Analysis of numerical algorithms for solving PDEs by solving the Diffusion equation

FYS3150 - Project 5

Authored by Philip Hoel, Theo G. Halvorsen
and Elakkyen Pathmanathan

December 18, 2020

UiO : Universitetet i Oslo

Contents

1 Abstract

This article looks at the diffusion equation. How it relates to other partial differential equation. It looks at how we can solve the diffusion equation analytically and numerically. The numerical analysis is about three different methods, the Backward Euler, Forward Euler, and Cranck-Nicolson method, perform solving a partial differential equation. We find that all methods work to some degree, but that the instability and inaccuracy of the Forward Euler method make it less in favor.

2 Introduction

Differential equations have a wide range of applications for different fields but are often used to describe and model dynamic systems. Methods for solving differential equations analytically can be both time consuming and difficult. Moreover, many of them can not be solved analytically. Therefore, being able to solve it numerically can be a powerful tool. It can often solve the equation faster and be based on more intuitive methods. Nevertheless, there is a catch to doing calculations numerically. Computers have limitations such as storage, computing power, and representation of numbers, which give rise to round-of-errors, leading to some loss of accuracy or, in the worst-case display, completely wrong digits. Because of this, it is vital to learn how to optimize code and find useful algorithms. In this project, we will study three different methods for solving the well-known diffusion equation, the implicit forward Euler, the explicit backward Euler, and the Crank-Nicholson, to look at how the methods differ in precision and efficiency. We will see how we can use diffusion equation, or heat equation, to simulate how the material's heat is distributed.
In this article, we will first introduce the diffusion equation and solve it analytically. Then we derive the methods and take a closer look at error and stability, followed by results and discussion.

3 Presenting the problem

As stated in the introduction, the equation we want to solve is the diffusion equation. The diffusion eqution is a special case of the Convetion-Diffusion equation

$$\frac{\partial c}{\partial t} = \nabla \cdot (D \nabla c) - \nabla (\mathbf{v} c) + R \tag{1}$$

When the bulk velocity becomes zero, this turns into the diffusion equation.

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

The diffusion equation describes the diffusion of material (particles) or energy. If we look at heat transfer in a material, the diffusion equation becomes a special case yet again and we call it the heat equation. For our project, we will at first ignore the physical interpretation of the equation and only look at the numerical aspect and then we will make an example plot as if it were modelling a rod and a plate in 2D. We have already seen the diffusion equation, but we present it yet again, now with its initial and boundary conditions.

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t}, t > 0, x \in [0, L] \tag{3}$$

with initial conditions

$$u(x, 0) = 0, \qquad 0 < x < L \tag{4}$$

for $L = 1$, and our boundary conditions will be

$$\begin{aligned} u(0,t) &= 0, \qquad t \geq 0 \\ u(L,t) &= 1, \qquad t \geq 0 \end{aligned} \tag{5}$$

3.1 Analytic solution

To be sure that the method we have implemented gives the correct answer, and to measure the method's accuracy further, we must solve for the analytical solution. We have our equation

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t} \tag{6}$$

and make a guess on a equation $F(x)G(t)$, and divide on both sides

$$\frac{\ddot{F}(x)}{F(x)} = \frac{\dot{G}(t)}{G(t)} = -\lambda^2 \tag{7}$$

since both of the equations are equal to each other and and they are derivatives of different variables, they must be equal to a constant [1]. We called this constant $-\lambda^2$. We now have two different ODEs

$$\dot{G}(t) = -\lambda^2 G(t) \tag{8}$$

and

$$\ddot{F}(x) + \lambda^2 F(x) = 0 \tag{9}$$

These are first and second order linear homogeneous differential equations, respectively and the general solutions are

$$G(t) = Ce^{-\lambda^2 t} \tag{10}$$

and

$$F(x) = Asin(\lambda x) + Bcos(\lambda x) \tag{11}$$

Since $u(0,t)=0$, we know that $Be^{-\lambda^2 t}=0$ and that $e^{-\lambda^2 t}$ (or anything else to the power of e, for that matter) can't be equal to zero. Thus $B=0$. Now our equation becomes

$$u(x,t)=Asin(\lambda x)e^{-\lambda^2 t} \tag{12}$$

If we let the time go to infinity, eventually the function becomes time independent. With other words, we will get a function $u(x,t)\to f(x)$ as $t\to\infty$. This will be the equilibrium temperature. This still needs to satisfy the boundary conditions of $u(x,t)$ and thus

