Diffusion equation: Analysis of numerical schemes

Tor-Andreas Bjone, Håvar Rinde Lund Jacobsen, Håkon Rinde Lund Jacobsen
(Dated: December 18, 2020)

In this report we have studied the stability and accuracy of three numerical schemes for partial differential equations. These schemes are the forward Euler (explicit), backward Euler (implicit) and Crank-Nicolson (implicit). The equation we have solved is the diffusion equation in one and two spacial dimensions, which has closed-form solutions in both cases. By comparing our numerical results with the analytical solutions, we have found that all three schemes provide accurate and stable solutions as long as the stability criteria for explicit schemes is fulfilled. When the stability criteria is slightly violated, the explicit scheme quickly diverges from the analytical solution, while the implicit methods show no sign on instability. All of our material involving numerical implementations are available on our Github (Bjone & Jacobsen, 2020)[6].

I. INTRODUCTION

In this project we have studied the numerical stability and accuracy of three methods (schemes) for solving partial differential equations (PDE's). The methods which we have looked at are:

- The Forward Euler scheme (explicit algorithm).
- The Backward Euler scheme (implicit algorithm).
- The Crank-Nicolson scheme (implicit algorithm).

To test these methods, we have used a classical example: The diffusion equation. On general form, it reads

$$\nabla^2 u = A\frac{\partial u}{\partial t}.$$

The diffusion equation is a partial differential equation in space and time. It has a variety of applications and is widely used in fields like natural science. The coefficient A depends on the physical system at hand. Examples of use could be to describe the diffusion of particles and heat in materials, or the prorogation of electrical signals in nerve cells. To solve these types of problems we need stable and accurate numerical schemes for partial differential equations.

In our studies, we have looked at a scaled and dimensionless version of the diffusion equation, with applications to any field of interest. First, we limit ourselves to one spacial dimension. Since we also have to include time, the diffusion equation has dimensionality $1 + 1$, which we will refer to as the one dimensional diffusion equation. In the one dimensional case, we shall test all of the schemes mentioned above, and compare them in terms of accuracy and stability. Moving on to the two dimensional diffusion equation, which has dimensionality $2 + 1$, we shall limit ourselves to the explicit scheme. The diffusion equation has closed-form analytical solutions in one and two dimensions which we can use to compare with our numerical results. This is important for validating our results.

The explicit and implicit schemes have both certain strengths and weaknesses. Explicit schemes are easier to implement and faster to solve, but they have an additional stability requirement which links step sizes in position and time. If the stability criterion is violated, the numerical solution quickly becomes unstable, as we will see later on in this report. Implicit schemes allows us to choose step sizes more freely, as they are stable for all combinations of steps in position and time. On the other hand, solving them often involve more complicated and slow algorithms. As long as the stability criterion is fulfilled, the methods mentioned above are roughly equal in terms of accuracy and stability within reasonable time intervals.

The report is structured into sections of theory, methods, results, discussion and conclusion. We have included some appendixes to further explain derivations of numerical algorithms and closed-form solutions. In theory, the diffusion equation is first introduced. Then we apply the numerical schemes on the one dimensional diffusion equation, deriving the algorithms belonging to each method. The explicit method is then applied to the two dimensional diffusion equation. Truncation errors, stability and closed-form solutions are discussed at the end of the theory. In methods, we explain how the numerical algorithms are implemented. Then, results are presented and described before moving on to discussing important aspects of the project, leading up to the conclusion.

II. THEORY

A. The Diffusion equation

The diffusion equation is a partial differential equation that can be used to model many different diffusion processes, for example heat diffusion in a material. It can be

used to describe physical processes such as heat distribution in a region using the temperature gradient, or density fluctuations in materials. It describes how the density u of quantities like particle density, energy density, temperature gradient or chemical concentration evolve in time (Hjorth-Jensen, 2015, pp. 301-325)[2]. Assuming that the flux density ρ obeys Gauss-Green theorem (Divergence theorem, Stokes's theorem), which is

$$\int_V \nabla \cdot \vec{\rho}\, dV = \oint_{\partial V} \vec{\rho} \cdot \vec{n}\, dS,$$

where V is a smooth region in space enclosed within the surface ∂V. The vector $\vec{n}$ is the normal (perpendicular) surface vector. We are interested in a solution in the region V. The Gauss-Green theorem leads to

$$\nabla \cdot \vec{\rho} = 0,$$

on differential form. This means that the flux density ρ is divergence free (it does not expand or contract).

The gradient ∇u points from regions of low concentration to regions of high concentration. In the case of heat diffusion, ∇u is the temperature gradient which points from regions of low temperature to regions of high temperature. We know that heat flows in the opposite direction of the temperature gradient (from high temperature to low temperature). The general assumption is therefore that $\vec{\rho}$ is proportional to ∇u, such that $\vec{\rho} = -D\nabla u$. Flux density points in the opposite direction of the gradient. This leads us to

$$\nabla \cdot \vec{\rho} = \nabla \cdot (-D\nabla u) = -D(\nabla \cdot \nabla u) = 0,$$

or

$$D\nabla^2 u = 0,$$

which is Laplace's equation (Hjorth-Jensen, 2015, pp. 301-325)[2]. The constant D can be coupled with various physical constants (for example the diffusion constant, specific heat capacity or conductivity). Coupling the rate of change of u with the flux density ρ we get

$$\frac{\partial u}{\partial t} = -\nabla \cdot \vec{\rho} = D\nabla^2 u,$$

this is the diffusion equation. In three-dimensional space, we have that

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2},$$

using Cartesian coordinates. The one-dimensional case is shown in equation (2), which is on dimensionless form. The multidimensional diffusion equation on dimensionless form is just

$$\nabla^2 u = \frac{\partial u}{\partial t}. \tag{1}$$

In the one-dimensional case, we limit x such that $x \in [0, L]$, where L is some length. We use initial conditions at $t = 0$,

$$u(x, 0) = g(x),$$

for $0 < x < L$, where $g(x)$ is a known function, only dependent on position x. The boundary conditions for $t \geq 0$ are

$$u(0, t) = a(t),$$

and

$$u(L, t) = b(t),$$

where $a(t)$ and $b(t)$ are functions only dependent on time t.

The two dimensional case is discussed in subsection (II D).

B. Diffusion equation in one dimension

The diffusion equation in one spacial dimension (Hjorth-Jensen, 2020)[1], (Hjorth-Jensen, 2015, pp. 301-325)[2] is just a simplified version of the general case (1):

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

here on dimensionless form. It's a linear partial differential equation in space x and time t. We see that $u(x, t)$ can be any diffusion function in one spacial dimension. We define $t \geq 0$ and $x \in [0, L]$, with $L = 1$.

An example of what boundary and initial conditions could be:

Boundary conditions: $u(0, t) = 0$ and $u(L, t) = 1$ for $t \geq 0$.

Initial conditions: $u(x,0) = 0$ for $0 < x < L$.

If we use $n+1$ integration points for $x \in [0,1]$, our step size is

$$\Delta x = \frac{1}{n+1}.$$

To check numerical stability, we have used three methods for partial differential equations (PDE-solvers). The first is the **explicit forward Euler algorithm** (Hjorth-Jensen, 2015, pp. 304-308)[2] with discretized versions of time $t_j = t_0 + j\Delta t$ and space $x_i = x_0 + i\Delta x$, where i and j are integers from zero to some finite number. The diffusion equation is approximated by a forward formula for time

$$\frac{\partial u(x,t)}{\partial t} \approx \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} = \frac{u(x_i,t_j+\Delta t) - u(x_i,t_j)}{\Delta t},$$

and a centered difference in space resulting in

$$\frac{\partial^2 u(x,t)}{\partial x^2} \approx \frac{u(x+\Delta x,t) + u(x-\Delta x,t) - 2u(x,t)}{\Delta x^2},$$

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

An **important criterion** for the **explicit methods** is to make sure that

$$\frac{\Delta t}{\Delta x^2} \leq \frac{1}{2}, \tag{3}$$

otherwise the numerical solution become unstable. This is further discussed in the subsection (II E).

