COMPUTING FINITE DIFFERENCES USING THE THOMAS ALGORITHM AND LU-DECOMPOSITION

Ole Halvor Egenæs, Marius Jonsson

September 18, 2016

ABSTRACT

This is a report submission for the first project of «Computational physics» at the Institute of Physics, University of Oslo, autumn 2016.

In our study we used the finite difference method to reduce an ordinary differential equation (ODE) to a linear algebra problem. We used the Thomas algorithm and LU-decomposition to find a solution. Thereafter we explored numerical precision, stability, numerical error for the two methods. We found that the Thomas algorithm complexity scales as $\mathcal{O}(n)$ while the LU method is $(2/3)n^3 + \mathcal{O}(n^2)$. Thus, the Thomas algorithm is fast but the LU-decomposition method is flexible. In our studies we found that the two algorithms perform similarly in other respects.

INTRODUCTION

The purpose of the project is to explore algorithms for efficiently solving the linear algebra problem induced by a typical inhomogeneous ODE. More specifically, we will solve the one-dimensional poission equation:

$$-\frac{\mathrm{d}^2}{\mathrm{d}x^2}v(x) = f(x). \tag{1}$$

We will use an algorithm which is based on linear algebra methods, called the Thomas Algorithm (after British physicist Llewellyn Thomas). We start by finding an approximation to the solution in terms of a set of linear equations. You will see that the method is closely related to the common iterative methods that we have seen in previous courses, such as the «Euler method», «Euler-Cromer» and «Runge-Kutta». The main difference is in how we evaluate the finite differences generated by the algorithms. The Thomas algorithm does this by a particularly smart Gauss elimination-process. Therefore, this report is not only of interest to those whom want to solve ODEs, you will see that the Thomas algorithm is efficient in solving a whole class of linear algebra problems, such as the computation of cubic splines (Hjorth-Jensen 2015, p. 189). The algorithm will be further specialized for the particular ODE to improve runtime by using fewer floating point operations (FLOPs). We will call this the specialized Thomas algorithm or specialized algorithm. Eventually, we will discover that both the Thomas algorithm and specialized algorithm is more efficient than an alternative provided by the Armadillo library which we will call the LU-decomposition-method or LU-method.

The ODE we will attempt to solve is the one-dimensional Poisson equation with Dirichlet boundary conditions $u(0) = 0$ and $u(1) = 0$ on the interval $[0, 1]$. We could in principle, as you will see, derive other algorithms based on similar principles for a variety of other ODEs. However, this would require changing the Thomas algorithm completely, and therefore this method is somewhat less flexible than the traditional

methods.

The report is structured by an introduction followed by «methods»-, «results and discussion»-sections. Then we summarize our findings and thoughts in the «conclusion and perspectives»-section.

METHODS

We begin by partitioning $[0, 1]$ into exactly n intervals. We denote these by $[x_j, x_{j+1}]$ and let

$$x_j = j/n \text{ for all } j \in \{0, 1, \cdots, n\} \equiv S_n \qquad \text{only if} \qquad [0, 1] = \bigcup_{i=0}^{n} [x_i, x_{i+1}].$$

We define $f_i = f(x_i)$ and similarly $v_i = v(x_i)$. Suppose we define $h = 1/n$, then we can write the one-dimensional poisson equation using the three-point finite difference:

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} + \mathcal{O}(h^2) = f_i \qquad \text{for all } i \in S_n \tag{2}$$

Suppose we ignore the term $\mathcal{O}(h^2)$ and multiply though by h^2: Then this is clearly a linear set of equations. We equivalently rewrite this as a tridiagonal $k \times k$-matrix A, and vectors $\mathbf{v}$ and $\mathbf{f}$ such that $A\mathbf{v} = \mathbf{f}$ by letting

$$A = \begin{bmatrix} a_1 & b_1 & & 0 & & 0 \\ c_1 & a_2 & b_2 & & 0 & \\ & c_2 & a_3 & b_3 & & 0 \\ 0 & & \ddots & \ddots & \ddots & \\ & 0 & & c_{k-2} & a_{k-1} & b_{k-1} \\ 0 & & 0 & & c_{k-1} & a_k \end{bmatrix}, \qquad \begin{aligned} a_{ij} &= 2 \\ b_{ij} &= -1 \\ c_{ij} &= -1 \end{aligned} \qquad \mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{pmatrix}, \qquad \mathbf{f} = h^2 \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_k \end{pmatrix}.$$