$$\frac{d^2 f}{dx^2}=0, \qquad f(0)=0, \qquad f(L)=1 \tag{13}$$

We integrate twice and get

$$f(x)=\frac{x}{L} \tag{14}$$

Now we can define a new function

$$v(x,t)=u(x,t)-f(x) \tag{15}$$

$$u(x,t)=v(x,t)+f(x) \tag{16}$$

If we now differentiate twice on both sides

$$\begin{aligned}\frac{\partial u}{\partial t}&=\frac{\partial v}{\partial t}+0\\ \frac{\partial^2 u}{\partial x^2}&=\frac{\partial^2 v}{\partial x^2}+0\end{aligned} \tag{17}$$

and we can see that $u(x,t)$ and $v(x,t)$ must both satisfy the same PDE. $v(x,t)$ is now a PDE with homogeneous boundary conditions. The general equation for $v(x,t)$ with homogeneous boundary conditions is

$$v(x,t)=\sum_{n=1}^{\infty}A_n sin(\frac{n\pi x}{L})e^{-(\frac{n\pi}{L})^2 t} \tag{18}$$

The A_n sum is what we call a fourier series. To find the Fourier coefficient, we use the equation [1]

$$A_n=\frac{2}{L}\int_0^L f(x)sin(\frac{n\pi x}{L})dx \tag{19}$$

We solve this integral

$$A_n=\frac{2(\pi n cos(\pi n)-sin(\pi n))}{\pi^2 n^2} \tag{20}$$

since $u(x,t)=f(x)+v(x,t)$, we get the final general equation for our PDE fit for our boundary conditions and initial conditions.

$$u(x,t)=\frac{x}{L}+\sum_{n=1}^{\infty}A_n e^{-(\frac{n\pi}{L})^2 t}sin(\frac{n\pi}{L}x) \tag{21}$$

4 Method

To further solve our differential equation, we need to discretize it. The basic strategy for defining a differentiated method is to evaluate a function at a few points, find the polynomial that interpolates the function at these points, and use the derivative of this polynomial to approximate the derivative of the function. We are looking at three widely used methods, the Euler method both backward and forward, and the implicit Crank-Nicolson method. We start looking at the Euler Forward method [1].

4.1 Explicit Scheme (Forward Euler)

Assume that we have a basic differential equation

$$\dot{y} = f(t, y) \tag{22}$$

and an interval from $[a, b]$, that we discretize into $t_0, t_2, ..., t_n$ points. That gives us approximations $y_0(= c), y_1(t_1), ..., y_n(t_n)$, with step size $h = (t_n - t_0)/n$ and a intial condition c. To construct the numerical method, we use Taylor expansion

$$y(t + h) = y(t) + h\dot{y}(t) + \frac{h^2}{2}\ddot{y}(t) + O(h^3) \tag{23}$$

We can get the forward Euler method by rewriting the Taylor expansion, truncating it to the first derivative, and write it in an iterative notation, namely

$$y(t_{i+1}) = y(t_i) + h\dot{y}(t_i) \tag{24}$$

then replace $\dot{y}$ with f

$$y(t_{i+1}) = y(t_i) + hf(t_i, y_i) \tag{25}$$

The same yields for our case with the partial differential equation, were we can consider our left-hand side $\frac{\partial^2 u}{\partial x^2}$ as the function f, and right-hand side $\frac{\partial u}{\partial t}$ as $\dot{y}$. Before we plug into our method, we need to define the expressions with Taylor expansion. However, this time we are using central differencing for the space derivative.
Assume that we have an interval $[0, L]x[0, T]$, that we discretize into $x_0, x_2, ..., x_n$ and $t_0, t_2, ..., t_n$ points. That gives us approximations
$u(x_0, t_0), u(x_1, x_{t0}), ..., u(x_n, t_0), u(x_0, t_1), ..., u(x_n, t_n)$ with step size Δt and Δx.