The second method is the **implicit Backward Euler algorithm** (Hjorth-Jensen, 2015, pp. 308-310)[2] with

$$\frac{\partial u(x,t)}{\partial t} \approx \frac{u(x,t) - u(x,t-\Delta t)}{\Delta t},$$

$$\frac{\partial u(x,t)}{\partial t} \approx \frac{u(x_i,t_j) - u(x_i,t_j-\Delta t)}{\Delta t}.$$

And with spacial centered difference as

$$\frac{\partial^2 u(x,t)}{\partial x^2} \approx \frac{u(x+\Delta x,t) + u(x-\Delta x,t) - 2u(x,t)}{\Delta x^2},$$

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

Both of these, the explicit Forward Euler and Backward Euler, has an error in time that goes like $O(\Delta t)$ and an error in space that goes like $O(\Delta x^2)$.

The third method is the **implicit Crank-Nicolson scheme** (Hjorth-Jensen, 2015, pp. 308-314)[2], which is a **time centered scheme** with center $t + \Delta t/2$. This method is explained a bit further in appendix (C), by equation (C1). The time derivative is

$$\frac{\partial u(x,t)}{\partial t} \approx \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} = \frac{u(x_i,t_j+\Delta t) - u(x_i,t_j)}{\Delta t},$$

and spatial second-order derivative reads

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

Truncation errors corresponding to the three different PDE-solvers are presented in table (I).

To simplify notation, we let

$$u(x_i,t_j+\Delta t) \to u_{i,j+1},$$

and

$$u(x_i+\Delta x,t_j) \to u_{i+1,j}.$$

C. Tridiagonal matrix system

In this subsection we will show how we can rewrite the one dimensional schemes as linear tridiagonal Toeplitz matrix equations, which means that the matrix involved is tridiagonal with diagonal elements constant. These equations are then solved using methods presented in subsection (III A).

For the **Explicit Forward Euler** scheme (Hjorth-Jensen, 2015, pp. 304-308)[2], we may write the diffusion equation as

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = \frac{u_{i+1,j} + u_{i-1,j} - 2u_{i,j}}{\Delta x^2},$$

defining $\alpha = \Delta t/\Delta x^2$, the equation above can be solved for the next time step $u_{i,j+1}$:

$$u_{i,j+1} = \alpha u_{i-1,j} + (1-2\alpha)u_{i,j} + \alpha u_{i+1,j}. \tag{4}$$

The equation above can be written as a matrix equation, with a tridiagonal Toeplitz matrix A on the form

$$A = \begin{bmatrix} 1-2\alpha & \alpha & 0 & 0 & ... & 0 \\ \alpha & 1-2\alpha & \alpha & 0 & ... & 0 \\ 0 & \alpha & 1-2\alpha & \alpha & 0 & ... \\ ... & ... & ... & ... & ... & ... \\ 0 & ... & ... & 0 & \alpha & 1-2\alpha \end{bmatrix}.$$

The discretized initial and boundary conditions are known as explained earlier. The way the explicit algorithm works is illustrated by figure (1). The first values (initial) are given by

$$u(x_i, 0) = u_{i,0} = g(x_i),$$

and the boundary values are known because

$$u(0, t_j) = u_{0,j} = a(t_j),$$
$$u(x_{n+1}, t_j) = u_{n+1,j} = b(t_j).$$

Therefore we only need to solve the diffusion equation for $i = 1, ..., n$ at the various time steps. We define a column vector V_j for each time step j which contains all $u_{i,j}$ values:

$$V_j = \begin{bmatrix} u_{1,j} \\ u_{2,j} \\ u_{3,j} \\ ... \\ u_{n,j} \end{bmatrix}. \tag{5}$$

The explicit scheme calculates the next step V_{j+1} using the previous step V_j through a matrix-vector multiplication

$$V_{j+1} = AV_j = A^{j+1}V_0, \tag{6}$$

where V_0 is the initial condition vector at $t = 0$, defined by $g(x)$.

Now we look at the **Implicit Backward Euler** scheme (Hjorth-Jensen, 2015, pp. 308-310)[2], which expresses the diffusion equation as

$$\frac{u_{i,j} - u_{i,j-1}}{\Delta t} = \frac{u_{i+1,j} + u_{i-1,j} - 2u_{i,j}}{\Delta x^2}.$$

Using $\alpha = \Delta t/\Delta x^2$ as before. The only unknown quantity is $u_{i,j-1}$, which is

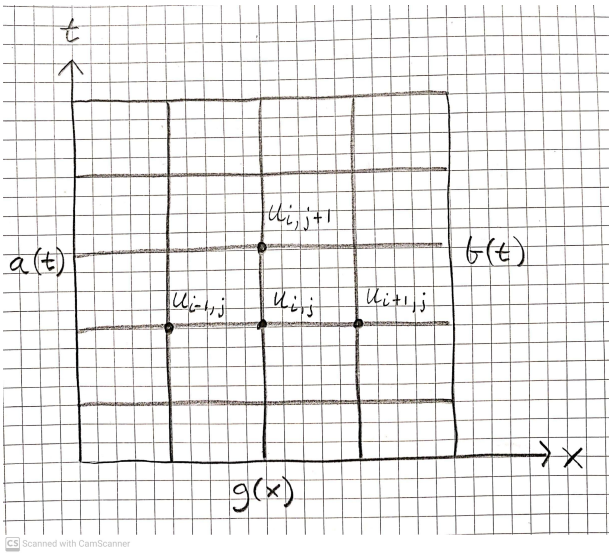

Figure 1. Discretization of integration area (x, t) of the one dimensional diffusion equation $u(x, t)$ (Hjorth-Jensen, 2015, pp. 304-308)[2], using the explicit scheme.

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

Defining a column vector V_j as before (5). The initial conditions and boundary conditions are the same as for the explicit scheme. For this scheme, the matrix involved becomes

$$A = \begin{bmatrix} 1+2\alpha & -\alpha & 0 & 0 & ... & 0 \\ -\alpha & 1+2\alpha & -\alpha & 0 & ... & 0 \\ 0 & -\alpha & 1+2\alpha & -\alpha & 0 & ... \\ ... & ... & ... & ... & ... & ... \\ 0 & ... & ... & 0 & -\alpha & 1+2\alpha \end{bmatrix}.$$

The implicit scheme calculates the previous vector V_{j-1} (backward in time) through

$$V_{j-1} = AV_j.$$

We exploit that α does not depend on time, meaning that we can rewrite the equation above as

$$A^{-1}V_{j-1} = A^{-1}AV_j = IV_j = V_j,$$

where $I = A^{-1}A$ is the identity matrix. We end up with the following algorithm:

$$V_j = A^{-1}V_{j-1} = A^{-j}V_0, \tag{7}$$

where V_0 is the initial condition vector at $t = 0$, defined by $g(x)$ as before. Since α does not depend on time, we only have to invert the matrix A once. The

way the implicit algorithm works is illustrated by figure (2).

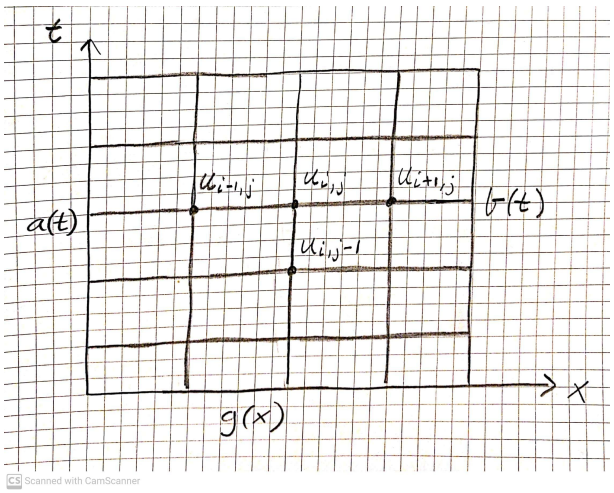

Figure 2. Discretization of integration area (x,t) of the one dimensional diffusion equation $u(x,t)$ (Hjorth-Jensen, 2015, pp. 308-310)[2], using the implicit scheme.

