Project 1

Erlend Lima[1], Frederik J. Mellbye[2], Aram Salihi[3], and Halvard Sutterud[4]

[1,2,3,4]University of Oslo, Oslo, Norway

September 7, 2017

Abstract

In this project three methods of solving the one-dimensional Poisson equation with Dirchelet boundary conditions are investigated. The equation was discretized, and written as a system of linear equations. This equates to a tridiagonal matrix equation, which was solved using the general Thomas algorithm, a specialized version of the Thomas algorithm and by LU-decomposing the matrix. The numerical error is then analyzed for both methods, and the algorithm speeds are investigated and discussed.

The specialization of the algorithm for our specific problem resulted in a x percent reduction in the amount of floating point operations. For large matrices ($n \sim 10^5$) the LU-decomposition requires more memory than what was available, which was not a problem with the Thomas algorithm. Also, the Thomas algorithm produced solutions with smaller errors than the LU method. Therefore the Thomas is deemed most suited for solving the problem examined in this paper.

Table 1: Summary of the errors

n	log error
10	-0.647811
100	-1.69284
1000	-2.69923
10000	-3.69988
100000	-4.69994
1000000	-5.9374

1 Introduction

Introducing where the Poisson equation comes from, why it valuable to solve it, how it is solved in this paper and explaining speed and error analysis.

Figure 1: Caption

2 Theory

2.1 Discretizing the Poisson equation

The one-dimensional Poisson equation with Dirchelet boundary conditions is

$$\frac{d^2u}{dx^2} = -f$$

We can approximate the second derivative of $u(x)$ using a Taylor expansion and solving for $\frac{d^2u}{dx^2}$:

$$\frac{d^2u}{dx^2} \approx \frac{u(x+h) + u(x-h) - 2u(x)}{h^2} + O(h^2)$$

The equation is discretized with grid points $x_1, x_2, \dots, x_n$. Imposing Dirchelet boundary conditions forces $x_1 = x_n = 0$, using the handy notation $u(x_i + h) = u_{i+1}$ and inserting the approximated second derivative into the Poisson equation yields

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

where $h = \frac{1}{n+1}$ is the step size, i.e. the distance between grid points, and $f_i = f(x_i)$. This discretized version of the one dimensional Poisson equation is conveniently a linear system of equations, and can therefore be written as the matrix equation

$$A\mathbf{u} = h^2\mathbf{f}$$

where

$$A = \begin{bmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & & \vdots \\ 0 & -1 & 2 & -1 & 0 & \\ \vdots & 0 & -1 & 2 & -1 & 0 \\ & & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & -1 & 2 \end{bmatrix}$$

is a $n \times n$ tridiagonal matrix, the only non-zero elements are on, directly above or directly below the diagonal. Row reducing the matrix can therefore be done in an efficient way using the Thomas algorithm, as we shall see in the next paragraph.

2.2 Solving a general tridiagonal matrix problem

A general tridiagonal matrix equation is on the form

$$\begin{bmatrix} b_1 & c_1 & 0 & \dots & \dots & 0 \\ a_2 & b_2 & c_2 & 0 & & \vdots \\ 0 & a_3 & b_3 & c_3 & 0 & \\ \vdots & 0 & a_4 & b_4 & c_4 & 0 \\ & & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & a_n & b_n \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_n \end{bmatrix}$$

where $v_i = h^2 f_i$ in the case examined in this project. To get the matrix in row reduced echelon form and solve the problem, the a_i's are first eliminated by Gaussian elimination. First, $\frac{a_2}{b_1}$ times the first row is subtracted from the second. This gives

$$\begin{bmatrix} b_1 & c_1 & 0 & \dots & \dots & 0 \\ 0 & b'_2 & c_2 & 0 & & \vdots \\ 0 & a_3 & b_3 & c_3 & 0 & \\ \vdots & 0 & a_4 & b_4 & c_4 & 0 \\ & & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & a_n & b_n \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} v_1 \\ v'_2 \\ v_3 \\ \vdots \\ v_n \end{bmatrix}$$

where $b'_2 = b_2 - \frac{a_2}{b_1} c_1$ and $v'_2 = v_2 - \frac{a_2}{b_1} v_1$. For the next row, exactly the same operation is used, but notice how the previous elements b'_2 and v'_2 now have been updated in the last iteration. Generalizing this argument for each row, a general expression for the diagonal elements is

$$b'_i = b_i - \frac{a_i}{b'_{i-1}} c_{i-1}$$

where $b'_1 = b_1$ and $i = 2, 3, ..., n$. Similarly, elements in the right-hand side vector $\mathbf{v}'$ are modified to

