Project 1 - FYS3150 Computational Physics

Fredrik Østrem (`fredost`)

September 19, 2016

Contents

1 Introduction

In this project, we solve the one-dimensional Possion equation

$$-u''(x) = f(x) \tag{1}$$

in the case where $u(0) = u(1) = 0$ and $f(x) = 100e^{-10x}$. We shall set up a matrix equation

$$A\mathbf{v} = \mathbf{b} \tag{2}$$

and generate efficient algorithms for solving this equation to get a numerical solution to the differential equaion. We shall also compare these algorithms to LU decomposition, and see how much more efficient the specialized algorithms are.

2 Methods

2.1 Approximate solution to $-u''(x) = f(x)$

We consider a differential equation on the form $-u''(x) = f(x)$ with $x \in (0, 1)$ and $u(0) = u(1) = 0$, and where f is a known function.

We consider the functions u and f at a sequence of equally spaces points $x_0, x_1, \ldots, x_n$ in the interval $[0, 1]$, so that $x_i = ih$ where $h = \frac{1}{n+1}$. We let $v_i = \tilde{u}(x_i) \approx u(x_i)$ be the value of our approximate solution at x_i, and let $b_i = h^2 f(x_i)$. Since $u(0) = u(1)$, we have the boundary condition $v_0 = v_{n+1} = 0$.

We can approximate $u''(x)$ by using the Taylor's expansion of $u(x)$, which yields:

$$u''(x_i) \approx \frac{u(x_i + h) + u(x_i - h) - 2u(x_i)}{h^2} \approx \frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} \tag{3}$$

which, when subsituted into our differential equation gives

$$2v_i - v_{i+1} - v_{i-1} = b_i \tag{4}$$

Since this is a linear equation with v_{i-1}, v_i, v_{i+1} as unknowns, we can write this as

$$\underbrace{\begin{pmatrix} 0 & 0 & \cdots & -1 & 2 & -1 & \cdots & 0 & 0 \end{pmatrix}}_{\mathbf{a}_i} \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{i-1} \\ v_i \\ v_{i+1} \\ \vdots \\ v_{n-1} \\ v_n \end{pmatrix} = b_i \tag{5}$$

We can do this for every index i, and $\mathbf{a}_i$ is shifted a single step to the right compared to $\mathbf{a}_{i-1}$. Therefore, when we look at all values of $i = 1, \ldots, n$, we get the tridiagonal matrix A:

$$A = \begin{pmatrix} \mathbf{a}_0 \\ \vdots \\ \mathbf{a}_{i-1} \\ \mathbf{a}_i \\ \mathbf{a}_{i+1} \\ \vdots \\ \mathbf{a}_n \end{pmatrix} = \begin{pmatrix} 2 & -1 & 0 & \cdots & & 0 \\ -1 & 2 & -1 & 0 & \iddots & \\ 0 & -1 & 2 & \ddots & 0 & \vdots \\ \vdots & 0 & \ddots & 2 & -1 & 0 \\ & \iddots & 0 & -1 & 2 & -1 \\ 0 & & \cdots & 0 & -1 & 2 \end{pmatrix} \tag{6}$$

such that $A\mathbf{v} = \mathbf{b}$.

2.2 General algorithm for solving tridiagonal matrix equations

In general, we can write an $n \times n$ tridiagonal matrix A as

$$A = \begin{pmatrix} b_1 & c_1 & 0 & \dots & \dots & \dots \\ a_1 & b_2 & c_2 & \dots & \dots & \dots \\ & a_2 & b_3 & c_3 & \dots & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ & & & a_{n-2} & b_{n-1} & c_{n-1} \\ & & & & a_{n-1} & b_n \end{pmatrix} \tag{7}$$

We can solve the matrix equation $A\mathbf{v} = \mathbf{b}$ in three steps:

1. Eliminate the lower diagonal $a_1, a_2, \dots, a_{n-1}$ through forward substitution.
   - $a'_{i-1} = 0$; $b'_i = b_i - \frac{a_{i-1}}{b'_{i-1}} \cdot c_{i-1}$; $f'_i = f_i - \frac{a_{i-1}}{b'_{i-1}} \cdot f'_{i-1}$
2. Eliminate the upper diagonal $c_1, c_2, \dots, c_{n-1}$ through backward substitution.
   - $c''_i = 0$; $f''_i = f'_i - \frac{c_i}{b'_{i+1}} \cdot f''_{i+1}$
3. Divide each row i by b_i to get only 1 elements along the main diagonal.
   - $v_i = \frac{f''_i}{b'_i}$

When doing these steps, we transform A into the identity matrix, and transform $\mathbf{b}$ into $\mathbf{v}$. (This algorithm has been implemented in the file `oppg_b.cc`.) We can see the results of the approximation in figure 1.