The **Implicit Crank-Nicolson scheme** (Hjorth-Jensen, 2015, pp. 310-313)[2] can be written as

$$\frac{u_{i,j+1}-u_{i,j}}{\Delta t} = \frac{u_{i+1,j}+u_{i-1,j}-2u_{i,j}}{2\Delta x^2} + \frac{u_{i+1,j+1}+u_{i-1,j+1}-2u_{i,j+1}}{2\Delta x^2}.$$

Multiplying both sides with Δt and using $\alpha = \Delta t/\Delta x^2$, we see that

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

Multiplying both sides by 2 and rearranging the $j+1$ terms to the left and j terms to the right, we get

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

which we can rewrite as

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

The last equation is a tridiagonal matrix equation, on the form

$$(2I+\alpha B)V_j = (2I-\alpha B)V_{j-1},$$

using column vectors V_j and V_{j-1} as before (5). Note that I is just the identity matrix, and B is

$$B = \begin{bmatrix} 2 & -1 & 0 & 0 & ... & 0 \\ -1 & 2 & -1 & 0 & ... & 0 \\ 0 & -1 & 2 & -1 & ... & 0 \\ ... & ... & ... & ... & ... & ... \\ 0 & ... & ... & 0 & -1 & 2 \end{bmatrix},$$

therefore, it is evident that $(2I+\alpha B)$ and $(2I-\alpha B)$ are tridiagonal Toeplitz matrices. Using that $(2I+\alpha B)^{-1}\cdot(2I+\alpha B) = I$, we obtain the following matrix equation for the Crank-Nicolson method

$$V_j = (2I+\alpha B)^{-1}(2I-\alpha B)V_{j-1}. \tag{8}$$

The algorithm is illustrated in figure (3).

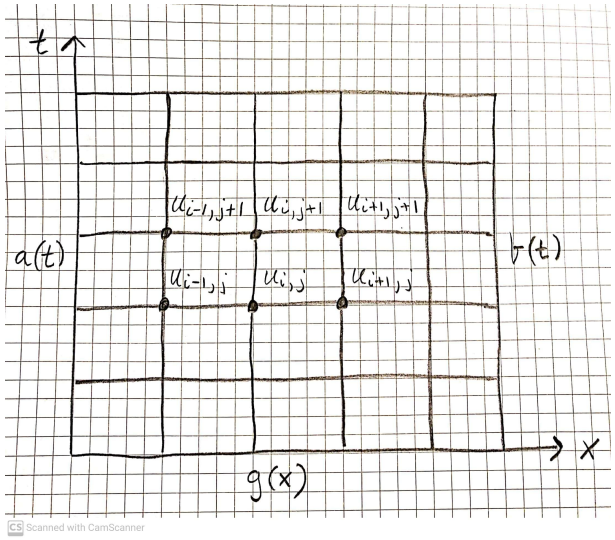

Figure 3. Discretization of integration area (x,t) of the one dimensional diffusion equation $u(x,t)$ (Hjorth-Jensen, 2015, pp. 310-313)[2], using the Crank-Nicolson scheme.

D. Diffusion equation in two dimensions

In two dimensions, the diffusion equation (1) is

$$\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)u(x,y,t) = \frac{\partial}{\partial t}u(x,y,t). \tag{9}$$

For this problem we will limit ourselves to the explicit forward Euler method (Hjorth-Jensen, 2015, p. 315)[2]. We define a square $L\times L$ lattice in the $xy-$plane, with $L=1$. This way, $x,y\in[0,1]$. To simplify further, we assume that x and y have the same step length: $\Delta x = \Delta y = h$, which is

$$h = \frac{1}{n+1},$$

using $n+1$ grid points for x and y. The discretization of positions and time is then

$$\begin{aligned} x_i &= x_0 + ih, \\ y_j &= y_0 + jh, \\ t_l &= t_0 + l\Delta t, \end{aligned}$$

where $x_0 = y_0 = t_0 = 0$. To simplify notation, we let

$$u(x_i, y_j, t_l) = u^l_{i,j}.$$

Our second derivative with respect to x then reads

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u^l_{i+1,j} + u^l_{i-1,j} - 2u^l_{i,j}}{h^2},$$

similarly for y, this reads

$$\frac{\partial^2 u}{\partial y^2} \approx \frac{u^l_{i,j+1} + u^l_{i,j-1} - 2u^l_{i,j}}{h^2}.$$

For the time derivative, we use Euler's forward formula. This reads

$$\frac{\partial u}{\partial t} \approx \frac{u^{l+1}_{i,j} - u^l_{i,j}}{\Delta t}.$$

Defining $\alpha = \Delta t/h^2$, and inserting these approximations into equation (9), and solving for the next forward time step $u^{l+1}_{i,j}$, we get

$$\begin{aligned} u^{l+1}_{i,j} &= u^l_{i,j} \\ &+ \alpha\left(u^l_{i+1,j} + u^l_{i-1,j} + u^l_{i,j+1} + u^l_{i,j-1} - 4u^l_{i,j}\right), \end{aligned} \quad (10)$$

Truncation errors and stability requirements for this method is discussed in the end of subsection (II E), presented in equation (11).

E. Truncation errors and stability

The truncation errors and stability criterion for the methods discussed in (II C) are given (Hjorth-Jensen, 2015, p. 312)[2] by table (I).

In terms of truncation error, the Crank-Nicolson scheme is better overall, while the forward and backward Euler methods are equally good. Notice how the implicit scheme are always stable, for all Δx and Δt. The explicit method has a stability criterion (3) which links the step sizes in position and time. To understand where the stability criterion comes from, we have to look at the matrices involved in equations (6), (7) and (8) for each of the three PDE-solvers. All of the implicit methods can be written on the form:

Table I. Truncation error and stability criterion for PDE-solvers.

PDE-solver	Truncation error	Stability criterion
Forward Euler	$O(\Delta x^2)$ and $O(\Delta t)$	$\Delta t \leq 0.5\Delta x^2$
Backward Euler	$O(\Delta x^2)$ and $O(\Delta t)$	None
Crank-Nicolson	$O(\Delta x^2)$ and $O(\Delta t^2)$	None

$$\hat{A} \cdot \vec{u} = \vec{r},$$

where $\hat{A}$ is the tridiagonal Toeplitz matrix and $\vec{r}$ is known. If we require our solution $\vec{u}$ to approach a definite value after some amount of time steps, the matrix $\hat{A}$ must satisfy:

$$\rho(\hat{A}) < 1,$$

where $\rho(\hat{A})$ is the spectral radius of $\hat{A}$. It is the smallest radius at which a circle centered at the origin in the complex plane contains all the eigenvalues λ of $\hat{A}$. The spectral radius is therefore defined as

$$\rho(\hat{A}) = max\{|\lambda| : \ det(\hat{A} - \lambda I) = 0\}.$$

If the matrix $\hat{A}$ is positive definite, this is always fulfilled. For that reason, there is no stability criterion for the implicit methods, because their respective matrices are always positive definite. For the explicit methods, we have to make sure that (3) is satisfied.

When we apply the explicit forward Euler method on the two dimensional diffusion equation, with step lengths $\Delta x = \Delta y = h$, as in subsection (II D), our stability criterion becomes (Lee, 2017)[4]:

$$\frac{\Delta t}{\Delta x^2} + \frac{\Delta t}{\Delta y^2} = 2\frac{\Delta t}{h^2} \leq \frac{1}{2}, \quad (11)$$

or just

$$\frac{\Delta t}{h^2} \leq \frac{1}{4}.$$

F. Closed-form analytic solutions

The diffusion equation (2) in **one dimension** has closed-form solutions to the continuous problem, which we can obtain analytically (Hjorth-Jensen, 2015, pp. 313-314)[2]. This is shown in appendix (B). The general form of the solution is

$$u(x,t) = \sum_{m=1}^{\infty} A_m \sin\left(\frac{m\pi}{L}x\right) e^{-(m\pi/L)^2 t},$$

with initial conditions for $0 < x < L$:

$$u(x,0) = g(x) = \sum_{m=1}^{\infty} A_m \sin\left(\frac{m\pi}{L}x\right).$$

We use Dirichlet boundary conditions: $u(0,t) = u(L,t) = 0$.

