FYS3150/4150 - Project 1
Nummerical methods for solving Poisson's equation

Aram Salihi[1], Adam Niewelgowski[1] & Jakob Borg[2]

[1]Department of Physics, University of Oslo, N-0316 Oslo, Norway

[2]Department of Theoretical Astrophysics, University of Oslo, N-0316 Oslo, Norway

September 9, 2018

Abstract

In this article we have chosen to study three different algorithms (Thomas algorithm, a tailored verison of the thomas alogorithm and the famous LU decomposisition) for solving poission equation. We wish to compare the speed and the precision of these three methods. We found out that the LU decomposisition is siginificantly slower than other two algorithms when compared, and the precision GIT repository.[1]

1 Introduction:

The Poission equation is one of the most important and widely used differential equation in different field of sciences, especially in physics (e.g arises when solving different potentials in theoretical physics).

$$\nabla^2 u = f \tag{1}$$

There is few cases in science general that this equation can be solved analytically, but quite often it is not solvable, therefore it is crucial to derive nummerical alogorithms which is time efficient and precise with minimal error. By using dirichlet boundaries and discretizing (**??**) this can be turned into a simple linear algebra problem on the form:

$$A\mathbf{u} = \mathbf{f} \tag{2}$$

Where A is a $N \times N$ tridiagonal matrix. We wish to investigate three different alogorithms mentioned in the abstract to solve this linear algebra problem. This problem can easily be solved by LU decomposisition but as we later find out, this algorithm turns out be very slow when increasing the dimension of the matrix. Due to the nature of tridiagonal matrix a more time effiecient and more precise method can be derived. The Thomas algorithm is a simple method which uses foward and backward Gaussian elimination. We will first present the general Thomas algorithm, a tailored version and LU decomposisition. We will then compare the result and time the time used when running N iteration.

2 Theoretical Model:

2.1 Discretizing Possion's equation:

For the sake of simplicity we will reduce Poisson equation to a one dimensional problem on the form:

$$\frac{\mathrm{d}^2 u(x)}{\mathrm{d}x^2} = f(x)\,, \quad x \in (0,1) \tag{3}$$

By assuming that the function v is a smooth function, we will be using a taylor approximation up to fourth degree and a approximation for the second derivative can be derived:

$$\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} \approx -\frac{2u(x) - u(x+h) - u(x-h)}{h^2} + \mathcal{O}(h^2) \tag{4}$$

[1]`https://github.com/einrone/FYS4150/`

(See appendix for derivation)By discretizing the function $u(x_i)$ to u_i and defining the grid points as $x_i = ih$, where h is defined to be $h = 1/(N+1)$. The above approximation can be formulated as:

$$\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} \approx -\frac{2u_i - u_{i+1} - u_{i-1}}{h^2} + \mathcal{O}(h^2) \tag{5}$$

By imposing Dirichlet boundaries $x_0 = 0$ and $x_{n+1} = 1$ we have that:

$$u(0) = u(1) \rightarrow u_0 = u_{n+1} = 0 \tag{6}$$

Using this approximation and substituting this into Poissons equation, we will then have:

$$-\frac{2u_i - u_{i+1} - u_{i-1}}{h^2} = f_i \tag{7}$$

where $f_i = f(x_i)$ for $i = 1, ..., n$. Using the imposed conditions we can turn this problem into a simple linear algebra problem on the form:

$$A\mathbf{u} = \mathbf{f} \tag{8}$$

where $\mathbf{u} = \{u_1, ..., u_n\}$, $\tilde{\mathbf{f}} = h^2\{f_1, ..., f_n\}$ and A is a $n \times n$ tridiagonal matrix on the form:

$$A = \begin{pmatrix} 2 & -1 & 0 & . & . & 0 \\ -1 & 2 & -1 & . & . & . \\ . & ... & ... & ... & . & . \\ . & . & ... & ... & -1 & 0 \\ . & . & . & -1 & 2 & -1 \\ 0 & . & . & 0 & -1 & 2 \end{pmatrix} \tag{9}$$

As we see the triadiagonal matrix has 2 along its diagonal elements and -1 on the upper and lower diagonal elements and has otherwise 0 on the matrix elements.