Right side

$$\begin{aligned} u(x, t + \Delta t) &= u(x, t) + \Delta t\frac{\partial u}{\partial t}(x, t) + \frac{\Delta t^2}{2}\frac{\partial^2 u}{\partial t}(x, t) + ... \\ u(x, t + \Delta t) &= u(x, t) + \Delta t\frac{\partial u}{\partial t}u(x, t) \end{aligned} \tag{26}$$

Left side

$$\begin{aligned}
&u(x+\Delta x,t) = u(x,t) + \Delta x\frac{\partial u}{\partial x}(x,t) + \frac{\Delta x^2}{2}\frac{\partial^2 u}{\partial x}(x,t)... \\
&u(x-\Delta x,t) = u(x,t) - \Delta x\frac{\partial u}{\partial x}(x,t) + \frac{\Delta x^2}{2}\frac{\partial^2 u}{\partial x}(x,t)... \\
&u(x+\Delta x,t) + u(x-\Delta x,t) = 2u(x,t) + \Delta x^2\frac{\partial^2 u}{\partial x}(x,t) \\
&\frac{\partial^2 u}{\partial x}(x,t) = \frac{u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)}{\Delta x^2}
\end{aligned} \tag{27}$$

We plug the equations in and write iteratively, namely

$$\begin{aligned}
&u(x,t+\Delta t) = u(x,t) + \frac{\Delta t}{\Delta x^2}(u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)) \\
&u_{i,j+1} = u_{i,j} + \frac{\Delta t}{\Delta x^2}(u_{i+1,j} - 2u_{i,j} + u_{i-1,j})
\end{aligned} \tag{28}$$

We have now derived the forward Euler method. Since we evaluate the diffusion equation, we introduce an associated constant α defined as $\alpha = \frac{\Delta t}{\Delta x^2}$. By introducing the new constant to the equation, we get

$$u_{i,j+1} = \alpha u_{i-1,j} + (1-2\alpha)u_{i,j} + \alpha u_{i+1,j} \tag{29}$$

4.2 Implicit Scheme (Backward Euler)

Backward Euler method works almost the same way as Forward Euler method, only that we evaluate at the current and past step. For us, that will be the current and previous step of the time derivation, and with the same central difference of the space derivation. We derive the method with Taylor expansion of u(x, t+1) and reuse of expression delta(x).

$$\begin{aligned}
&u(x,t-\Delta t) = u(x,t) - \Delta t\frac{\partial u}{\partial t}(x,t) + \frac{\Delta t^2}{2}\frac{\partial^2 u}{\partial t}(x,t) + ... \\
&u(x,t-\Delta t) = u(x,t) - \Delta t\frac{\partial u}{\partial t}(x,t)
\end{aligned} \tag{30}$$

With the derivation from equation (16) we have

$$\begin{aligned}
&u(x,t-\Delta t) = u(x,t) \\
&-\frac{\Delta t}{\Delta x^2}(u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)) \\
&u_{i,j-1} = u_{i,j} - \frac{\Delta t}{\Delta x^2}(u_{i+1,j} - 2u_{i,j} + u_{x-1,t}) \\
&u_{i,j-1} = (1+2\alpha)u_{i,j} - \alpha u_{i+1,j} - \alpha u_{i-1,j}
\end{aligned} \tag{31}$$

We have now derived Backward Euler and come to the last method, the Crank – Nicolson method.

4.3 Crank-Nicholson

To derive the Crank-Nicolson method, we need the Taylor expansion of $u(x+\Delta x,t)$, $u(x-\Delta x,t)$, $u(x,t+\Delta t)$, $u(x+\Delta x,t+\Delta t)$, and $u(x-\Delta x,t+\Delta t)$ around $t'=t+\frac{\Delta t}{2}$. We combine the aforementioned Taylor expansions, namely

$$\begin{aligned} u(x+\Delta x,t+\Delta t) = u(x,t') + \frac{\partial u(x,'t)}{\partial x}\Delta x + \frac{\partial u(x,t')}{\partial t}\frac{\Delta t}{2} \\ + \frac{\partial^2 u(x,t')}{2\partial x^2}\Delta x^2 + \frac{\partial^2 u(x,t')}{2\partial t^2}\frac{\Delta t^2}{4} + \frac{\partial^2 u(x,t')}{\partial x \partial t}\frac{\Delta t}{2}\Delta x \end{aligned} \tag{32}$$

$$\begin{aligned} u(x-\Delta x,t+\Delta t) = u(x,t') - \frac{\partial u(x,'t)}{\partial x}\Delta x + \frac{\partial u(x,t')}{\partial t}\frac{\Delta t}{2} \\ + \frac{\partial^2 u(x,t')}{2\partial x^2}\Delta x^2 + \frac{\partial^2 u(x,t')}{2\partial t^2}\frac{\Delta t^2}{4} - \frac{\partial^2 u(x,t')}{\partial x \partial t}\frac{\Delta t}{2}\Delta x \end{aligned} \tag{33}$$