The coefficients A_m is the Fourier coefficients of $g(x)$, and it is therefore determined by the Fourier transformation:

$$A_m = \frac{2}{L}\int_0^L g(x)\sin\left(\frac{m\pi}{L}x\right)dx.$$

To test our PDE-solvers, we have chosen to derive an analytical solution of the diffusion equation using $g(x) = 1$. This way, the expression for the coefficients A_m becomes

$$\begin{aligned} A_m &= \frac{2}{L}\int_0^L \sin\left(\frac{m\pi}{L}x\right)dx \\ &= -\frac{2}{m\pi}\left[\cos\left(\frac{m\pi}{L}x\right)\right]_0^L \\ &= \frac{2}{m\pi}\left(1-\cos(m\pi)\right). \end{aligned}$$

The analytical solution is then

$$u(x,t) = \frac{2}{\pi}\sum_{m=1}^{\infty}\frac{1-\cos(m\pi)}{m}\sin\left(\frac{m\pi}{L}x\right)e^{-(m\pi/L)^2 t}. \tag{12}$$

We will compare this analytical solution to the solution obtained from the Crank-Nicolson scheme. In order to achieve a high level of accuracy, we want to use as large m as possible. We let $m_{max} = M$, such that the analytical solution is

$$u(x,t) = \frac{2}{\pi}\sum_{m=1}^{M}\frac{1-\cos(m\pi)}{m}\sin\left(\frac{m\pi}{L}x\right)e^{-(m\pi/L)^2 t}. \tag{13}$$

The **two-dimensional** diffusion equation has analytical solutions as well. This solution is derived in appendix (B).

We use boundary conditions:

$$\begin{aligned} u(0,y,t) &= 0, \\ u(x,0,t) &= 0, \\ u(1,y,t) &= 0, \\ u(x,1,t) &= 0, \end{aligned}$$

and initial conditions for $0 < x, y < 1$ as $u(x,y,0) = 1$. Figure (4) illustrate initial and boundary conditions.

Equation (B1) is based on the boundary and initial conditions presented above. In our case, $f(x,y) = 1$ is our initial conditions, and we can use this to calculate the Fourier coefficients A_{mn} given by equation (B2). These coefficients are

$$\begin{aligned} A_{mn} &= \frac{4}{L^2}\int_0^L\int_0^L \sin\left(\frac{m\pi x}{L}\right)\sin\left(\frac{n\pi x}{L}\right)dxdy \\ &= \frac{4}{L^2\pi^2 mn}(1-\cos(m\pi))(1-\cos(n\pi)), \end{aligned}$$

and then we use these to calculate the final specialized analytical solution:

$$u(x,y,t) = \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} A_{mn}\sin\left(\frac{m\pi x}{L}\right)\sin\left(\frac{n\pi y}{L}\right)e^{-\lambda^2 t}, \tag{14}$$

where

$$\lambda^2 = \left(\frac{\pi}{L}\right)^2(m^2+n^2).$$

Like in the one-dimensional case, we have to limit ourselves to some finite numbers M and N in the summations above (14). Such that

$$u(x,y,t) = \sum_{n=1}^{N}\sum_{m=1}^{M} A_{mn}\sin\left(\frac{m\pi x}{L}\right)\sin\left(\frac{n\pi y}{L}\right)e^{-\lambda^2 t}. \tag{15}$$

III. METHOD

A. Numerical algorithms for solving the diffusion equation

Introducing algorithms for explicit and implicit methods for solving the diffusion equation. For implicit

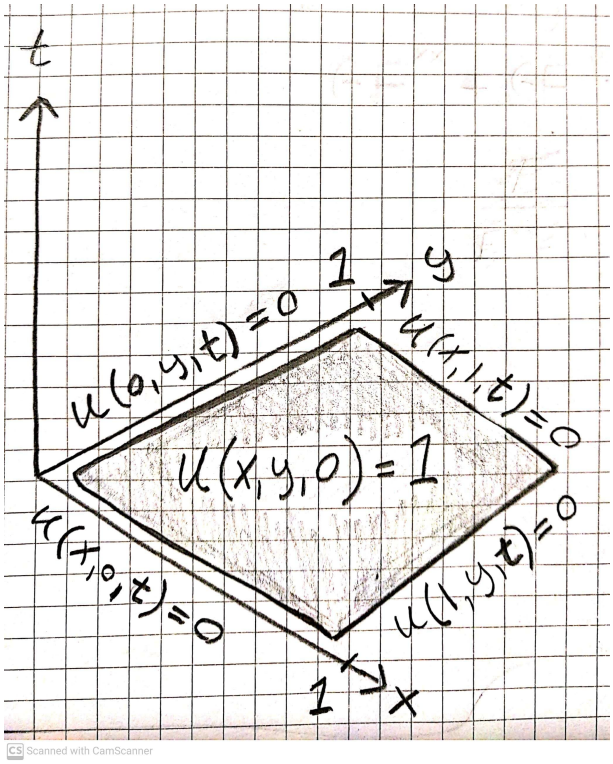

Figure 4. Initial and boundary conditions for the two dimensional diffusion equation illustrated.

methods we use an efficient algorithm for solving linear matrix equations numerically (II C), using forward and backward substitution by Gaussian elimination. The derivation of the algorithm is shown on a tridiagonal matrix in appendix (A).

We have obtained matrix equations for the one dimensional diffusion equation, using three different solvers for partial differential equations. All of the methods involve some kind of tridiagonal Toeplitz matrix, which we can exploit.

- Explicit Forward Euler: (6).
- Implicit Backward Euler: (7).
- Crank-Nicolson scheme: (8).

The explicit method are the easiest to implement. When integrating numerically, the next step $j+1$ is determined by the previous step j, such that

$$V_{j+1} = AV_j.$$

In both the one dimensional and two dimensional diffusion equation case, there is no need for involving tridiagonal matrices in order to solve diffusion equation. This is mainly the reason these methods are easier to implement numerically. The explicit algorithm for the two dimensional diffusion equation is given by equation (10). But, as mentioned earlier, we have to make sure that the stability criterion (3), (11) are fulfilled.

The implicit methods involve inverting matrices. However, storing large matrices and then inverting them results in larger use of memory and CPU-time. Instead, we want to solve these equations using the forward and backward substitution algorithm for tridiagonal matrices (A). This way, there is no need for storing matrices. The backward Euler method has a matrix equation on the form

$$AV_j = V_{j-1},$$

where we can solve for V_j using the forward and backward algorithm.

