FYS 4150 - Computational Physics Project 1: Solving Poisson's equation in one dimension

Maren Rasmussen Markus Leira Asprusten Metin San

4. September 2018

Abstract

This project involves solving the one-dimensional Possion equation with dirichlet boundary conditions using two different algorithms. The first method is the tridiagonal matrix algorithm while the second is the LU decomposition. The conclusion of the project is that a specialized version of the tridiagonal algorithm is much faster.

1. Introduction

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

2. Theory

2.1. **The Poisson Equation.** Poisson's equation is a classical and well known differential equation from electromagnetism. The equation describes the potential field Φ generated from a charge distribution $\rho(\mathbf{r})$. For three dimensions, the equation is given by

$$\nabla^2\Phi = -4\pi\rho(\mathbf{r}),$$

where $\nabla =$ is the Laplace operator. If both Φ and $\rho(\mathbf{r})$ is spherically symmetric, the equation can be simplified to an one dimensional equation in r,

$$\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\Phi}{dr}\right) = -4\pi\rho(r).$$

By substituting $\Phi = \phi(r)/r$, we can simplify the equation even more, giving

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

The final equation can be written in a general form by letting $\phi \to u$ and $r \to x$, and defining the right hand side of the equation as f,

$$-u''(x) = f(x).$$

In this study we will assume that the source term is $f(x) = 100e^{-10x}$ with $x \in [0, 1]$. We will also assume that we have Dirichlet boundary conditions, such that $u(0) = u(1) = 0$. With these assumptions the equation have an exact solution on the form $u(x) = 1 - (1 - e^{-10})x - e^{-10x}$.

2.2. **Discretization of the problem.** To solve the Poisson equation numerically, we need to discretize the problem. The discritation can be done using $n + 1$ x-values, such that $x \in [x_0, x_1, x_2, ..., x_n]$ with $x_0 = 0$ and $x_n = 1$, and $u(x_i) = u_i$. Then we have that

$$x_i = x_0 + ih,$$

where $h = (x_n - x_0)/n$ is the step size.

A discretized version of $\frac{d^2u(x)}{dx^2}$ can be found by using Taylor expansion. We know that

$$u(x+h) = u(x) + hu' + \frac{h^2}{2!}u'' + O(h^2)$$
$$u(x-h) = u(x) - hu' + \frac{h^2}{2!}u'' + O(h^2)$$

The $O(h^2)$-term is the remaining terms from the Taylor expansion, or the error we get by excluding these terms. By adding $u(x + h)$ and $u(x - h)$, we can derive the desired expression:

$$u(x+h) + u(x-h) = 2u(x) + \frac{2}{2!}h^2u'' + O(h^2)$$
$$u''(x) = \frac{u(x+1) + u(x-h) - 2u(x)}{h^2} + O(h^4)$$
$$u'' = \frac{u_{i+1} + u_{i-1} - 2u_i}{h^2} + O(h^2)$$

So by excluding the rest of the terms in the Taylor expansion, and defining $f_i^* = f_i h^2 = f(x_i)h^2$, our equation is given by:

$$-u_{i+1} - u_{i-1} + 2u_i = f_i^*.$$

For each x-value we then get a set of linear equations,

$$\begin{aligned} -v_2 - v_0 + 2v_1 &= f_1^* \\ -v_3 - v_1 + 2v_2 &= f_2^* \\ &\vdots \\ -v_n - v_{n-2} + 2v_{n-1} &= f_{n-1}^*. \end{aligned}$$

It is easy to see that the right hand side of the equation set, can represent as a vector on the fllowing form

$$\hat{\mathbf{f}} = \begin{bmatrix} f_1^* \\ f_2^* \\ \vdots \\ f_{n-1}^* \end{bmatrix}$$

The left hand side of the equation set we can represent as a matrix product on the following form

$$\hat{\mathbf{A}}\hat{\mathbf{u}} = \begin{bmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \end{bmatrix}.$$

This means that the set of equations can be written as a linear algebra problem on the form

$$\hat{\mathbf{A}}\hat{\mathbf{u}} = \hat{\mathbf{f}}.$$

?. Results

?. Conclusion / Discussion ellerno

The results presented in section ? shows that by only including the first terms of the Taylor expansion to approximate the second derivative, does not give a big error. At least not when using a big n. [...]

Section ? shows that even though we know from the mathematical theory that a bigger n and a smaller stepsize h shoud give better results, this is not the case nummerically. The reason for this, is that there is a limit on how accurate a number can be represented on a computer. Beacuse of this limit, the computer has to make round offs, which increases the total error. This means that there has to be a ideal stepsize h, giving the best results. From figure ?, we found

General Algorithm	Special Algorithm
0.106575	0.066029
0.106026	0.066126
0.115654	0.066128
0.106315	0.068742
0.106237	0.067858
0.105891	0.067724
0.106279	0.068543
0.107324	0.068350
0.105830	0.067883
0.106344	0.068466

Average Time Spent	
0.1072475	0.068466

that this ideal stepsize was ??. [...]

From table ? we can see that it was not big difference in the run time between the general and the special algorithm. The special was only a bit faster, even thogh it had half as many flops as the general. But when we ran the program on a older computer with a worse processor we could see a much bigger difference between the two run times. On this computer the special algorithm was almost twice as fast as the general. This is also what we expected, since this algorithm had half as many FLOPS as the general.

Bare noen tanker, trenger ikke være med.