University of Oslo

FYS4150

Kristine Moseid, Helene Aune og Helle Bakke

Introduction to numerical projects

September 18, 2016

Contents

1 Abstract **2**

2 Introduction **2**

3 Methods **2**
3.1 Tridiagonal matrix 3
3.2 Relative error 4
3.3 LU decomposition 4

4 Results **4**
4.1 Tridiagonal algorithm for special case 5
4.2 Relative error 5
4.3 LU decomposition 5

5 Conclusion **6**

6 Appendix **6**

7 References **7**

1 Abstract

- Tease the reader
- Write last

2 Introduction

- Motivate the reader
- What have we done
- Structure of report

The aim of this project is to solve the one-dimensional Poisson equation with Dirichlet boundary conditions by rewriting it as a set of linear equations. We will be solving the equation

$$\frac{d^2\phi}{dr^r} = -4\pi r\rho(r)$$

By letting $\phi \to u$ and $r \to x$ it is simplified to

$$-u''(x) = f(x), \quad x \in (0,1), \quad u(0) = u(1) = 0$$

where we define the discretized approximation to u as v_i with grid point $x_i = ih$ in the interval from x_0 to $x_{n+1} = 1$, and the step length as $h = 1/(n+1)$.

By doing this we will be able to create algorithms for solving the tridiagonal matrix problem, and find out how efficient this is compared to other matrix elimination methods.

3 Methods

- Describe methods and algorithms
- Explain
- Calculations to demonstrate the code, verify results(benchmarks)

3.1 Tridiagonal matrix

With the bounadry condition $v_0 = v_{n+1} = 0$, the approximation of the second derivative of u was written as

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

where $f_i = f(x)$. We then rewrote the equation as a linear set of equations:

$$-(v_{i+1} + v_{i-1} - 2v_i) = h^2 f_i$$

We set $h^2 f_i = d_i$, and solved this equation for a few values of i.

$i = 1$:

$$\begin{aligned} -(v_{1+1} + v_{1-1} - 2v_1) &= d_1 \\ -(v_2 + v_0 - 2v_1) &= d_1 \\ -v_2 - 0 + 2v_1 &= d_1 \end{aligned}$$

$i = 2$:

$$\begin{aligned} -(v_{2+1} + v_{2-1} - 2v_2) &= d_2 \\ -v_3 - v_1 + 2v_2) &= d_2 \end{aligned}$$

$i = 3$:

$$\begin{aligned} -(v_{3+1} + v_{3-1} - 2v_3) &= d_3 \\ -v_4 - v_2 + 2v_3) &= d_3 \end{aligned}$$

We saw that this could be written as a linear set of equations $\mathbf{Av} = \mathbf{d}$,

$$\begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \dots \\ v_{n-1} \\ v_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ \dots \\ d_{n-1} \\ d_n \end{pmatrix}$$

3.2 Relative error

We computed the relative error in the data set $i = 1, ..., n$ by using the expression

$$\epsilon_i = log_{10}\Big(\Big|\frac{v_i - u_i}{u_i}\Big|\Big)$$

We implemented the cosed-form solution $u(x) = 1 - (1 - e^{-10})x - e^{-10x}$ to our code and calculated the relative error when increasing n to $n = 10^7$.

3.3 LU decomposition

In this exercise we used the C++ library *'armadillo'* to solve the LU decomposition for matrices of size 10×10, 100×100 and 1000×1000. We implemented the linear equation

$$\mathbf{A} = \mathbf{Lu}$$

and used

```
1    // find the LU decomposition
2    lu(L,U,A);
```

to solve it.
We also tried to run the code for a matrix of size $10^5 \times 10^5$.

4 Results

- Present results
- Critical discussion
- Put code etc. on GitHub and explain to reader where they can find it
- Explanatory figures with captions, labels etc.

4.1 Tridiagonal algorithm for special case

n	CPU time general	CPU time special
10	4e-6	2e-6
10^2	8e-6	6e-6
10^3	6.7e-5	4.3e-5
10^4	6.08e-4	4.79e-4
10^5	0.004186	0.002699
10^6	0.037559	0.22339

Table 4.1 CPU time for general and special algorithm

We saw that the special algortihm was a bit faster than the general.

4.2 Relative error

In our python-code we took the minimum value of each list of error values. This was because the error values became negative, as a result of $u(x)$ being exponetial. We created a table of the relative error results:

n	ϵ
10	-1.1797
10^2	-3.08804
10^3	-5.08005
10^4	-7.07936
10^5	-9.0049
10^6	-6.77137
10^7	-12.8074

Table 4.2 Table of the relative error ϵ for increasing n

We saw that the error became smaller when n increased, but for $n = 10^6$ this was not the case. Why?

4.3 LU decomposition

The number of floating point operations for the LU decomposition is $\frac{2}{3}n^3$(**?**). We noticed that the program used some time to execute the command. The matrix of size $10^5 \times 10^5$ did not execute. This was because the memory

needed to create the matrix was too big for our computers. We calculated how much memory was needed to create the matrix:

$$\begin{aligned}\frac{10^5 \cdot 10^5 \cdot 64}{8} &= \frac{8 \cdot 10^{10}}{1024} \text{ B} \\ &= \frac{78125000}{1024} \text{ kB} \\ &= \frac{762939}{1024} \text{ MB} \\ &= 74.5 \text{ GB}\end{aligned}$$

It was with good reason that our computers where not able to execute the program for this matrix size.

5 Conclusion

- Main findings
- Perspectives on improvement and future work

Although it may be tempting to use *'armadillo'* to solve tridiagonal matrix decompositions, our results showed that it was more efficient to create algorithms to solve the problem. The CPU time for forward and backward substitution was shorter than for the LU decomposition. The relative error when solving the algorithms became smaller for larger values of n. However, the figures showed us that we did not need very large values of n to get a good approximation to the exact function. Calculating the algorithms for large vectors took a longer time, and we learned that we did not have to do this to get a good result.

6 Appendix

- Additional calulations
- Selected calulations with comments
- Code, if necessary
- Appendix can be pushed to GitHub!

To run the programs, one needs to give an input argument. This argument varies between $1, ..., 7$, and refers to the exponent of 10 for each n-value. This applies to the programs *'project1_main.cpp'* and *'project1_c.cpp'*.

The program *'project1_main.cpp'* contains the algorithms calculated in exercise 1b, the calculation of the relative error ϵ_i in exercise 1d and the calculation of CPU time. The program *'project1_c.cpp'* contains the specialized algorithm, and the QT folder contains the LU decomposition using *'armadillo'*.

The program *'project1_python.py'* contains the plotting of figures in exercise 1b, as well as a calculation of the maximum value in each error list. All textfiles are created in the main C++ program, and imported into the Python program for further computations.

7 References

- Reference to material we based our work on(lecture notes etc.)
- Find scientific articles, books etc.
- BibTex - extract references online