The Crank-Nicolson scheme has a matrix equation on the same form

$$(2I + \alpha B)V_j = (2I - \alpha B)V_{j-1}.$$

Since V_{j-1} is known, and the entire right side is just a column vector, we can solve this equation for V_j using the forward and backward algorithm. The algorithm goes like this: Starting off with a matrix equation on the form

$$AV_j = V_{j-1},$$

where we have a Toeplitz tridiagonal matrix A on the form

$$A = \begin{bmatrix} b & a & 0 & 0 & ... & 0 \\ a & b & a & 0 & ... & 0 \\ 0 & a & b & a & ... & 0 \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & ... & 0 & a & b \end{bmatrix}.$$

To solve the equation for V_j, we use preform Gaussian elimination (forward substitution) on the matrix (A, V_{j-1}). Since V_{j-1} is known and given by

$$V_{j-1} = \begin{bmatrix} u_{1,j-1} \\ u_{2,j-1} \\ ... \\ u_{n,j-1} \end{bmatrix},$$

we see that

$$(A, V_{j-1}) = \begin{bmatrix} b & a & 0 & 0 & ... & 0 & u_{1,j-1} \\ a & b & a & 0 & ... & 0 & u_{2,j-1} \\ 0 & a & b & a & ... & 0 & u_{3,j-1} \\ ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & ... & 0 & a & b & u_{n,j-1} \end{bmatrix}.$$

Using the forward substitution algorithm, we bring this matrix on reduced form, such that

$$(A, V_{j-1}) \sim \begin{bmatrix} \tilde{b}_1 & \tilde{a}_1 & 0 & 0 & ... & 0 & \tilde{u}_1 \\ 0 & \tilde{b}_2 & \tilde{a}_2 & 0 & ... & 0 & \tilde{u}_2 \\ 0 & 0 & \tilde{b}_3 & \tilde{a}_3 & ... & 0 & \tilde{u}_3 \\ ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & ... & 0 & 0 & \tilde{b}_n & \tilde{u}_n \end{bmatrix}.$$

The algorithm for bringing the matrix on this form goes like:

$$\begin{aligned} \tilde{b}_1 &= b_1, \\ \tilde{a}_1 &= a_1, \\ \tilde{u}_1 &= u_{1,j-1}, \end{aligned}$$

then we update for $j = 2, ..., n$:

$$\tilde{a}_i = \tilde{b}_{i-1},$$

$$\tilde{b}_i = \frac{b}{a}\tilde{b}_{i-1} - \tilde{a}_{i-1},$$

$$\tilde{u}_i = \frac{u_i}{a}\tilde{b}_{i-1} - \tilde{u}_{i-1}.$$

Solving these equations, we obtain a set of equations for $u_{1,j}, u_{2,j}, ..., u_{n,j}$ which can be solved by backward substitution, meaning we solve for $u_{n,j}$ first:

$$u_{n,j} = \frac{\tilde{u}_n}{\tilde{b}_n},$$

then for $j = n-1, ..., 1$:

$$u_{i,j} = \frac{\tilde{u}_i - \tilde{a}_i u_{i+1,j}}{\tilde{b}_i}.$$

This way, we obtain the solution for the new time j, presented earlier in equation (5), which is

$$V_j = \begin{bmatrix} u_{1,j} \\ u_{2,j} \\ ... \\ u_{n,j} \end{bmatrix}.$$

B. Implementation of numerical methods

We start by implementing the algorithms(Euler forward, Euler backward and Crank-Nicolson) in one dimension. We start off by creating a vector u and u_{new} for the current and previous timestep. This is done using the armadillo library (Armadillo, 2020)[5], when we initialize the class.

```
u = unew = zeros(n+1);
```

For all of the methods we set the initial conditions to be the same.

```
//Boundary conditions g(x)
  for (int i = 1; i < n; i++) {
    u(i) = func(dx*i);
  }

  // Boundary conditions a(t) b(t) (zero here)
  unew(n) = u(n) = u(0) = unew(0) = 0;
```

We set the function $g(x)$ to return one at all points to compare with the closed-form solution.

```
float diffusion::func(float x){
  return 1;
}
```

We then write the first step to file. For Euler-Forward we set the diagonal elements

```
  a = alpha;
  b = (1-2*alpha);
```

and start iterating over timesteps.

```
int k = 0;
  //going trough timesteps
  for (int t = 1; t <= tsteps; t++) {
    //going trough xsteps
    for (int i = 1; i < n; i++) {
    // Discretized diff eq
      unew(i) = a * u(i-1) + b * u(i) + a * u(i+1);
    }
    //setting boundary and updating u
    unew(0) = unew(n) = 0;
    u = unew;

    //printing to file
    if (k==saved_tsteps){
      writetofile(outfile,u);
      k=0;
    }
    else{
      k+=1;
    }
  }
```

We go over all values in u_{new} with our Forward algorithm as seen in equation (4). We set up a counter k, so that we can have an interval between the number of timesteps we write to file. The function *writetofile*, writes a timestep u to file.

For the Backward-Euler method we need to do an invertion operation. We have a tridiagonal matrix and use the algorithms as described in the section above.

```
void tridiag_solver::forward_solver(vec &v_vec, vec &g_vec){
  //Setting inital values for the tilde vectors
  b_tilde(0) = b; a_tilde(0) = a; g_tilde(0) = g_vec(0);


  //precalculating b/a
  float b_over_a = b*1./a;
  //Solving the forward algorithm, specialized
  for(int i=1;i<n+1;i++){
    a_tilde(i) = b_tilde(i-1);
    b_tilde(i) = b_over_a*b_tilde(i-1)-a_tilde(i-1);
    g_tilde(i) = g_vec(i)*b_tilde(i-1)*1./a    - g_tilde(i-1);
  }
}

void tridiag_solver::backward_solver(vec &v_vec, vec &g_vec){
  //Setting up the endpoint
```

```
  v_vec(n) = g_tilde(n)*1./b_tilde(n);

  //Backward algorithm
  for(int j=n-1;j>=0;j--){
    v_vec(j) = (g_tilde(j) - a_tilde(j) *v_vec(j+1))*1./b_tilde(j);
  }
}
```

This program is specialised for the case when the diagonal elements are equal ($a = c$), to make the computation times faster. As seen in the code, this program takes in the address of the vectors and will therefore change them without returning it. For the Backward-Euler we therefore need to create a pointer to this class, giving it the length of the vector and the diagonal elements.

```
  tridiag_solver *solver;
  solver = new tridiag_solver(n,a,b);
```

We now implement the Backward-Euler time iteration. The difference between the Forward and Backward functions is that we now have to change the diagonal elements

```
a = -alpha;
b = (1+2*alpha);
```

and that we don't run trough the elements in u, but rather sends u to the Forward/Backward solver in tridiag.

```
for (int t = 1; t <= tsteps; t++) {
    //forward-backward solving
    solver->forward_solver(unew,u);
    solver->backward_solver(unew,u);

    //setting boundary and updating u
    unew(0) = unew(n) = 0;
    u = unew;
```

For Crank-Nicolson we also change the diagonal elements

```
a = - alpha;
b = 2 + 2*alpha;
```

but now instead solve an equation for r before solving $Au = r$. We also use armadillo to set up a vector r with length $n + 1$. We set up a class pointer in the same way as the Backward-Euler method.

```
  for (int t = 1; t <= tsteps; t++) {
// Calculate r for use in tridag,
//right hand side of the Crank Nicolson method
for (int i = 1; i < n; i++) {
  r(i) = alpha*u(i-1) + (2 - 2*alpha)*u(i) + alpha*u(i+1);
}
r(0) = 0;
r(n) = 0;
// Then solve the tridiagonal matrix
solver -> forward_solver(u,r);
solver -> backward_solver(u,r);
//setting boundary
u(0) = 0;
u(n) = 0;
```

We will look at the methods against the analytical solution described in the subsection (II F). We make sure that the stability criteria $\alpha \leq 1/2$ as described in the assignment text (Hjorth-Jensen, 2020)[1]. We base Δt on this criterion and Δx, so that $\Delta t = \alpha \Delta x^2$. We will look at the implication of varying Δx.

We will implement the analytical solution in python, since we will do the comparrison and plotting there.

```
def u_analytic(t_,m_max,n,x):
    u_analytic_ = np.zeros(n+1)
    for m in range(1,m_max):
        Am = 2/(m*np.pi)*(1-np.cos(m*np.pi))
        u_analytic_+=Am*np.sin(m*np.pi*x)*np.exp(-m**2*np.pi**2*t_)
    return u_analytic_
```

Here we have that $M = m_{max}$, t is the timestep, n is the size of the matrix and x is a linspace between zero and one ($x \in [0, L]$). We use numpy to set up the arrays and matplotlib for the plotting. We will look at the different methods against eachother and the error in the methods against the analytical solution. To find the error in the methods we create a function

$$\varepsilon(t) = \sum_{i=1}^{n} \frac{|u_{i,analytic} - u_{i,numerical}|}{n} \tag{16}$$

to give us the average error of $u_{numerical}$ at a timestep j.

