FYS3150 - Project 4

Steffen Brask

9. november 2014

https://github.com/steffemb/project4/tree/master/Levering

Abstract

In this project we will test the numerical stability of thre different algorithms for solving differential equations.

Introduction

In this project we are going to test the stability and errors of thre methods for solving partial differential equations. We will write a program that solves the diffusion equation with the explicit and implicit Euler scheme, and last the Crank Nichols scheme.

the equation to solve is:

$$\frac{\partial u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t} \tag{1}$$

with initial conditions $u(x,0) = 0, 0 < x < d$, and $u(0,t) = 1, t > 0$, and $u(d,t) = 0, t > 0$

Solution

First we write a code for the explicit Euler scheme. Mathematically this is written:

$$\frac{u(x_i, t_j + \Delta t) - u(x_i, t_j)}{\Delta t} = \frac{u(x_i + \Delta x, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j)}{\Delta x^2}. \tag{2}$$

This is fairly easy to write an algorithm for, we only need to solve for $t_j + \Delta t$, and we get the equation:

$$u(x_i, t_j + \Delta t) = u(x_i, t_j) + \frac{\Delta t}{\Delta x^2} u(x_i + \Delta x, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j). \tag{3}$$

The algoritm now only needs a for loop over j, and i, which is the time and space indexes. Note that the i indexes must exlude the first and last indexes.

The second task is to implement the Implicit Euler scheme. We startwith the mathematical expression:

$$\frac{u(x_i,t_j)-u(x_i,t_j-\Delta t)}{\Delta t}=\frac{u(x_i+\Delta x,t_j)-2u(x_i,t_j)+u(x_i-\Delta x,t_j)}{\Delta x^2}. \tag{4}$$

This time we solve for $t_j-\Delta t$, and we get:

$$u(x_i,t_j-\Delta t)=u(x_i,t_j)-\frac{\Delta t}{\Delta x^2}u(x_i+\Delta x,t_j)-2u(x_i,t_j)+u(x_i-\Delta x,t_j) \tag{5}$$

Lets define $\alpha=\frac{\Delta t}{\Delta x^2}$. And rewrite the equation as a linear equation.

$$u(x_i,t_j-\Delta t)=u(x_i,t_j)-\alpha\hat{A}u(x_i,t_j), \tag{6}$$

where $\hat{A}$ is the matrix:

$$\mathbf{A}=\begin{pmatrix} -2 & 1 & 0 & \dots & \dots & 0 \\ 1 & -2 & 1 & 0 & \dots & \dots \\ 0 & 1 & -2 & 1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & 1 & -2 & 1 \\ 0 & \dots & & 0 & 1 & -2 \end{pmatrix} \tag{7}$$

further:

$$u(x_i,t_j-\Delta t)=u(x_i,t_j)(\hat{I}-\alpha\hat{A}), \tag{8}$$

And we can define:

$$\hat{B}=(\hat{I}-\alpha\hat{A}) \tag{9}$$

$$\mathbf{B}=\begin{pmatrix} 1+2\alpha & -\alpha & 0 & \dots & \dots & 0 \\ -\alpha & 1+2\alpha & -\alpha & 0 & \dots & \dots \\ 0 & -\alpha & 1+2\alpha & -\alpha & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -\alpha & 1+2\alpha & -\alpha \\ 0 & \dots & & 0 & -\alpha & 1+2\alpha \end{pmatrix} \tag{10}$$

Now scince $\hat{B} = \hat{B}^{-1}$, $u(x_i, t_j) = \hat{B}u(x_i, t_j - \Delta t)$. This we can solve with a forward and backward substitution. For details see project 1.

The last task is to implement the Crank Nichols scheme. We can here also start with the mathematical expression:

$$\frac{u(x_i, t_j + \Delta t) - u(x_i, t_j)}{\Delta t} = \frac{1}{2}\left(\frac{u(x_i + \Delta x, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j)}{\Delta x^2} + \frac{u(x_i + \Delta x, t_j + \Delta t) - 2u(x_i, t_j + \Delta t) + u(x_i - \Delta x, t_j + \Delta t)}{\Delta x^2}\right). \tag{11}$$