As we explained, the Thomas algorithm works by using Gauss elimination. If you attempt to write down the equations describing the process, you would in the special case of a tridiagonal matrices arrive at the following formulas:

$$\widetilde{a}_i = a_i - \frac{c_{i-1}b_{i-1}}{\widetilde{a}_{i-1}}, \quad \widetilde{f}_i = f_i - \frac{c_{i-1}}{\widetilde{a}_{i-1}} f_{i-1} \quad v_{i-1} = \frac{\widetilde{f}_{i-1} - b_{i-1}}{\widetilde{a}_{i-1}}. \quad \text{(Hjorth-Jensen 2015, pp. 170-173)}$$

The last two expressions simplify for $a_{ij} = 2, b_{ij} = -1, c_{ij} = -1$ to:

$$\widetilde{f}_i = f_i + f_{i-1}\frac{i}{i+1}, \qquad v_i = \left(\frac{i}{i+1}\right)(u_{i+1} + \widetilde{f}_i).$$

We will refer to computation of $\widetilde{a}_i$ and $\widetilde{f}_i$ as the forward substitution (from Gauss-elimination) and computation of v_i as backward substitution. Thus it was straightforward to implement the algorithm in C++ by

```
// forward substitution

for(int i = 1; i<n; i++){
       double p = c[i-1]/a[i-1];
       a[i] = a[i]-b[i-1]*p;
       f[i] = f[i]-f[i-1]*p;
}
.
```

```
// backward substitution

v[0] = 0;
v[n+1] = 0;
v[n] = f[n-1]/a[n-1];
for(int i = n-1; i > 0; i--){
       v[i] = (f[i-1]-b[i-1]*v[i+1])/a[i-1];
}
```

Alternatively, we can use LU-decomposition to solve the set of equations. This method not only works for tridiagonal matrices, but for any non-singular matrix A. LU-decomposition allows us to rewrite $A = LU$, for two triangular matrices L and U such that $\det L = 1$. After computing L and U explicitly, one can reduce the problem to two simple sets of equations. To see this, observe that there exist an $\mathbf{y} \in \mathbb{R}^k$ such that $L\mathbf{y} = \mathbf{f}$. Since L^{-1} exists by $\det L = 1$, we can write:

$$U\mathbf{v} = L^{-1}\mathbf{f} = L^{-1}L\mathbf{y} = \mathbf{y}. \tag{3}$$

This equation along with $L\mathbf{y} = \mathbf{f}$ are triangular, which we can solve quickly in the next section. The implementation in C++ requires few lines of code. It is sufficient to write `lu(L,U,A); y = solve(L,f); v = solve(U,y);` to complete the whole procedure.

We made effort to check that the algorithms produce the results we expect. In our case, we let $f(x) = 100e^{-10x}$. You can check that $u(x) = 1-(1-e^{-10})x-e^{-10x}$ then becomes a solution by insertion. We will confirm that for any practical purpose, the numerical solution converges uniformly to u on $[0, 1]$. Readers are free to check this by downloading and compiling the source code at `http://github.com/oleheg1/Fys3150`. We will discuss the convergence of the solution along with other performance considerations in the following results.

RESULTS AND DISCUSSION

By computation, we found that the complexity of the Thomas-algorithm is $\mathcal{O}(n)$ and the LU-algorithm $(2/3)n^3 + \mathcal{O}(n^2)$. This was also confirmed by literature (Hjorth-Jensen 2015, p. 173) (Golub and Van Loan 1996, p. 98). We attempted to find a lower bound on the step length which we could practically compute with the LU-method. The method succeeded for $h = 10^{-4}$, but failed for $h = 10^{-5}$ since the required memory for double precision floating point format is exactly 80GB. For the Thomas algorithm however, the order of magnitude for the memory is a megabyte, and the procedure completed in less than a millisecond on our ordinary desktop computer. To be more specific, we've produced a table of the run-times for the two algorithms, discriminating between the Thomas algorithm and the specialized Thomas algorithm. Please see table 1.