Moving on to two dimensions we will use the explicit scheme. We set up a matrix for the next timestep u_{new} and for the current u using armadillo. (Armadillo, 2020)[5].

```
//setting up matrixes
  mat u_, unew_;
  u_ = unew_ = mat(n+1,n+1);
```

Then we initialize the matrices, setting the boundaries to zero and the function $f(x, y) = 1$ at all points. This is for the same reason as last time, to create an easy analytical solution.

```
  //setting boundary
  for(int i=0;i<n+1;i++){
    u_(0,i) = 0;
    u_(n,i) = 0;
    u_(i,0) = 0;
    u_(i,n) = 0;
  }
  //Setting initial conditions
  for(int i=1;i<n;i++){
    for(int j=1;j<n;j++){
      u_(i,j) = func(1);
    }
  }
  unew_ = u_;
```

We write the initial condition to file using a function *writetofile2d*, this function takes in the u-matrix as an argument and writes it to file. We then implement the algorithm as shown in equation (10).

```
for (int t = 1; t <= tsteps; t++) {
    //Forward algorithm
    for(int i=1;i<n;i++){
      for(int j=1;j<n;j++){
        unew_(i,j) = u_(i,j) + alpha*(u_(i+1,j)+u_(i-1,j)+
        u_(i,j+1)+u_(i,j-1)-4*u_(i,j));
      }
    }

    //setting boundary and updating u
    for(int i=0;i<n+1;i++){
      unew_(0,i) = 0;
      unew_(n,i) = 0;
      unew_(i,0) = 0;
      unew_(i,n) = 0;
    }

    u_ = unew_;
```

```
    //then printing u_ to file
    ...
}
```

To check this solution we create the analytical solution in python as described in subsection (II F). Since we set $\Delta x = \Delta y$, we get the stability criteria from equation (11), which is that $\alpha \leq 1/4$.

```
def F(x,y,n,m):
    return np.sin(m*np.pi*x)*np.sin(n*np.pi*y)

def G(t,lmd):
    return np.exp(-lmd**2 * t)

def u_analytic(x,y,m_max, t_, n_len):
    u_analytic = np.zeros((n_len+1,n_len+1))
    n_max = m_max
    for i in range(0,n_len+1):
        for j in range(0,n_len+1):
            for m in range(1,m_max):
                for n in range(1,n_max):
                    lmd = np.pi * np.sqrt(m**2+n**2)
                    A_nm = 4/(n*m*np.pi**2) * (1-np.cos(m*np.pi))*
                    (1-np.cos(n*np.pi))
                    u_analytic[i,j] += A_nm * F(x[i],y[j],n,m)
                    * G(t_, lmd)

    return u_analytic
```

Since this is an approximation of the analytical solution we will have to run with as high M as possible, but the processing time will be higher with the double sum. We let the sums go up to the same values. Here we will also create a epsilon function to evaluate the numerical solution.

$$\varepsilon(t) = \sum_{i=1}^{n}\sum_{j=1}^{m} \frac{|u_{i,j,analytic} - u_{i,j,numerical}|}{nm}, \qquad (17)$$

where i and j is the indexes of the discretized solution. We set up the matrix so that $n = m$. This will give us the average error of $u_{numerical}$ at each time step.

IV. RESULTS

A. One dimensional diffusion equation

When we have a fixed position step Δx, we use the stability criterion for the Forward scheme (3) to determine the time step Δt. This way, all the schemes have the same step sizes when compared. In this part we look at results obtained from solving the one dimensional $(1+1)$ diffusion equation (2), using three schemes: Forward Euler, Backward Euler and Crank-Nicolson. We have compared these methods in terms of stability and accuracy. Using $\Delta x = 0.1$ and $\Delta t = 5 \cdot 10^{-3}$:

- Explicit Forward Euler: Figure (5).
- Implicit Backward Euler: Figure (6).
- Implicit Crank-Nicolson: Figure (7).

These are three dimensional plots. The results are roughly equal, only minor differences between the schemes. The stability criterion for the explicit scheme is fulfilled. We have compared the solutions at to different times in a two dimensional plot to see the differences, shown in figure (8).

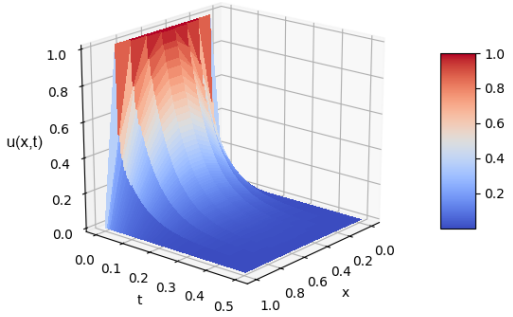

Figure 5. Forward Euler solution of the one dimensional diffusion equation (2). Using $\Delta x = 0.1$ and $\Delta t = 0.005$.

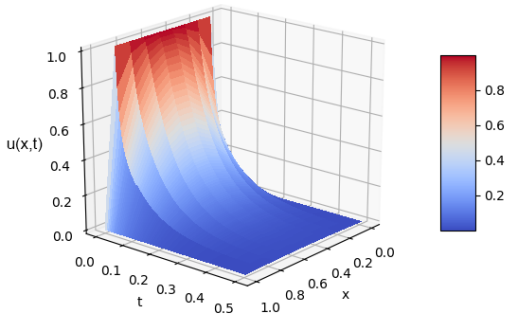

Figure 6. Backward Euler solution of the one dimensional diffusion equation (2). Using $\Delta x = 0.1$ and $\Delta t = 0.005$.

We have done similar comparisons using $\Delta x = 0.01$ and $\Delta t = 5 \cdot 10^{-5}$:

- Explicit Forward Euler: Figure (9).
- Implicit Backward Euler: Figure (10).
- Implicit Crank-Nicolson: Figure (11).

Because of higher resolution, the plots are less choppy. The curves $u(x,t)$ are smooth, meaning that our numerical solutions are better approximations. When

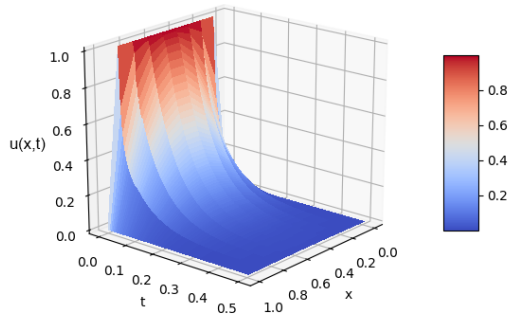

Figure 7. Crank-Nicolson solution of the one dimensional diffusion equation (2). Using $\Delta x = 0.1$ and $\Delta t = 0.005$.

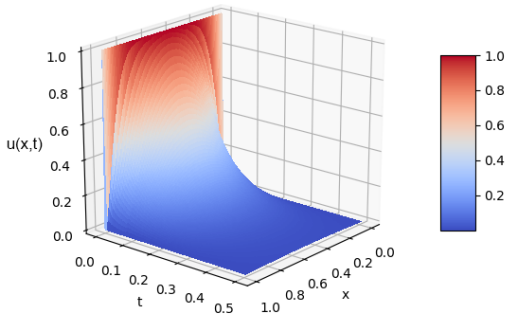

Figure 9. Forward Euler solution of the one dimensional diffusion equation (2). Using $\Delta x = 0.01$ and $\Delta t = 5 \cdot 10^{-5}$.