$$v'_i = v_i - \frac{a_i}{b'_{i-1}} v_{i-1}$$

where $v'_1 = v_1$ and $i = 2, \ldots, n$. After completing the forward substitution, the matrix equation reads

$$\begin{bmatrix} b'_1 & c_1 & 0 & \ldots & \ldots & 0 \\ 0 & b'_2 & c_2 & 0 & & \vdots \\ 0 & 0 & b'_3 & c_3 & 0 & \\ \vdots & 0 & 0 & b'_4 & c_4 & 0 \\ & & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & \ldots & 0 & 0 & b'_n \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ \\ u_n \end{bmatrix} = \begin{bmatrix} v'_1 \\ v'_2 \\ v'_3 \\ \vdots \\ \\ v'_n \end{bmatrix}$$

By inspection, the solution for the final point u_n is explicitly given by the bottom row:

$$b'_n u_n = v'_n \to u_n = \frac{v'_n}{b'_n}$$

The rest of the solution $u(x_i)$ is then computed by backward substitution, given u_i one can compute u_{i-1}. For instance, row $n-1$ reads

$$\begin{aligned} b'_{n-1} u_{n-1} + c_{n-1} u_n &= v'_{n-1} \\ u_{n-1} &= \frac{v'_{n-1} - c_{n-1} u_n}{b'_{n-1}} \end{aligned}$$

In general then, the back substitution is done by

$$u_i = \frac{v'_i - c_i u_{i+1}}{b'_i}$$

where $i = n-1, n-2, \ldots, 1$.

2.3 Optimizing the Thomas algorithm for a specific case

In the case presented in this project, $a_i = c_i = -1$ and $b_i = 2$. The fact that there are just two different values in the matrix greatly simplifies the calculations, since all the calculations involving only a_i, b_i and c_i can be done beforehand. Also, it is possible to derive an analytic expression for b'_i. This is given by

$$b'_i = \frac{i+1}{i}$$

and is found and proved by induction in appendix A?.

2.4 Solving the general tridiagonal matrix problem by LU-decomposition

Another way to solve matrix equations is LU-decomposing A, and then solving two matrix equations. This is a frequently used method for solving more general sets of linear equations. The LU-decomposition is a form of Gaussian elimination, where A is factored into matrices L and U. L is lower triangular with 1 in each element on the diagonal, and U is upper triangular. The matrix equation is

$$A\mathbf{u} = \mathbf{v}$$
$$LU\mathbf{u} = \mathbf{v}$$

Introducing $\mathbf{y} = U\mathbf{u}$, the solution $\mathbf{u}$ can now be computed in two steps. First, $\mathbf{y}$ is found by solving

$$L\mathbf{y} = \mathbf{v}$$

which is then inserted in LABEL

$$U\mathbf{u} = \mathbf{y}$$

and solved for $\mathbf{u}$.

2.5 Comparison of number of floating point operations for the three methods

Calculation of number of floating point operations.

2.6 Analytic solution for a specific problem

In this paper the following special case of the Poisson equation is examined:

$$-\frac{d^2u}{dx^2} = 100e^{-10x} \tag{1}$$

where $0 < x < 1$ and $u(0) = u(1) = 0$, i.e. Dirchelet boundaries. An analytic solution is obtained by integrating with respect to x twice

$$u(x) = -e^{-10x} + A + Bx$$

Imposing the boundary condition $x(0) = 0$ gives $u(0) = 0 = -1 + A$, so $A = 1$. Applying $x(1) = 0$ gives $0 = -e^{-10} + 1 + B$, so $B = e^{-10} - 1$. Inserting the integration constants returns the analytical solution

$$u(x) = 1 - (1 - e^{-10})x - e^{-10x} \tag{2}$$

where $0 < x < 1$ and $u(0) = u(1) = 0$.

3 Method

The algorithms were implemented in C++, Julia and Fortran, with the purpose of learning multiple programming languages and verifying the correctness of the implementations. The source code is available in the attatched GIT-repository. The calculations were performed several times, and for each method the algorithm times were averaged to produce a mean time. The implementations are explained and listed in the following sections with pseudocode.