From the implementation, we can see that the first step uses 5 floating-point operations for every row except the first, the second step uses 5 for every row except the last, and the third step uses 2 for every row; in total, the number of floating-point operations is $5(N-1) + 5(N-1) + 2N = 12N - 10$.

2.3 Improved algorithm for constant diagonals

We can improve

3 Results

We run the algorithm from section 2.2 from different values of n. In figure 1 we see that for small values of N, the curve keeps the right shape but is quite a bit off from the analytic solution; however, for $N = 1000$ it is almost indistinguishable from the analytic solution. In table 2, where we have listed the maximum value of

$$\varepsilon_i = \log_{10} \left| \frac{v_i - u_i}{u_i} \right| \tag{8}$$

for each N, as a function of $\log_{10}(h)$ where h is the step length, we see that the relative error becomes decreases exponentially as h decreases.

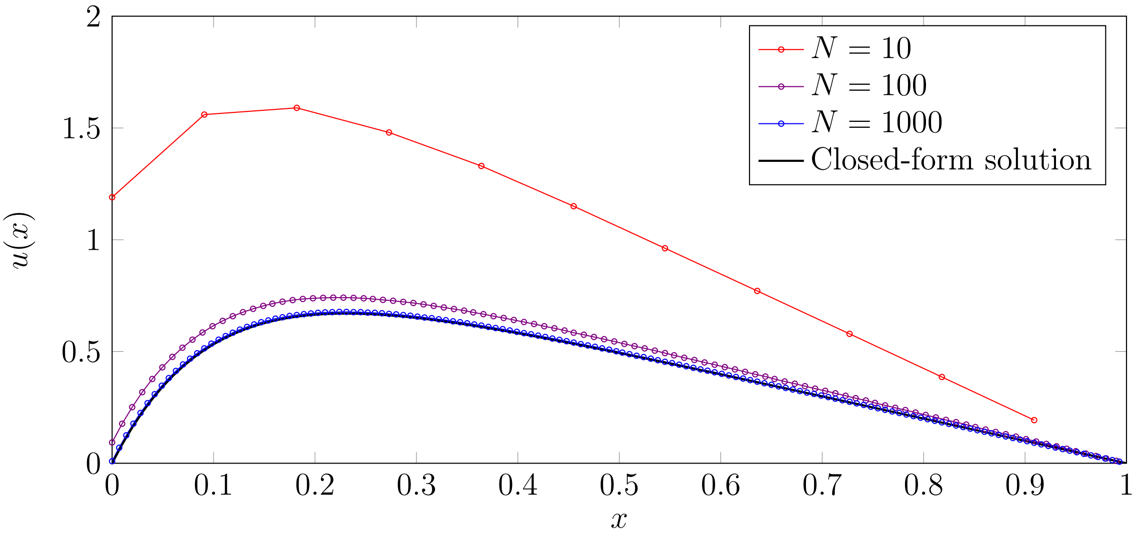

Figure 1: Plot of approximate solutions based on the method described in section 2.2

$\log_{10}(h)$	ε_{max}
-1	$3.17 \cdot 10^{-1}$
-2	$3.85 \cdot 10^{-2}$
-3	$3.95 \cdot 10^{-3}$
-4	$3.96 \cdot 10^{-4}$
-5	$3.96 \cdot 10^{-5}$
-6	$4.10 \cdot 10^{-6}$
-7	$1.34 \cdot 10^{-6}$

Table 1

Table 2: $\varepsilon_{\max}$ as a function of $\log_{10}(h)$.

If we compare the runtime of our tridiagonal algorithm to one using LU decomposition of the matrix (the implementation given in the course's code files), we see that

N	Time (tridiagonal) / ms	Time (LU decomposition) / ms
10	0.001	0.008
100	0.005	2.062
1,000	0.051	2 009.920

Table 3: Running times for our specialized algorithm, and a LU decomposition algorithm.

the tridiagonal algorithm is very fast, even for large N, while the LU decomposition runs at about 1 s for $N = 1000$:

There's a big difference here because our simple algorithm for tridiagonal matrices only needs on the order of N float-point operations, so it run in less than a millisecond, even for very large matrices. However, LU decomposition is much more general, and needs on the order of N^3 different floating-point operations for each matrix. If N is very large, the number of operations grows very fast.

4 Conclusions

From the results we got, we can conclude that numerically solving a differential equation by using a matrix equation can work well, but we a small enough step length ($N = 10$ is way off, for instance) and a sufficiently efficient algorithm for solving the matrix equation (LU decomposition is too slow for large N).

A Appendix

All files used in this project can be found at `https://github.com/frxstrem/fys3150/tree/master/project1`. The following code files are used:

- `oppg_b.cc`
- `oppg_d.cc`
- `oppg_e.cc`