Fys4150 Project 1

Aksel Graneng

September 10, 2019

Abstract

enter abstract here

1 Introduction

2 Theory

2.1 Vectorized second derivative

If we have a 1d data-set on the form:

$$V(\vec{x}) = [v_0,\ v_1,\ \cdots,\ v_{n-1},\ v_n,\ v_{n-1},\ \cdots]$$

Then we can write the second derivative of the data-set as:

$$f_n = -\frac{v_{n+1} + v_{n-1} - 2v_n}{\Delta x^2}$$

Where Δx is the change in variable we are derivating based on; usually time.

Rather than calculating the second derivatives of this data-set individually, we can instead calculate them all at the same time using linear algebra. This can be done by finding a matrix A such that

$$A\vec{V} = \vec{f}$$

Where:

$$\vec{f} = \begin{bmatrix} 2v_i - v_2 \\ -v_1 + 2v_2 - v_3 \\ \vdots \\ -v_{n-1} + 2v_n - v_{n+1} \\ -v_n + 2v_{n+1} - v_{n+2} \\ \vdots \end{bmatrix}$$

As multiplying $\vec{f}$ with $\frac{1}{\Delta x^2}$ would give us an array containing all the second derivatives. We can see that $\mathbf{A}$ must be:

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & 0 & \cdots \\ -1 & 2 & -1 & 0 & \cdots \\ 0 & -1 & 2 & -1 & \cdots \\ \vdots & 0 & -1 & 2 & -1 \end{bmatrix}$$

3 Method

3.1 General linear equation solver

The first thing our program does is solve a set of general linear equations on the form $\mathbf{Av} = \mathbf{f}$. This is done by putting the matrix $\mathbf{M}$ on row reduced echeleon form:

$$\mathbf{M} = \begin{bmatrix} b_1 & c_1 & 0 & 0 & \cdots f_1 \\ a_2 & b_2 & c_2 & 0 & \cdots f_2 \\ 0 & a_3 & b_3 & b_3 & \cdots f_3 \\ \vdots & 0 & a_4 & b_4 & \cdots f_4 \end{bmatrix}$$

Where $\mathbf{M}$ is a $n \times (n+1)$ matrix. This is generally quite simple to solve using forward and backward substitution, but when n becomes very large (in my experience, larger than 10^5) we get problems with memory as well as computation time, as the most simple row reduction algorithm has $2(n+1)n^2$ FLOPS.
To avoid this problem, we use the fact that most of the elements in the matrix is zero. In fact, there are at most 4 elements in each row that isnt zero. By removing every element that is zero from the matrix (except one in for the first and last line), we get:

$$\mathbf{M}^* = \begin{bmatrix} b_1 & c_1 & 0 & f_1 \\ a_2 & b_2 & c_2 & f_2 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & a_n & b_n & f_n \end{bmatrix}$$

So now we have a $n \times 4$ matrix which contains all the information we need.

We now use forward substitution on $\mathbf{M}^*$ just like we would on $\mathbf{M}$, except this time we "roll" the next row every time we subtract, so that b_i is subtracted from a_{i+1} and c_i is subtracted from b_{i+1}. At the same time, we are careful to subtract f_i from f_{i+1}. This means that index notation must be used, rather than numpy.roll.
This is done in a loop that has n iterations. First it does 3 operations, one on each element in the row except for the first (which is zero), along with a division, to normalize it. Then it does 3 subtraction and multiplication operations on the next row.

Next is backwards substitution. Again we use a loop with n iterations. Here we only do 2 subtractions and multiplications, on the last and second last element of each row.
All in all, this sums up to 14 FLOPS.

After it was done, the general linear equation solver was tested on:

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

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

Which has a closed-form solution:

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

3.2 Specific linear equation solver

The matrix we are working with in this project is very simple. All values of a, all values of b and all values of c are equal. That is to say:

$$\text{for any } i \wedge j \in [1,\ n],\ a_i = a_j,\ b_i = b_j,\ c_i = c_j$$

This means that we are practically doing the same operations n times when we are row reducing our matrix. Rather than doing this, we can limit our

algorithm to only operate on our values for f, as these are unknown. We simply need to look at what the row reduction algorithm does to the matrix to see what it does to f, and then skip right to that part when calculating it numerically.

Studying the forward substitution, we see, starting with $n = 2$:

$$f_n^* = \frac{n}{n+1}(f_n - f_{n-1}^*)$$

Thus, the n'th row ends up as (excluding all the zeros):

$$[0,\ 1,\ -\frac{n}{n+1},\ f_n^*]$$

The very last row ends up as (with $n =$ size of f):

$$[0,\ 0,\ 1,\ -(f_{-1} - f_{-2}^*)\frac{n}{n+1}]$$

The first value of f (n = 1) ends up as:

$$f_1^* = \frac{1}{2}f_1$$

And then we look at forward substitution. We can see that we subtract:

$$\frac{n-i}{n-i-1}f_{-(i-1)}^{**}$$

from f_{-i}^* to find our final value f_{-i}^{**}
The process of forward substitution ends with the last line being:

$$f_1^* = f_1^* + 2f_2^{**}$$

4 Results

4.1 General linear equation solver

We tested the general linear equation solver for datapoints $n = 10,\ n = 100,\ n = 1000$