Next, we investigated what could be said about the error induced by the numerical approximation. In short, we found that the error for the Thomas algorithm and LU-method was almost identical for all $10^{-4} \leq h \leq 1$. It was impossible to compare the methods for smaller step lengths because of insufficient memory to store the matrices required for the LU-method. To be more precise, suppose we estimate v_i by $\widehat{v}_i$, then the absolute error, e_n, and relative error, ε_n, are given by

$$e_n = \max\left\{|\widehat{v}_i - v_i| \ : \ i \in \{1, 2, \cdots, n\}\right\}, \qquad \text{and} \qquad \varepsilon_n = \max\left\{\left|\frac{\widehat{v}_i - v_i}{v_i}\right| \ : \ i \in \{0, 1, 2, \cdots, n\}\right\},$$

and the values of e_n are given in table 1 for all three methods. In addition, for the Thomas algorithm we plotted the relative error, ε_n and the numerical, and exact solutions overlaid. These plots are contained in figure 1.

It seems natural to explore whether one method is stable while the other is not. In general, the Thomas algorithm and LU decomposition methods are not stable. However they are stable if either (i) A is symmetric positive definite, (ii) A is totally non-negative, (iii) L, U have strictly positive diagonal elements and negative or zero off-diagonal elements(Higham 2002, pp. 175-176). In our case, the latter is the applies, and thus both the LU-decomposition- and Thomas-algorithm is stable. Since we've now presented all of our results, we are ready for the discussion.

Method	$h = 10^{-2}$	$h = 10^{-3}$	$h = 10^{-4}$
Thomas (s)	$2 \cdot 10^{-6}$	$4.1 \cdot 10^{-5}$	$4.1 \cdot 10^{-4}$
Special (s)	$2 \cdot 10^{-6}$	$1.5 \cdot 10^{-5}$	$1.7 \cdot 10^{-4}$
LU (s)	$2.2 \cdot 10^{-3}$	$1.5 \cdot 10^{-1}$	$2.2 \cdot 10^{0}$

n	h	Thomas	LU
10^1	10^{-1}	$4.4 \cdot 10^{-2}$	$4.4 \cdot 10^{-2}$
10^2	10^{-2}	$5.5 \cdot 10^{-4}$	$5.5 \cdot 10^{-4}$
10^3	10^{-3}	$5.6 \cdot 10^{-6}$	$5.6 \cdot 10^{-6}$
10^4	10^{-4}	$5.6 \cdot 10^{-8}$	$5.6 \cdot 10^{-8}$

Table 1: Left: Table of run times for the Thomas algorithm, specialised Thomas algorithm, and the LU-method measured in seconds (s). Right: Table of the absolute error e_n for the Thomas algorithm and LU-decomposition method as a function of number of intervals n and step length, h.

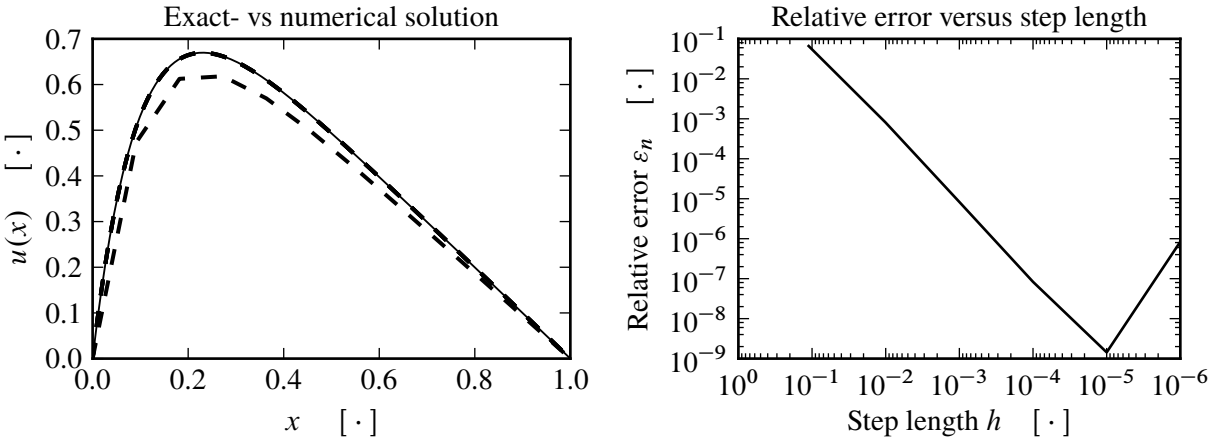

Figure 1: Left: Comparison of numerical solution relative to the exact solution for our function u. The exact solution is plotted in solid curve. While the numerical solutions are dashed for $h = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}$. One sees from the plot that for values of $h < 10^{-2}$, it is impossible to discriminate between the numerical solutions at this y-scale. Right: Plot of the relative error ε_n for step lengths $h = 10^{-j}$, $j \in \{1, 2, 3, 4\}$ using the Thomas algorithm. Computation of the relative error support the hypothesis that the relative error is mainly due to the finite difference approximation, and that other mechanisms dominate for $h < 10^{-5}$.