2.2 Solving a general tridiagonal problem (deriving Thomas algorithm):

Recall the tridiagonal matrix from the previous section, we will now introduce a general $n \times n$ tridiagonal matrix on the form:

$$A\mathbf{u} = \tilde{\mathbf{f}} \qquad \rightarrow \qquad \begin{pmatrix} b_1 & c_1 & 0 & \dots & \dots & 0 \\ a_1 & b_2 & c_2 & 0 & \dots & 0 \\ 0 & a_2 & b_3 & c_3 & 0 & \dots \\ 0 & 0 & a_3 & b_4 & c_4 & \dots \\ \vdots & & & & \ddots & \vdots \\ 0 & & \dots & & a_{n-1} & b_n \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ \vdots \\ s_n \end{pmatrix} \tag{10}$$

In order to solve this equation we must perform a forward and a backward substitution. By performing Gaussian elimination on the second row in order to get a pivot coloumn, we must multiply the first row with a_1/b_1 and substract the first row with the second row. This will give us the matrix:

$$\begin{pmatrix} b_1 & c_1 & 0 & \dots & \dots & 0 \\ 0 & d_2 & c_2 & 0 & \dots & 0 \\ 0 & a_2 & b_3 & c_3 & 0 & \dots \\ 0 & 0 & a_3 & b_4 & c_4 & \dots \\ \vdots & & & & \ddots & \vdots \\ 0 & & \dots & & a_{n-1} & b_n \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} \tilde{f}_1 \\ s_2 \\ \tilde{f}_3 \\ \tilde{f}_4 \\ \vdots \\ \tilde{f}_n \end{pmatrix} \tag{11}$$

Where we have defined d_2 and s_2 to be:

$$d_2 = b_2 - c_1\frac{a_1}{b_1} \qquad s_2 = \tilde{f}_2 - \tilde{f}_1\frac{a_1}{b_1} \tag{12}$$

By repeating this procudere we will see a pattern, and the foward substitution is derived to be (if we define a initial conditon, $d_1 = b_1$ and $\tilde{f}_1 = s_1$):

$$d_i = b_i - c_{i-1}\frac{a_{i-1}}{d_{i-1}} \qquad s_i = \tilde{f}_i - s_{i-1}\frac{a_{i-1}}{d_{i-1}} \tag{13}$$

We have now successfully produced pivot elements on every coloumns in the matrix.

$$\begin{pmatrix} d_1 & c_1 & 0 & \dots & \dots & 0 \\ 0 & d_2 & c_2 & 0 & \dots & 0 \\ 0 & 0 & d_3 & c_3 & 0 & \dots \\ 0 & 0 & 0 & d_4 & c_4 & \dots \\ \vdots & & & & \ddots & \vdots \\ 0 & & \dots & & 0 & d_n \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ \vdots \\ s_n \end{pmatrix} \tag{14}$$

The equations above can be iterated from $i = 1$ to n and will give us the correct matrix elements when its reduced to its row echelon form. In order to achieve the solution set $\mathbf{u}$ we must perform a backward substitution. Consider following (written out the matrix equation):

$$b_1v_1 + c_1v_2 = s_1 \tag{15}$$
$$d_2v_2 + c_2v_3 = s_2 \tag{16}$$
$$. \tag{17}$$
$$. \tag{18}$$
$$. \tag{19}$$
$$d_{n-1}u_{n-1} + c_{n-1}u_n = s_{n-1} \tag{20}$$
$$d_nu_n = s_n \tag{21}$$

If we look at the second last and last equation in (**??**) and solve for u_n and u_{n-1} we have:

$$u_n = s_n/d_n \qquad u_{n-1} = \frac{s_{n-1} - u_nc_{n-1}}{d_{n-1}} \tag{22}$$

changing the dummy index n to j and using $u_n = s_n/d_n$ as the initial value when performing backward substitution, the equation we obtain is:

$$u_{j-1} = \frac{s_{j-1} - u_jc_{j-1}}{d_{j-1}} \tag{23}$$

Where j iterates from $j = n - 1$ down to $j = 1$. Thus the general algorithm to solve the general tridiagonal matrix problem is:

Foward substitution

$$d_i = b_i - c_{i-1}\frac{a_{i-1}}{d_{i-1}} \qquad s_i = \tilde{f}_i - s_{i-1}\frac{a_{i-1}}{d_{i-1}} \qquad 1 \leq i < N + 1 \tag{24}$$

Backward substitution

$$u_{j-1} = \frac{s_{j-1} - u_jc_{j-1}}{d_{j-1}} \qquad N - 1 \leq j \leq 1 \tag{25}$$

As we later see, this method turns out be a very efficient and precise algorithm when comparing to LU-decomposisition.