$$\begin{aligned} u(x+\Delta x,t) = u(x,t') + \frac{\partial u(x,'t)}{\partial x}\Delta x + \frac{\partial u(x,t')}{\partial t}\frac{\Delta t}{2} \\ - \frac{\partial^2 u(x,t')}{2\partial x^2}\Delta x^2 + \frac{\partial^2 u(x,t')}{2\partial t^2}\frac{\Delta t^2}{4} - \frac{\partial^2 u(x,t')}{\partial x \partial t}\frac{\Delta t}{2}\Delta x \end{aligned} \tag{34}$$

$$\begin{aligned} u(x-\Delta x,t) = u(x,t') - \frac{\partial u(x,'t)}{\partial x}\Delta x - \frac{\partial u(x,t')}{\partial t}\frac{\Delta t}{2} \\ + \frac{\partial^2 u(x,t')}{2\partial x^2}\Delta x^2 + \frac{\partial^2 u(x,t')}{2\partial t^2}\frac{\Delta t^2}{4} + \frac{\partial^2 u(x,t')}{\partial x \partial t}\frac{\Delta t}{2}\Delta x \end{aligned} \tag{35}$$

$$u(x,t\Delta t) = u(x,t') + \frac{\partial u(x,t')}{\partial t}\frac{\Delta t}{2} + \frac{\partial^2 u(x,t')}{2\partial t^2}\Delta t^2 \tag{36}$$

$$u(x,t) = u(x,t') - \frac{\partial u(x,t')}{\partial t}\frac{\Delta t}{2} + \frac{\partial^2 u(x,t')}{2\partial t^2}\Delta t^2 \tag{37}$$

by introducing alpha and iterative notation, we have our method

$$-\alpha u_{i-1,j} + (2+2\alpha)u_{i,j} - \alpha u_{i+1,j} = \alpha u_{i-1,j-1} + (2-2\alpha)u_{i,j-1} + \alpha u_{i+1,j-1} \tag{38}$$

4.4 Tridiagonal method

Our equation

$$u_{i,j-1} = (1+2\alpha)u_{i,j} - \alpha u_{i+1,j} - \alpha u_{i-1,j} \tag{39}$$

is a essentially a system of linear equations, since we will do the same equation for all i and j. To solve systems of equations, the most effective way will be to use a matrix. We rewrite our equation as a matrix

$$\begin{bmatrix} 1+2\alpha & -\alpha & 0 & 0 & 0 & ... & ... \\ -\alpha & 1+2\alpha & -\alpha & 0 & 0 & ... & ... \\ 0 & -\alpha & 1+2\alpha & -\alpha & 0 & ... & ... \\ ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & 0 & -\alpha & 1+2\alpha & -\alpha \\ ... & ... & ... & 0 & 0 & -\alpha & 1+2\alpha \end{bmatrix} \begin{bmatrix} u_{1,j-1} \\ u_{2,j-1} \\ u_{3,j-1} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ u_{n,j-1} \end{bmatrix} = \begin{bmatrix} u_{1,j} \\ u_{2,j} \\ u_{3,j} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ u_{n,j} \end{bmatrix} \tag{40}$$

To solve a linear matrix equation, one usually uses the Gaussian Elimination or LU Decomposition. Our matrix is what is called a tridigonal matrix. A tridiagonal matrix is a special type of matrix, where all the entries are zeros except the diagonal and the immediate entries above and below the diagonol. This means that we can use the tridigonal method, which is a much more efficient method. The Gaussian elimination requires $\frac{2n^2}{3} + \mathcal{O}(n^2)$ floating-point operations, while the tridiagonal method uses $\mathcal{O}(n)$. To show the algorithm, we use a matrix with $n = 4$ and we set the diagonals to be b_i and the entries immediate above and below to be a_i and the rest is zeros. We divide the steps into forward substitution and backward substitution. We start by deriving the forward substitution

$$\begin{bmatrix} b_1 & a_1 & 0 & 0 \\ a_1 & b_2 & a_1 & 0 \\ 0 & a_2 & b_3 & a_3 \\ 0 & 0 & a_3 & b_4 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \tag{41}$$

$$\begin{bmatrix} b_1 & a_1 & 0 & 0 & q_1 \\ a_1 & b_2 & a_1 & 0 & q_2 \\ 0 & a_2 & b_3 & a_3 & q_3 \\ 0 & 0 & a_3 & b_4 & q_4 \end{bmatrix} \tag{42}$$