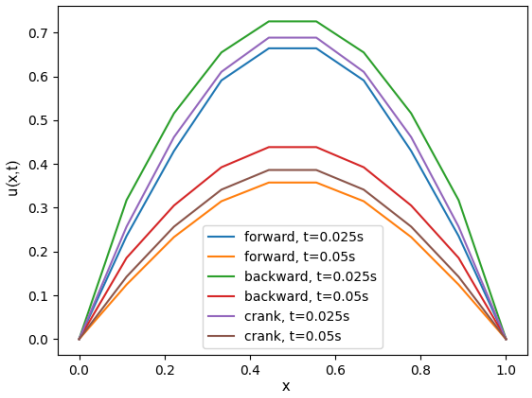

Figure 8. Comparing results from three schemes at two time points. Solution of the one dimensional diffusion equation (2). Using $\Delta x = 0.1$ and $\Delta t = 0.005$.

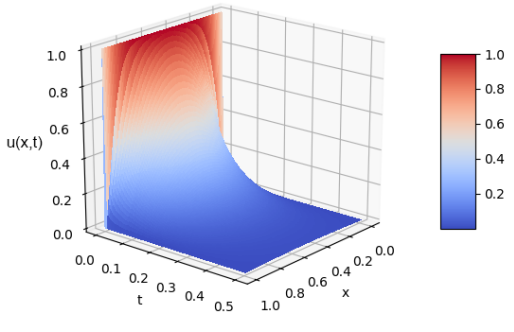

Figure 10. Backward Euler solution of the one dimensional diffusion equation (2). Using $\Delta x = 0.01$ and $\Delta t = 5 \cdot 10^{-5}$.

comparing the schemes at this resolution there are no visible differences, as shown in figure (12).

We have used the analytical solution (13) with different M to compare with our numerical approximations. First, we have used $\Delta x = 0.1$ and $\Delta t = 0.005$. In figure (13), we see that the numerical approximation using the Crank-Nicolson scheme is off by roughly 30% at time $t = 0.25$. In one dimension, we also see that the analytical solution is good even with $M = 1$. This is due to the shape of the curve $u(x, t)$, which is quite sinusoidal and therefore easy to approximate with Fourier series. Even so, for the upcoming comparisons, we have used $M = 100$ to make sure that the analytical solution has adequate precision. Looking at a earlier time, $t = 0.025$, we see however that all of our numerical approximations are quite good when compared to the analytical solution, shown in figure (14). To confirm

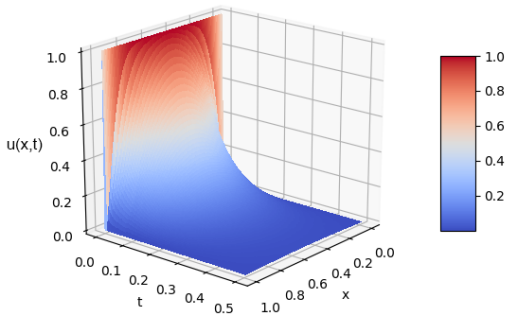

Figure 11. Crank-Nicolson solution of the one dimensional diffusion equation (2). Using $\Delta x = 0.01$ and $\Delta t = 5 \cdot 10^{-5}$.

this, we have used equation (16) to calculate the average

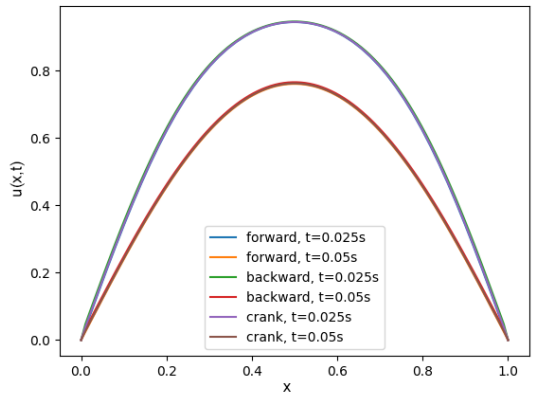

Figure 12. Comparing results from three schemes at two time points. Solution of the one dimensional diffusion equation (2). Using $\Delta x = 0.01$ and $\Delta t = 5 \cdot 10^{-5}$.

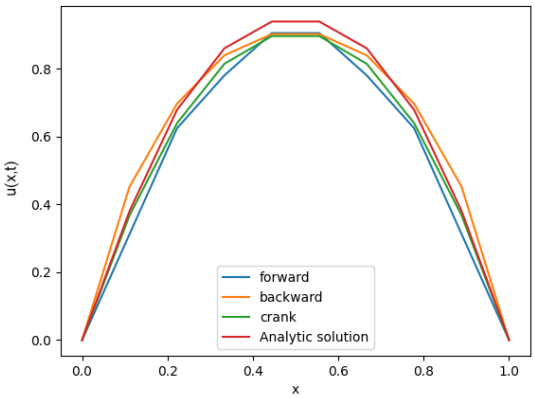

Figure 14. Numerical approximations vs analytical solution: The one dimensional diffusion equation (2), at time $t = 0.025$. Using $\Delta x = 0.1$, $\Delta t = 0.005$ and $M = 100$.

error in each time step. This is shown in figures (15), (16) and (17). We see that the average error is roughly the same for all schemes at this resolution, and as time increases, the average error decreases. This is because both the analytical and numerical solutions goes to zero. The Backward Euler method seems to have slightly less error when looking at early developments in time (15).

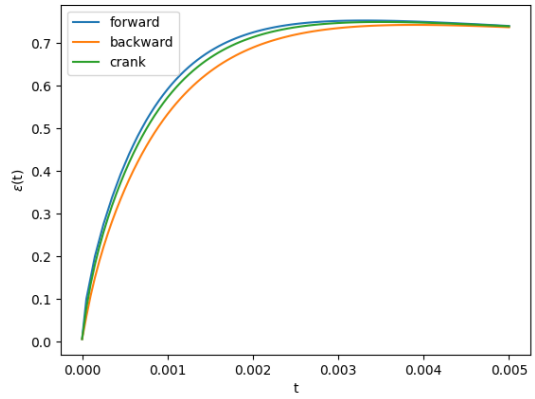

Figure 15. Average error (16) between our numerical approximations and the analytical solution. One dimensional diffusion equation (2). Using $\Delta x = 0.1$, $\Delta t = 0.005$ and $M = 100$.

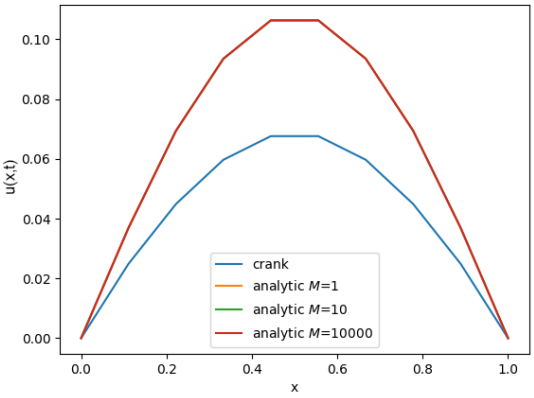

Figure 13. Crank-Nicolson vs analytical solution: The one dimensional diffusion equation (2), at time $t = 0.25$. Using $\Delta x = 0.1$ and $\Delta t = 0.005$.

We have done the same comparisons with higher resolution. Using $\Delta x = 0.01$ and $\Delta t = 5 \cdot 10^{-5}$. In figure (18) we see the comparison between Crank-Nicolson and the analytical solution (13) with different M at time $t = 0.025$. The approximation is good. At this resolution, the analytical solution varies slightly for low M, that is $M < 10$. But there is almost no difference between $M = 10$ and $M = 10^4$. In the upcoming comparisons, we have used $M = 100$. In figure (19), all schemes are compared to the analytical solution at $t = 0.025$. There are almost no difference between the schemes and the analytical solution because of increased resolution. The average error (16) plots in figure (20) and (21) confirms this, showing that the average error in all schemes are almost identical. As time increases, the average error decreases because all the solutions goes to zero.