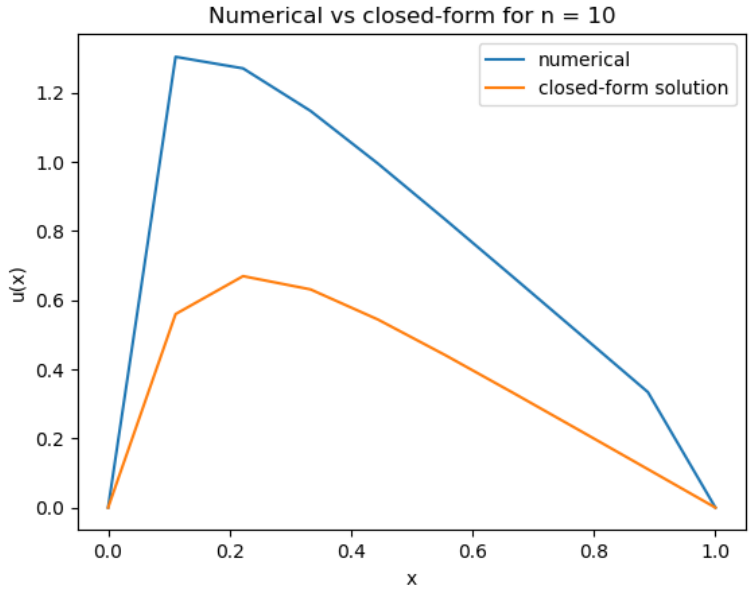

Figure 1: The numerical solution vs the closed-form solution for 10 data-points. Here we can see that they deviate massively.

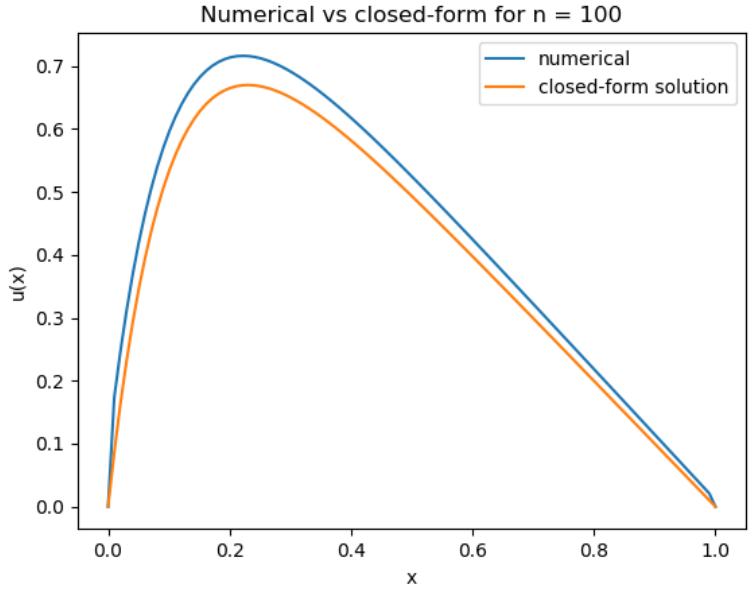

Figure 2: The numerical solution vs the closed-form solution for 100 data-points. Here we can see that the difference is decreasing.

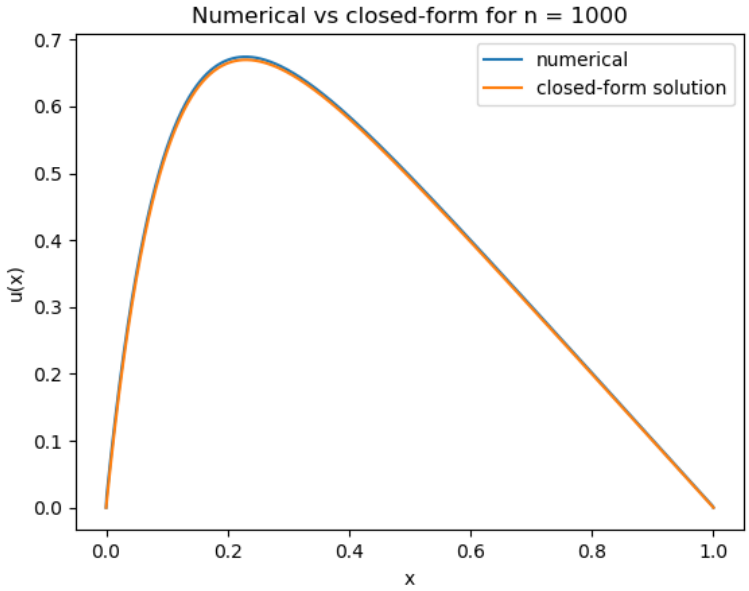

Figure 3: The numerical solution vs the closed-form solution for 100 data-points. Here we can see that the difference has been even more decreased, and it is becoming hard to tell them apart.

5 Discussion

6 Conclusion