3.1 The general Thomas algorithm

The general Thomas algorithm consists of the forward and backward substitutions examined in the theory section. In pseudocode the loop was implemented as

```
b_prime[0] = b[0];
v_prime[0] = v[0];

// Forward substitution
for(i = 1; i <= n-1; i++)
    b_prime[i] = b[i] - (a[i]/b_prime[i-1])*c[i-1];
    v_prime[i] = v[i] - (a[i]/b_prime[i-1])*v_prime[i-1];
}
// Backward substitution
for(i = n-2; i >= 1; i--){
    u[i] = (v_prime[i] - c[i]*u[i+1])/b_prime[i];
}
```

Listing 1: Thomas algorithm implementation for a general tridiagonal matrix

3.2 The optimized algorithm

Because of the fact that the matrix examined in this paper is both tridiagonal and symmetric, further optimizations to the algorithm can be made. Since $a_i = c_i = -1$ and $b'_i = (i+1)/i$ can be precomputed, the algorithm simplifies to

```
// b_prime is now precomputed
for(i = 1; i <= n-1; i++)
    b_prime[i] = (i+1)/i;

v_prime[0] = v[0];

// Forward substitution
for(i = 1; i <= n-1; i++)
    v_prime[i] = v[i] + (v_prime[i-1]/b_prime[i-1]);
}
// Backward substitution
for(i = n-2; i >= 1; i--){
    u[i] = (v_prime[i] + u[i+1])/b_prime[i];
}
```

Listing 2: Optimized version of Thomas algorithm for a symmetrix tridiagonal matrix

3.3 The LU-decomposition

Solving the matrix equation using the LU-decomposition method was done with functions from the Armadillo C++ Linear Algebra Library. First, the tridiagonal matrix A is implemented, and then the Armadillo functions `lu()` and `solve()` are used to solve the two factored matrix equations. In pseudocode, the implementation is

```
#include <armadillo>

for (row = 0; row < size-1; row++){
    A(row, row) = 2;
    if (row > 0)
        A(row, row-1) = -1;
    if (row < size-1)
        A(row, row+1) = -1;

lu(L,U,A);
y = solve(L, btilde);
u = solve(U, y);
```

Listing 3: Pseudocode of LU-decomposition method implementation

3.4 Error analysis

The relative error of the numerical solution at x_i is given by

$$\epsilon_i = \left| \frac{v_i - u_i}{u_i} \right|$$

where v_i is the numerically calculated value and u_i is the analytic value. The implementation of the analytic solution is rather straightforward. For several n the errors ϵ_i are calculated using the above formula, and the maximum error $\epsilon_{\mathrm{n,max}}$ is extracted, saved and later plotted against n. For obvious reasons, these calculations the optimized version of the Thomas algorithm were used.

4 Results and discussion

4.1 Solution using the algorithms

Plot osv.

4.2 Error and error analysis

Plot av feil for mange n.

4.3 Timing the algorithms

Tabell med hastigheter. kjør mange ganger, ta avg.

5 Conclusion

6 References

7 Appendix

7.1 Deriving a general expression for the diagonal elements after forward substituting

The general expression for the diagonal elements in A after forward substituting was

$$b_i' = b_i - \frac{a_i}{b_{i-1}' c_i}$$

for $i = 2, \ldots, n$. Since $a_i = c_i$ for all i in the special case examined in this paper, the expression for b_i' simplifies to

$$b_i' = b_i - \frac{1}{b_{i-1}'}$$

In an attempt at further simplify the expression, a few iterations are calculated, and a pattern seems to emerge:

$$\begin{aligned}
b_1' &= b_1 = 2 \\
b_2' &= b_2 - \frac{1}{b_1'} = 2 - \frac{1}{2} = \frac{3}{2} \\
b_3' &= b_3 - \frac{1}{b_2'} = 2 - \frac{2}{3} = \frac{4}{3} \\
b_4' &= b_4 - \frac{1}{b_3'} = 2 - \frac{3}{4} = \frac{5}{4}
\end{aligned}$$

By inspection, b_i' seems to follow the pattern

$$b_i' = \frac{i+1}{i}$$

which is assumed valid for $i = 1, 2, \ldots, n$. A proof by induction is used to show that the formula is indeed correct for all $i \geq 1$.

For any integer $i \geq 1$, let $S(i)$ denote the statement

$$S(i) : b_i' = 2 - \frac{1}{b_{i-1}'} = \frac{i+1}{i}$$

where we have defined $b_1' = 2$.

In the case $i = 1$, $S(1)$ is obviously true because b'_1 is defined as $b'_1 = b_i = 2$. Now, for some fixed $j \geq 1$, assume that $S(j)$ holds. Then, b_{j+1} is given by

$$b'_{j+1} = 2 - \frac{1}{b'_j}$$

But since $S(j)$ holds, $b'_j = \frac{j+1}{j}$, and

$$b'_{j+1} = 2 - \frac{j}{j+1} = \frac{j+2}{j+1} = \frac{(j+1)+1}{(j+1)}$$

which proves $S(i)$ is true for all $i \geq 1$.