Our results show that the total absolute error for both methods is $\mathcal{O}(h^2)$. This is in accordance with equation 2, supporting the hypothesis that the error due to the approximation we make by rewriting the ODE as a finite difference. However, it's still possible that the error is due to other mechanisms. This is because we do not have total control over all the contributions to the error. For example, it could be possible that other errors are systematically cancelling, giving a false impression that the error is due to the finite difference-approximation. But we note that the error is identical for both methods for step-lengths larger that 10^{-4}. Since the arithmetic required for the two methods is vastly different, this is almost conclusive evidence that the error is not due to the numerical arithmetic. And strengthens our belief that the error is due to the finite difference approximation. Since the errors are identical for both methods, there is no reason to favour one over the other.

As we noted, the required memory for the Thomas algorithm is approximately the square-root of the memory required for the LU-decomposition method. We saw that for sufficiently small step lengths, the LU-method is almost useless. This is where the Thomas algorithm may be useful: Due to the low memory consumption and low complexity, it is difficult to imagine a more efficient algorithm for the purpose. Since it was possible to run the Thomas algorithm for $h < 10^{-6}$, we discovered that as the step length decreased, other mechanisms than the finite difference-approximation began to dominate. Se figure 1 for an illustration. It was impossible to discover these mechanisms using the LU-method, because almost any computer has insufficient memory to complete the computation.

For most practical ODE solving, the Thomas algorithm is clearly relatively specialized. This is because users must necessarily convert the set of equations to a tridiagonal matrix to solve the problem at hand. In addition, to guarantee that the solution is stable, we were only able to find a small number of sufficient criteria to ensure stability. This is also a a problem of the LU-method. However, to solve a system of equations using the LU-method, it is convenient that the LU method only requires that the set of equations are non-singular. But due to the high cost of human labour, for most purposes, the versatility of the LU-method will possibly overshadow the computational advantages, particularly in solving ODEs.

CONCLUSIONS AND PERSPECTIVES

In solving ODEs, we've found no evident reason to prefer the Thomas algorithm, the specialized Thomas algorithm or LU-method in terms of stability or avoidance of computational errors. However, we have seen that for certain linear algebra problems, especially Gauss-elimination of tridiagonal matrices (with specifications ensuring stability), the Thomas algorithm has a computational advantage which is probably difficult to beat by any method. In particular, the complexity of the LU-method is two orders higher, and can therefore not match the Thomas algorithm. Some could argue that there is an added benefit of LU decomposition, in that we get a simple formula for the determinant. However, in solving ODEs, this information has little use, especially since LU decomposition requires that the derminant is nonzero at the outset. In contrast, whenever we can reduce the problem at hand to a set of equations which is not tridiagonal, the LU method wins. As we have seen, the Thomas algorithm only deals with a very small class of matrices. And if one looks beyond the solutions of ODEs, there is certainly situations where the added information gathered by LU-decomposition is useful for theoretical purposes. Furthermore, the LU-method will possibly be the method of choice in situations where both the Thomas algorithm and LU-method are applicable. This is due to the flexibility that the LU-method has, which could be critical when researchers cannot rule out the possibility that the complexity of the project may increase beyond the scope of the Thomas algorithm.

In closing, we're asked to give constructive criticism to the course administration regarding the organization of project 1. At the time of writing, the authors believe that the design and execution of project 1 was beneficial to almost all students, and we have no further comments at this time. This concludes our report for project 1.

LITERATURE CITED

[1] Gene H. Golub and Charles Van Loan. *Matrix Computations*. 3rd ed. London: The John Hopkins University press, 1996.

[2] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. 2nd ed. Philadelphia: Society for Industrial and Applied Mathematics, 2002.

[3] Morten Hjorth-Jensen. *Computational Physics*. Oslo: University of Oslo, Aug. 2015.