Project 1

Øyvind E. Elgvin
09.09.19

Abstract

Effectiveness is measured of the three different algorithms for computing a Poisson equation; generalized, specialized, and LU decomposition with matrices of various sizes. Also, calculations of the relative error between the generalized algorithm and the exact solution were made.

Introduction

The overarching idea in this study is to evaluate the effectiveness of the Gaussian elimination, and specifically the Thomas algorithm [1], in a generalized, and specialized way in comparison to the LU decomposition, for computing the Poisson equation. Time is measured, and the limits for roundoff errors are explored in an attempt to map out the best alternative for the computation for the different sizes of n, in real nxn matrices.

The algorithms are programmed in c++ where the standard generalized Gaussian elimination is set up with a forward and backward substitution. The Thomas algorithm is then simplified to account for the specific case where a positive definite tridiagonal matrix is used with different sizes.

The Library, Armadillo, is used to do an LU decomposition, as an alternative version of the Gaussian elimination.

The time it took to do the three different algorithms were measured over different sizes of n and put in a table. Also, the RAM limit for doing the LU decomposition is calculated and tested.

The relative error between the general Gaussian elimination and the exact solution is measured to give a sense of the limits of roundoff errors.

The structure is going to be pretty standard with an abstract, introduction, methods, results, conclusions, appendix, and bibliography.

Theoretical models and methods

Here is the link to the github repository code:

https://github.com/jkjkjkjkjkjkjkjkjk/GitCompPhys/tree/master/Project%201

The problem is set up with a function, $f(x)$, and with an exact solution $u(x)$:

$$f(x) = 100e^{-10x}$$
$$u(x) = 1 - \left(1 - e^{-10}\right)x - e^{-10x}$$

Where b_tilde = h*h*f(i) in an equation:

$$\mathbf{Av} = \tilde{\mathbf{b}}$$

and

$$\mathbf{A} = \begin{bmatrix} b_1 & c_1 & 0 & \dots & \dots & \dots \\ a_1 & b_2 & c_2 & \dots & \dots & \dots \\ & a_2 & b_3 & c_3 & \dots & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ & & & a_{n-2} & b_{n-1} & c_{n-1} \\ & & & & a_{n-1} & b_n \end{bmatrix}$$

For the **generalized** case of the Gaussian elimination forward and backward substitution is implemented like this:

```
//forward general substitution
for (int i = 2; i < n+1; i++){
     b[i] = b[i] - a[i-1]*c[i-1] / b[i-1];
     b_tilde[i] = b_tilde[i] - b_tilde[i-1]*c[i-1] / b[i-1];
}

//backwards general substitution
v_general[n] = b_tilde[n] / b[n];
for (int i = n-1; i > 0; i--){
     v_general[i] = (b_tilde[i] - c[i]*v_general[i+1]) / b[i];
}
```

Where the vectors are defined with dynamical memory allocation and an n+2 length to seclude the endpoints from the calculations, basically to get the boundary conditions right, for example like this:

```
double* v_general = new double[n+2];
```

For the **specialized** version of the Gaussian elimination where the matrices elements are known is simplified to this:

```
// Specialised Gauss
// forward specialized substitution
for (int i = 2; i < n+1; i++){
     b_tilde_spes[i] = b_tilde_spes[i] + b_tilde_spes[i-1] /
     (b_spes[i-1]);
}

// backwards specialized substitution
v_spes[n] = b_tilde_spes[n] / b_spes[n];
for (int i = n-1; i > 0; i--){
     v_spes[i] = (b_tilde_spes[i] + v_spes[i+1]) / (b_spes[i]);
}
```

The **LU decomposition** is tried implemented via the armadillo library as follows, but errors are issued for the two last lines. Not sure what is needed for the full LU decomposition.

```
lu(L,U,A);
//y = solve( L * b_tilde );
//v_LU = solve( U * y );
```

The **time** for the different algorithms is taken by the standard c++ function clock_t:

```
clock_t start, finish;
start = clock();

// calculations here

finish = clock();
     double timeused = double (finish - start)/(CLOCKS_PER_SEC );
     cout << setiosflags(ios::showpoint | ios::uppercase);
     cout << setprecision(10) << setw(20) << "Time used for the
     general algorithm = " << timeused  << endl;
```

The **relative error** is measured by the log10 formula:

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

Roundoff error in the computer leads to error in the relative error calculation, which can be seen after 10^-5 in table nr 2.

The calculation for the LU decomposition is slower than the Thomas algorithm, but with bigger matrices, the LU decomposition takes way longer, as the table nr 1 shows. This is because the LU decomposition takes $\frac{2}{3}n^3$ FLOPS to compute.