3 Optimizing Thomas algorithm for a special case:

It turns out that the previous algorithm which is already quite fast can be modified to a faster algorithm. Recall from previous that the diagonal element had values $a_1, ..., a_{n-1}$, $b_1, ..., b_{n-1}$ and $c_1, ..., c_{n-1}$ where we assumed that none of these values were the same to keep it general. We will now define that:

$$a_1, ..., a_{n-1} = -1 \tag{26}$$
$$b_1, ..., b_{n-1} = 2 \tag{27}$$
$$c_1, ..., c_{n-1} = -1 \tag{28}$$

By doing this we can rewrite our general algorith to fit this specific case: (må forklare nede hvordan man ser at d blir sånn):

Foward substitution

$$d_i = \frac{i+1}{i} \qquad s_i = \tilde{f}_i + \frac{s_{i-1}}{d_{i-1}} \qquad 1 \le i < N+1 \tag{29}$$

Backward substitution

$$u_{j-i} = \frac{s_{j-1} + u_j}{d_{j-1}} \qquad N-1 \le j \le 1 \tag{30}$$

This operation reduces number of FLOPS (see section LATER LOOL - der vi skal beregne dette i diskusjon) for each calculation what makes algorithm more effective.

SKAL vi skrive om LU eller bare gi formelen og hvoran det funker og gi kilde til bevis og sånt? og ellers nøye oss med resultater og diskusjon om presisjon og hastighet hovedsakelig?

4 Method:

Every algorithm listed below is implemented in C++. Armadillo library is used in LU-decomposition.

4.1 General algorithm:

As derived above in section (GIVE SECTION), Thomas algorithm has two loops, forward and backward substitution

```
//forward
for(int i = 2; i <= N+1; i++){
        b[i] -= (a[i-1]*c[i-1])/b[i-1];
        s[i] -= s[i-1]*(a[i-1]/b[i-1]);
    }
//backward
v[N-1] = s[N-1]/b[N-1]; //initial - last element
for(int i = N+1; i > 1; i--){
        v[i-1]= (s[i-1] - c[i-1]*v[i])/b[i-1];
    }
```

we notice that the algorithm is for the general use where every value of matrix elements can be different.

4.2 Specific algorithm:

Implementation of the algorithm for the specified matrix is as follows

```
//forward
    for(int i = 1; i <= N+1; i++){
        b[i] = (i+1.0)/((double)i);
        s[i] += s[i-1]/(b[i-1]);
    }
//backward
    v[N-1] = s[N-1]/b[N-1]; //initial - last element
    for(int i = N+1; i > 1; i--){
        v[i-1] = (s[i-1] + v[i])/b[i-1];
    }
```

algorithm still needs to run through two loops with a few operations, but we can clearly see the simplicity of this algorith compared with general one.

4.3 LU-decomposition:

For calculation with LU-decomposition, we need first to implement initialization of the matrix A

```
        LU - matrise A
```

then we can use lu and solve module to find the solution

```
        kode med lu og solve
```

4.4 Precision:

To be able to say something about accuracy of the numerical solution, we can calculate the relative error given by

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

where u_i is the discretized exact solution and v_i is the simulated solution.

5 Results:

bilder av løsningen og error, vi kan prøve å plotte LU error sammen med gaussian error i en for å se om det er noen forskjell der -beskrive hvorfor feilen øker igjen og hvorfor den faller som den gjør, med stigningstallet 2 i log-log plot

lage tabell med tidene og diskutere dette med flops og beregne de simulation_general.png

Figure 1: Plot of simulated solution compared with the exact function