This expression can be a bit misleading. If we stare a bit at this equation we see that there are two parts that look the same as in the two schemes above. What isn't clear in eq. 10 is that we wish to solve with the explicit scheme first and the use tis solution in the implicit. It really is as easy as that but let me clarify a couple of details first. We rewrite the equation:

$$2(u(x_i, t_j + \Delta t) - u(x_i, t_j)) = \alpha\left((u(x_i + \Delta x, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j)) + (u(x_i + \Delta x, t_j + \Delta t) - 2u(x_i, t_j + \Delta t) + u(x_i - \Delta x, t_j + \Delta t))\right). \tag{12}$$

We see that we get a factor of two which changes our $\hat{B}$ matrix, so $\hat{B}' = (2\hat{I} - \alpha\hat{A})$

and the algoritm is now simply:

solve u(x,t) with explicit scheme.

put these u(x,t) values into the implicit sceme and solve using the $\hat{B}'$ matrix.

put u(0,t) $+=$ alpha

The last point is a cool trick to let us solve for u staigt away without changing to "simpler" boundary conditions. This comes from the first row of matrix $\hat{B}'$. In project 1 we had a matrix with two fictive points in the begining and end that where 0. So they gave us no problems. Now the points are $-\alpha$. if we write out the equation it gieldes, it goes as follows:

$$-\alpha u_0 + (2 + 2\alpha)u_1 - \alpha u_2 = u_1 \Rightarrow (2 + 2\alpha)u_1 - \alpha u_2 = u_1 + \alpha \tag{13}$$

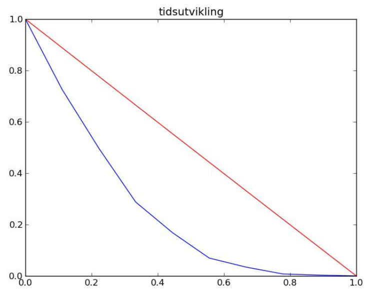
So we simply add α to the first entry.

Results

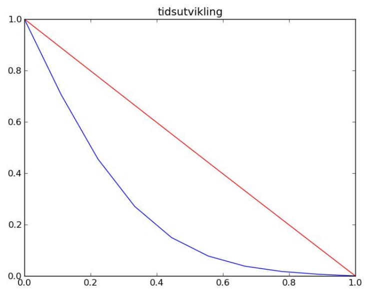
It is hard to get good results when i havent managed to find the closed form solution. My final expression tok form:

$$\sum_{n=1} \frac{(-1)^{n+1}}{n\pi} sin(\frac{\pi x}{L}) \sum_{k=0} \frac{(\frac{-n^2\pi^2 t}{L})^k}{k!} \tag{14}$$

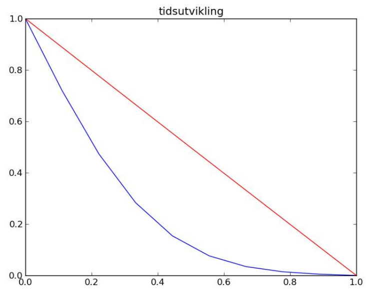
But i did not manage to get rid of the sumations. So instead of error analysis i wil mainly speak about stability.



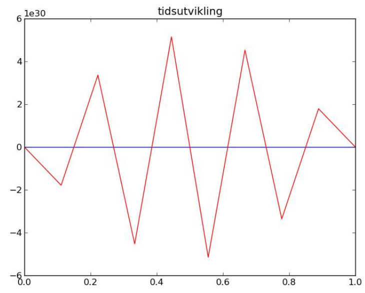
(a) explicit, dt = 0,006



(b) implicit, dt = 0,006



(c) crank nichols, dt = 0,006



(d) explicit, dt = 0,01

Figur 1: *Plot of solutions with the different schemes*

These plots are mostly interesting seen against the truncation errors, which i don't have. But We see that for the explicit scheme we lose stability at dt = 0,01, in general stability is lost if $\alpha < \frac{1}{2}$. In contrast to the implicit scheme that seemed to never loose stability, although the error will probably be way over axeptable values for large dt's.

The Crank Nichols Scheme is somewhere in between. I lost stability at dt = 0.04

Conclusion

It's hard to conclude with annything out from these humble results. I am guessing that the Crank Nichols scheme has the best truncation error, and will therefore be the algorithm of choise. It is important to note though that it is also the heaviest algorithm by far, scince we basically do both the other algorithms calculations in it. In the case where we need stability over error, the implicit scheme will be best. The implicit scheme seems to always give a reasonable solution. So if we only need to see "how" the system evolves this will always give us a satisfactory solution. The explicit scheme is the lightest, and therefore it might be the best choise if we are doing large computations.