Results and discussion

The generalized and specialized algorithms of the Gaussian elimination gives precisely the same approximation for the solution, which then gets closer and closer to the exact solution for larger and larger matrices. As shown in the methods section the generalized algorithm uses 9n FLOPS and the specialized uses 4n FLOPS.

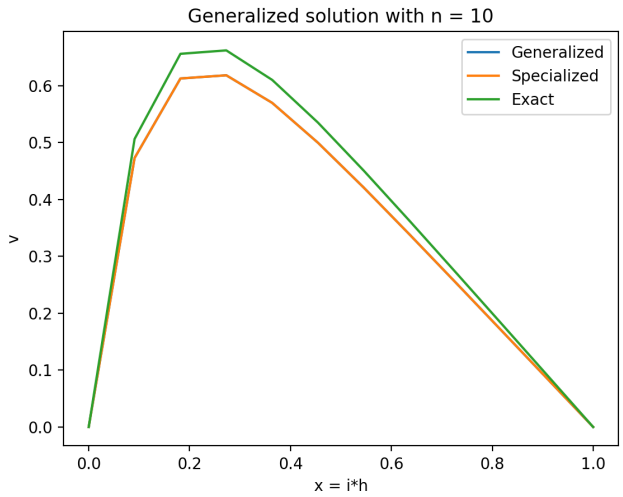

Figure 1: Generalized, specialized, and exact solution of the Gaussian elimination with n = 10

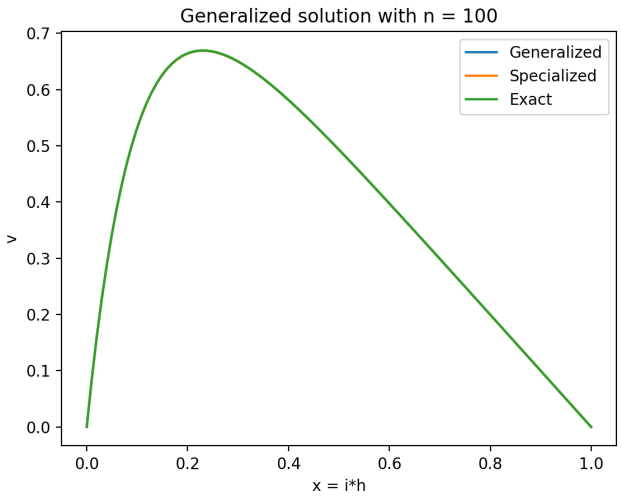

Figure 2: Generalized, specialized, and exact solution of the Gaussian elimination with n = 100

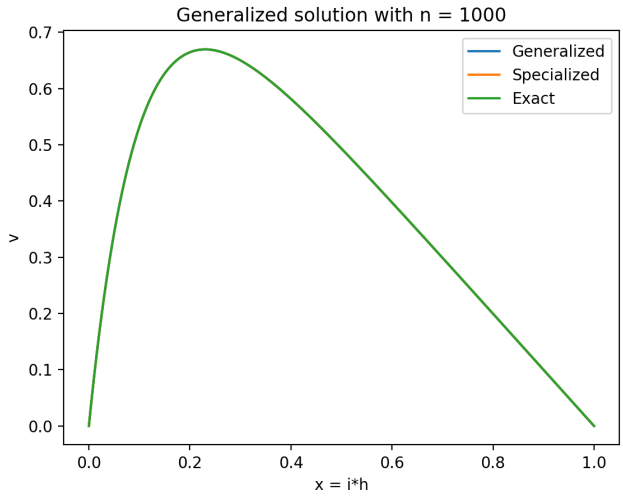

Figure 3: Generalized, specialized, and exact solution of the Gaussian elimination with n = 1000

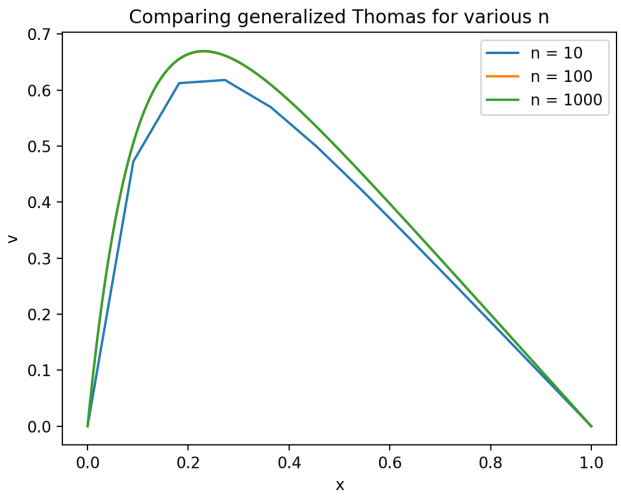

Figure 3: Generalized, specialized, and exact solution of the Gaussian elimination with n = 1000