$$\begin{bmatrix} b_1 & a_1 & 0 & 0 & q_1 \\ 0 & b_2 - a_2 a_1/b_1 & a_1 & 0 & q_2 - q_1 a_1/b_1 \\ 0 & a_2 & b_3 & a_3 & q_3 \\ 0 & 0 & a_3 & b_4 & q_4 \end{bmatrix} \tag{43}$$

$$\begin{bmatrix} b_1 & a_1 & 0 & 0 & q_1 \\ 0 & \widetilde{b}_2 & a_1 & 0 & \widetilde{q}_2 \\ 0 & a_2 & b_3 & a_3 & q_3 \\ 0 & 0 & a_3 & b_4 & q_4 \end{bmatrix} \tag{44}$$

$$\begin{bmatrix} b_1 & a_1 & 0 & 0 & q_1 \\ 0 & \widetilde{b}_2 & a_1 & 0 & \widetilde{q}_2 \\ 0 & 0 & b_3 - a_2 a_1/\widetilde{b}_2 & a_3 & q_3 - \widetilde{q}_2 a_1/\widetilde{b}_2 \\ 0 & 0 & a_3 & b_4 & q_4 \end{bmatrix} \tag{45}$$

$$\begin{bmatrix} b_1 & a_1 & 0 & 0 & q_1 \\ 0 & \widetilde{b}_2 & a_1 & 0 & \widetilde{q}_2 \\ 0 & 0 & \widetilde{b}_3 & a_3 & \widetilde{q}_3 \\ 0 & 0 & a_3 & b_4 & q_4 \end{bmatrix} \tag{46}$$

$$\begin{bmatrix} b_1 & a_1 & 0 & 0 & q_1 \\ 0 & \widetilde{b}_2 & a_1 & 0 & \widetilde{q}_2 \\ 0 & 0 & \widetilde{b}_3 & a_3 & \widetilde{q}_3 \\ 0 & 0 & 0 & \widetilde{b}_4 & \widetilde{q}_4 \end{bmatrix} \tag{47}$$

We can eventually see a pattern as we continue and we are left with the equations

$$\begin{aligned} \widetilde{b}_i &= b_i - \frac{a_i a_{i-1}}{\widetilde{b}_{i-1}} \\ \widetilde{q}_i &= q_i - \frac{q_{i-1} a_{i-1}}{\widetilde{b}_{i-1}} \end{aligned} \tag{48}$$

We now show the backward substitution. We start dividing by $\widetilde{b}_i$

$$\begin{bmatrix} b_1 & a_1 & 0 & 0 & q_1 \\ 0 & \widetilde{b}_2 & a_1 & 0 & \widetilde{q}_2 \\ 0 & 0 & \widetilde{b}_3 & a_3 & \widetilde{q}_3 \\ 0 & 0 & 0 & 1 & z_4 \end{bmatrix} \tag{49}$$

$$\begin{bmatrix} b_1 & a_1 & 0 & 0 & q_1 \\ 0 & \widetilde{b}_2 & a_1 & 0 & \widetilde{q}_2 \\ 0 & 0 & 1 & 0 & z_3 \\ 0 & 0 & 0 & 1 & z_4 \end{bmatrix} \tag{50}$$

$$\begin{bmatrix} b_1 & a_1 & 0 & 0 & q_1 \\ 0 & 1 & 0 & 0 & z_1 \\ 0 & 0 & 1 & 0 & z_3 \\ 0 & 0 & 0 & 1 & z_4 \end{bmatrix} \tag{51}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & z_1 \\ 0 & 1 & 0 & 0 & z_2 \\ 0 & 0 & 1 & 0 & z_3 \\ 0 & 0 & 0 & 1 & z_4 \end{bmatrix} \tag{52}$$

We again see a pattern emerge and we are left with the equation

$$z_i = \frac{\widetilde{q}_i - z_{i+1} a_{i-1}}{\widetilde{b}_i} \tag{53}$$

By splitting up and rewriting the equation, we get our algorithm

Algorithm 1: Tridiagonal algorithm

```
for $i = 0$ to $n$ do
    a[i-1] /= b[i-1]
    u[i,j] /= b[i-1]
    b[i-1] = 1.

    u[i+1,j] += u[i,j]*alpha
    b[i] += a[i-1]*alpha
end

u[n,j] /= b[n-1]
b[n-1] = 1.

for $i = n$ to 0 do
    u[i-1,j] -= u[i,j]*a[i-2]
end
```

