University of Oslo

FYS4150

Kristine Moseid, Helene Aune og Helle Bakke

Introduction to numerical projects

September 17, 2016

Contents

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

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

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 $n = 10^7$.

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 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 4.1 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?

5 Conclusion

- Main findings
- Perspectives on improvement and future work

6 Appendix

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

7 References

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