The specialized version of the Gaussian elimination is faster, which the values measured shows:

Matrices nxn	Generalized	Specialized	LU decomposition
n = 10	2,000000000E–06	1,000000000E–06	4,300000000E–05
n = 100	5,000000000E–06	1,000000000E–06	0,0009060000000
n = 1000	2,100000000E–05	2,000000000E–05	0,06595000000
n = 1e4	0,0002280000000	0,0002110000000	38,30249400
n = 1e5	0,002921000000	0,002168000000	–
n = 1e6	0,02653600000	0,02180500000	–

Table nr 1: overview of the measured time each algorithm used.

The LU decomposition, on the other hand, takes a longer time than the Thomas algorithm, as seen in table 1. This is because of the larger number of flops the LU needs to compute, which is

$$\frac{2}{3}n^3 \text{ Flops}$$

The LU decomposition works up the point of n = 10.000, and after that, the RAM which is needed is 10^5 x 10^5 x 8 bytes = 80 x 1000^3 bytes = 80 Gb, which most commercial PC's don't have.

The relative error is measured and presented in table nr 2.

Matrices nxn	Relative error
n = 10	–1,179697
n = 100	–3,088036
n = 1000	–5,080051
n = 1e4	–7,079356
n = 1e5	–9,004896
n = 1e6	–6,771379
n = 1e7	–13,007

Table nr 2: Relative error for n from 10 to 1e7

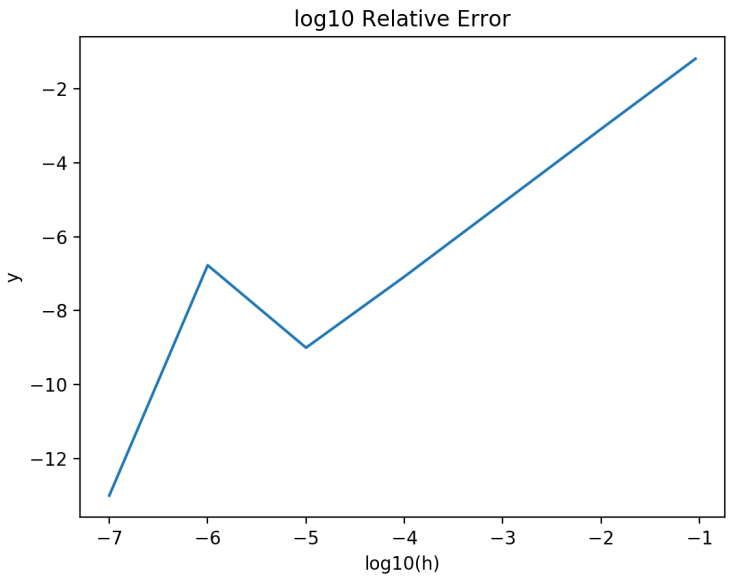

Figure nr 4: plot of the relative error

Conclusion and perspective

The main findings in this study are that there are indeed different time lengths for the different algorithms in cord with the number of flops each algorithm needs. The specialized Thomas algorithm is the fastest, and the LU decomposition is the slowest just as the number of flops would predict.

For future work, I would suggest checking the implementation of the LU decomposition, and to explore other algorithms as well to see if anyone can beat the timings of the specialized Thomas algorithm.

Appendix

Here I present the way which leads to the algorithms. The approximation of the second derivation can be written as follows:

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

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

The above6 is a linear combination where you can see it as a multiplication of two matrixes, one with numbers, $\hat{\mathbf{T}}$, and one with *u*'s, $\hat{u}$, like this:

$$\hat{\mathbf{T}}\hat{u} = h^2\hat{f}_i$$

Which can be written as:

$$\mathbf{A}v = \tilde{b}$$

where $\tilde{b} = h^2\hat{f}_i$.

The Poisson equation reads that

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

where

$$f(x) = 100e^{-10x}$$

and

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

So if $u(x)$ gets derivated twice you get:

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

$$u'(x) = 10e^{-10x} - e^{-10} - 1$$

$$u''(x) = -100e^{-10x}$$

and:

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

Bibliography

[1]

Nicholas J. Higham (2002). *Accuracy and Stability of Numerical Algorithms: Second Edition*. SIAM. p. 175. ISBN 978-0-89871-802-7.