4.5 Errors and stability

We have now derived the methods, and we can already begin to look at what we can expect from the methods by looking back at the derivations. We already know that when we approximate a function, it will never be wholly exact, and therefore, there will be some errors in the solutions. When we use Taylor expansion, there is an error regarding the degree of the polynomial. That means the degree of derivative we want to truncate to will give some remainder, determining the degree of error. The remainder is defined as

$$R_n(f,a)(x) = \frac{(x-a)^{(n+1)}}{(n+1)!} f^{(n+1)}(\xi) \tag{54}$$

where ξ is a point between a and x, if $x < a$ [2]. If we now consider the Taylor expansions for $\frac{\partial u}{\partial t}$ and $\frac{\partial^2 u}{\partial x^2}$, we can define an error, namely

$$\Big|\frac{\partial u}{\partial t} - \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t}\Big| = \frac{\Delta t^2}{2!}\frac{\partial^2 u}{\partial t}(\xi_t) \tag{55}$$

$$\Big|\frac{\partial^2 u}{\partial x} - \frac{u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)}{\Delta x^2}\Big| = \frac{\Delta t^3}{3!}\frac{\partial^3 u}{\partial x}(\xi_x) \tag{56}$$

We see the factor that makes the error smaller from the expression is the stride length, Δx, and Δt . That means, if we reduce the stride length, it will correspond to a smaller truncation deviation. However, that does not necessarily mean that the total error will shrink if we shrink the stride length infinitely. We also need to be aware of the round of error, which adds up when the stride length tends to zero.

What is also a vital remark is the method's stability, the certainty that the method converges towards a solution. In order to take a closer look at this, we must use studies from iterative schemes. We have something called the spectral radius, which is determined by the largest eigenvalue of the matrix. The spectral radius defines a circle with center at zero in the complex plane which contains all of the eigenvalues of the matrix. The spectral radius has to be smaller than one so that the solution converges after some amount of time steps.This requirements is met if the matrix is positive definite. The implicit scheme is always stable for all δt and δx since all eigenvalues will always be positive, on the other hand explicit scheme has possibilities of negative eigenvalues therefore must be selective of δt and δx to achieve correct solution. The same reasoning for implicit scheme's stability can also be applied to Crank-Nicolson scheme's stability too.

Scheme	Error	Stability Req.
Crank-Nicolson	$\mathcal{O}(\Delta x^2)$ and $\mathcal{O}(\Delta t^2)$	Stable for all Δx and Δt
Backward Euler	$\mathcal{O}(\Delta x^2)$ and $\mathcal{O}(\Delta t)$	Stable for all Δx and Δt
Forward Euler	$\mathcal{O}(\Delta x^2)$ and $\mathcal{O}(\Delta t)$	$\Delta t < \frac{1}{2}\Delta x^2$

Table 1: A table showing all the different schemes, with their error and stability requirements

5 Results

5.1 1D Equation

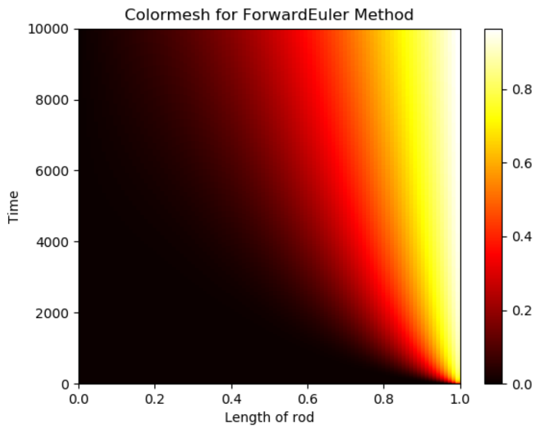


Figure 1: Colormesh for the forward Euler scheme. Shows heat transfer over time. $n = 100$, $t = 10000$, $dt = 0.0001$

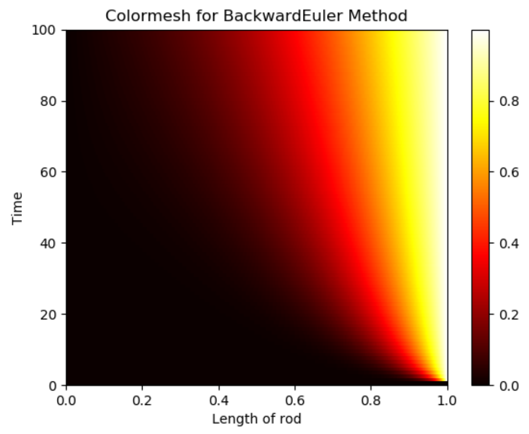


Figure 2: Colormesh for backward Euler scheme. $n = 1e^5$, $t = 100$, $dt = 0.01$

