ' ' << totaltin
//myfile << '0' << ' ' << '0' << ' ' << '0' << ' ' << totaltime <<
}

myfile.close();
}
```

```
    return 0;
}
```

Attachment 2: python script for ploting values from a file

```
import matplotlib.pyplot as plt
from math import *
from numpy import*

def readfile(name):
    infile = open('%s.dat' % name, 'r')
    data = []
    x_list = []

    while True:
        line = infile.readline()
        if not line:
            break
        numbers = line.split()
        dat = float(numbers[0])
        x = float(numbers[1])
        #flux = float(numbers[2])

        data.append(dat)
        x_list.append(x)
        #fluxs.append(flux)
    infile.close()

    return [asarray(data), asarray(x_list)]

name = 'file_10'
data = readfile(name)[0]
x = readfile(name)[1]

exact = 1.-(1-exp(-10))*x-exp(-10*x)

plt.plot(x, data, '-b', label = 'my_algoritm')
plt.title(name)
plt.hold('on')
plt.plot(x, exact, '-r', label = 'exact')
plt.legend()
plt.ylabel('u(x)')
plt.xlabel('x')
```

```
plt.savefig('plot_N_%s.jpg' %name)
plt.show()
```

Attachment 3: python script for computing and ploting error

```
import matplotlib.pyplot as plt
from math import *
from numpy import*

def readfile(name):
    infile = open('%s.dat' % name, 'r')
    data = []
    x_list = []

    while True:
        line = infile.readline()
        if not line:
            break
        numbers = line.split()
        dat = float(numbers[0])
        x = float(numbers[1])
        #flux = float(numbers[2])

        data.append(dat)
        x_list.append(x)
        #fluxs.append(flux)
    infile.close()

    return [asarray(data), asarray(x_list)]

name = 'file_1000'
data = readfile(name)[0]
x = readfile(name)[1]
exact = 1.-(1-exp(-10))*x-exp(-10*x)

error = zeros(len(data))
for i in range(len(data)):
    if data[i] == 0:
        error[i] = 0.
    else:
        error[i] = log10(abs( (exact[i]-data[i])/data[i] ))

print max(abs(error))
```

```
max_error = [-1.15000920684, -3.08918388604, -5.53390627447, -10.92432
error_n = [10, 100, 1000, 10000]

plt.plot(error_n, max_error, '-b', label = 'error')
plt.title('error(n)')
plt.legend()
plt.ylabel('max_relative_error')
plt.xlabel('n')
plt.savefig('max_error.jpg')
plt.show()

"""
plt.plot(x, error, '-b', label = 'error')
plt.title(name + ' error')
plt.legend()
plt.ylabel('relative error')
plt.xlabel('x')
plt.savefig('error_n_1000.jpg')
plt.show()
"""
```

Attachment 4: python script for ploting benchmark times

```
import matplotlib.pyplot as plt
from math import *
from numpy import*

my_algo = [0, 0, 0, 0]
LU_algo = [0, 0, 0.05, 5.84]
n = [10, 100, 1000, 10000]


plt.plot(n, my_algo, '-b', label = 'my algoritm')
plt.title('bencmark of algoritms')
plt.hold('on')
plt.plot(n, LU_algo, '-r', label = 'LU_algo')
plt.legend()
plt.ylabel('time [s]')
plt.xlabel('n')
plt.savefig('benchmark.jpg')
plt.show()
```