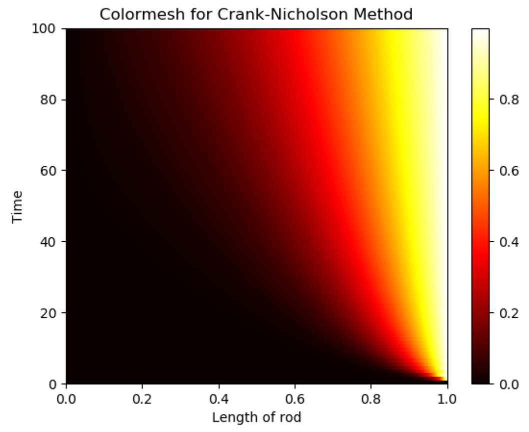


Figure 3: Show the development of the heat transfer in a rod of length one over time. The x-axis is the length of the rod and the y-axis is the time steps. This plot is generated with the Crank-Nicolson method at $n = 10^4$, $t = 100$ and $dt = 0.001$

5.2 2D Equation

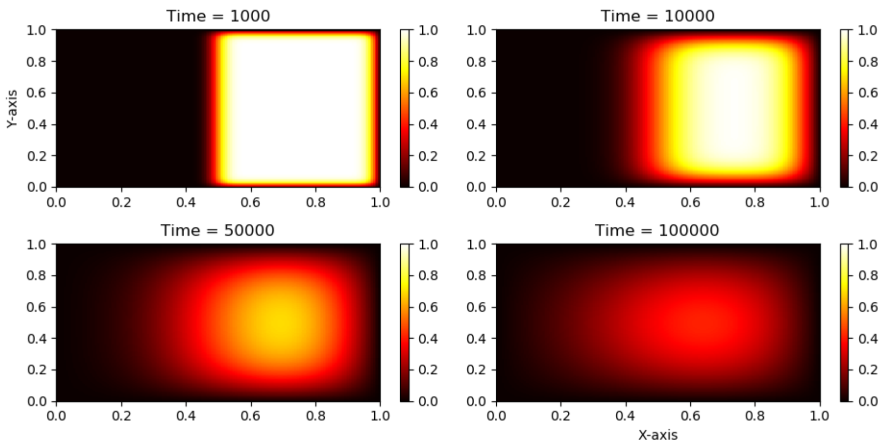
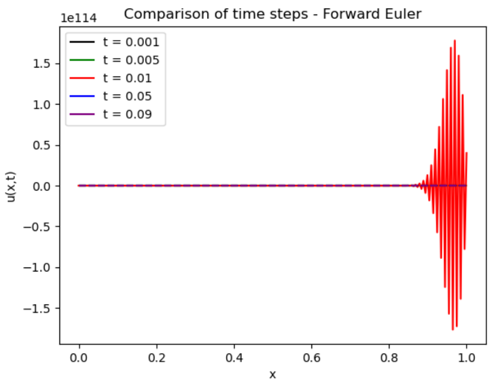
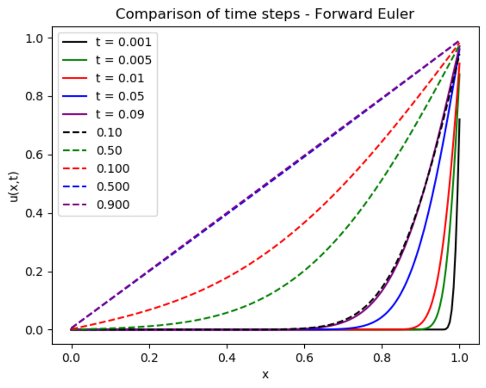


Figure 4: Shows the development of the 2D heat equation at different moments in time. The colorbar shows the output of the equation for the interval $[0, 1]$. The boundary conditions are 0 around the plate and the initial conditions are 1 for $x > 0.5$. The time steps were set to $t = 10^6$, $n = 200$ and $dt = 10^{-11}$

As we can see from the plots, the heat distribution becomes more and more linear with time. The next plots are to see the deviation for the different methods.

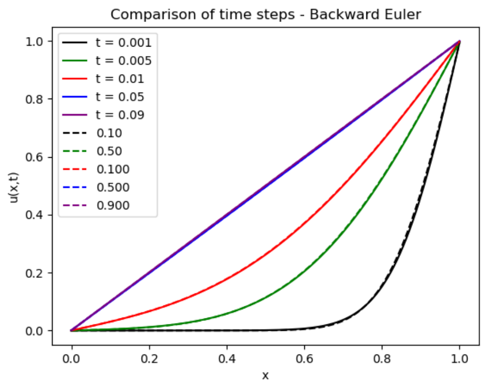


(a) Unstable plot

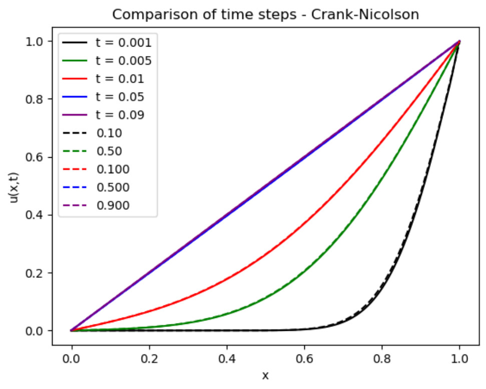


(b) Stable plot

Figure 5: Shows the plot of forward Euler against the analytic solution. The analytic is the dashed line and the whole line is the numerical. In the first plot, $dt = 0.0001$ and in the second plot $dt = 0.0001$.



(a) Backward Euler



(b) Crank-Nicolson

Figure 6: Shows the plot of backward euler and crank-nicolson against the analytical solution. $n = 2000$, $t = 1000$ and $dt = 0.01$

5.2.1 Runtime

Scheme	Runtime [s]
Forward Euler	0.0263584
Backward Euler	0.210564
Crank-Nicolson	0.225547

Table 2: Table showing runtime for the different schemes. Trial runs where run with $n = 10000$, $t = 100$ and $dt = 0.001$. The runtimes are an average of 100 runs

6 Discussion

There is not much to be gained from the first plots of the heat distribution, but it indicates whether the methods are correct intuitively. On the other hand, for the next plots where we compare the analytical solution and the numerical solution, we can see how the methods excel in error. Where the implicit methods have a better approximation, laying closer to the analytical line relative to the explicit.
One possible explanation for why the implicit methods work better than the explicit ones is how the method naturally accumulates over previous approximations to find the next approximations. In the explicit method, we need three previous points to find the next point, so we know from the previous analysis of error that each approximation has a deviation relative to the Taylor polynomial. That means that the more points we consider in the calculation for the next point, the greater the chance we accumulate over deviations, which becomes

larger for each step. In the implicit methods, we need, unlike the explicit, only one previous point to decide the next. That ensures that the chance of large deviations remains lower than the deviation of the explicit method.

To justify the answers we have received, we take a closer look at the method and implementation. We count the number of flops in the implementation where the explicit method gets $n * t * 7$ flops, and the implicit have $t * n * 10$ and Crank-Nicolson have $t * n * 17$ flops. There is not much difference between them, but it makes up a little. However, there are other factors to be considered in runtime, whether an algorithm is in-place or not. That means whether we use more intermediate storage to arrive at the answer than one. We use multiple intermediate storages in our implicit methods, which provide more load and stores to the program, leading to a greater cache-miss chance, which brings a longer time.

7 Conclusion

Partial differential equations are a wast field inside mathematics, and the same is the field of numerical analysis. However, they go hand in hand. The diffusion equation especially, which is used in many other sciences than just mathematics and physics. In our project, we have seen how the diffusion equation relates to other known PDEs, and we have seen how it can be solved analytically. Since it can be solved analytically, it is perfect to further the numerical analysis by solving it and testing it to our numerical method. That is crucial to learn more about how we can solve unsolvable differential equations. We have seen three different numerical methods and how they differ and how they are similar. The forward Euler is a faster (although not much) but less stable algorithm. The implicit scheme and Crank-Nicolson are relatively similar and highly stable. Nevertheless, we were able to use the explicit scheme to solve the two-dimensional diffusion equation. If we had more time in this project, we could have seen how the two-dimensional diffusion equation does in an implicit algorithm.

References

[1] Morten Hjort-Jensen. “Computational Physics Lectures: Partial differential equations”. In: (August 2015), pp. 1–20. URL: http://compphysics.github.io/ComputationalPhysics/doc/pub/pde/pdf/pde-print.pdf.

[2] Knut Mørken. “Numerical Algorithms and Digital Representation”. In: (September 2017), pp. 217–354. URL: https://www.uio.no/studier/emner/matnat/math/MAT-INF1100/h17/kompendiet/matinf1100.pdf.

8 Github link

Github