B. Two dimensional diffusion equation

In this part we look at the results obtained from solving the two dimensional $(2 + 1)$ diffusion equation (9). For this problem, we have chosen to limit ourselves

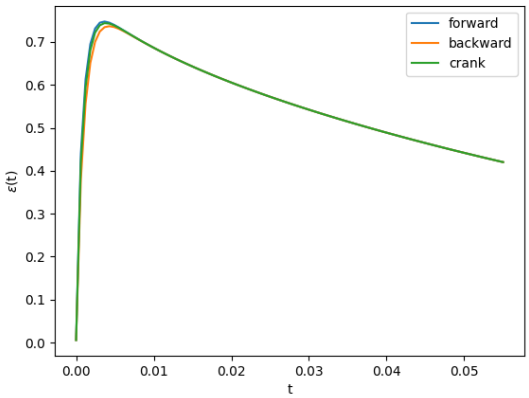

Figure 16. Average error (16) between our numerical approximations and the analytical solution. One dimensional diffusion equation (2). Using $\Delta x = 0.1$, $\Delta t = 0.005$ and $M = 100$.

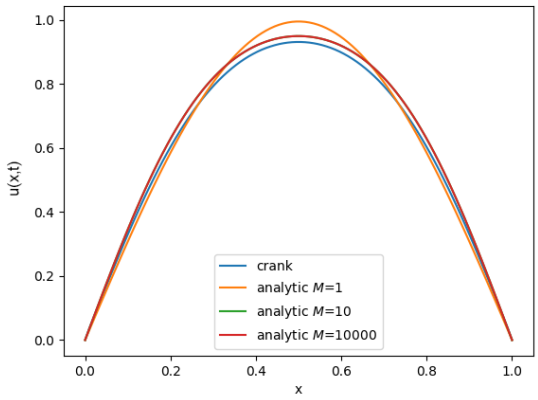

Figure 18. Crank-Nicolson vs analytical solution: The one dimensional diffusion equation (2), at time $t = 0.025$. Using $\Delta x = 0.01$ and $\Delta t = 5 \cdot 10^{-5}$.

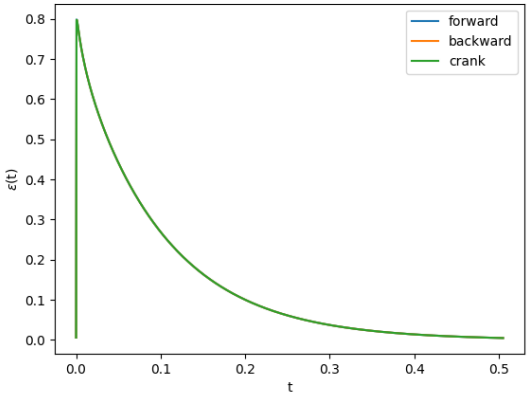

Figure 17. Average error (16) between our numerical approximations and the analytical solution. One dimensional diffusion equation (2). Using $\Delta x = 0.1$, $\Delta t = 0.005$ and $M = 100$.

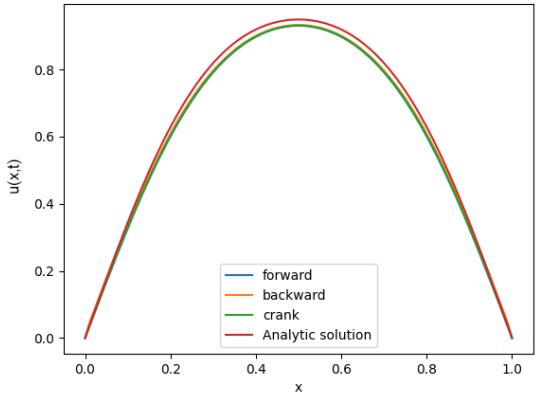

Figure 19. Numerical approximations vs analytical solution: The one dimensional diffusion equation (2), at time $t = 0.025$. Using $\Delta x = 0.01$, $\Delta t = 5 \cdot 10^{-5}$ and $M = 100$.

to the explicit scheme (10). The step sizes h and Δt are decided using the stability criterion defined in equation (11). We have obtained the closed-form analytical expression (15) which we to compare with. First, we look at how the analytical expression depends on the number of Fourier terms. We have used $M = N$ when calculating the Fourier series for the analytical expression.

Using $M = 10$, $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$ (stability criterion for explicit scheme is fulfilled), we see from figure (22) that the analytical solution is quite choppy at the boundary and initial conditions. Ideally the function should have a box-shape where $u(x, y, 0) = 1$ for $x, y \in [0, 1]$. Because of the low amount of Fourier terms, the analytical expression fails to approximate the flat area where the initial conditions are defined. We see that in some places $u(x, y, 0) \approx 1.2$. However, even with a low number M, the solution is much better looking as we move forward in time. This is shown by figure (23) and (24).

We let $M = 100$, thereby increasing the number of Fourier terms by a large amount, the analytical solution is much better at representing the initial conditions. This is important for comparison with the numerical scheme. The results are shown in figure (25), (26) and (27).

Now we look at the results obtained from the numerical approximation, that is the Forward Euler scheme (10). We have used the same integration step sizes as with the analytical solution: $h = 0.048$ and

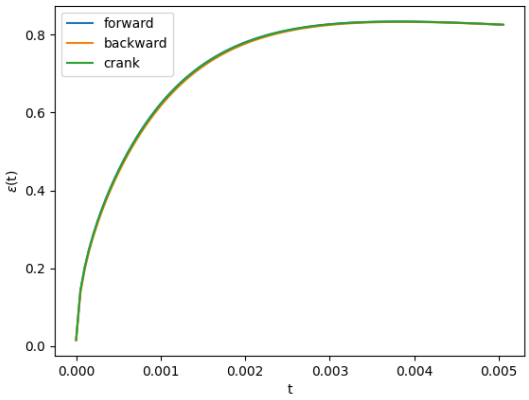

Figure 20. Average error (16) between our numerical approximations and the analytical solution. One dimensional diffusion equation (2). Using $\Delta x = 0.01$, $\Delta t = 5 \cdot 10^{-5}$ and $M = 100$.

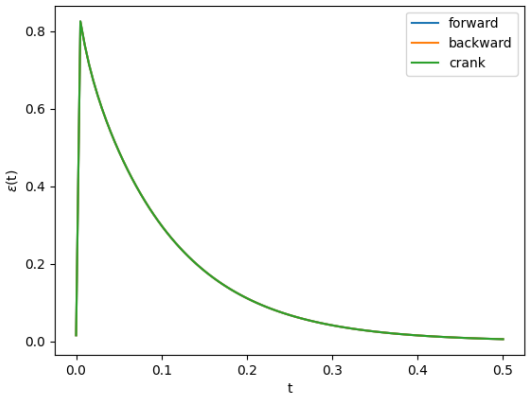

Figure 21. Average error (16) between our numerical approximations and the analytical solution. One dimensional diffusion equation (2). Using $\Delta x = 0.01$, $\Delta t = 5 \cdot 10^{-5}$ and $M = 100$.

$\Delta t = 5.7 \cdot 10^{-4}$, because this fulfils the stability criterion (11). At time $t = 0$, the initial conditions are well represented (box-shape) as we see in figure (28). As we move forward in time, the numerical solution resembles the analytical one quite well, as we see in figure (29) and (30).

We have compared the numerical solution to the analytical solution with $M = 10$ and $M = 100$ by plotting contours of the difference between the analytical and numerical function values at every point in the xy-plane at different times. These plots show the absolute difference between the analytical and numerical solutions. Figure (31), (32) and (33) show these comparisons with $M = 10$ at time $t = 0$, $t = 0.073$

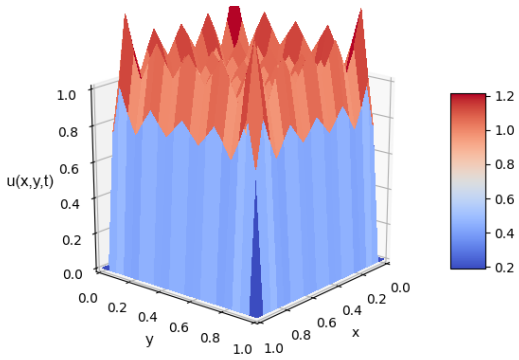

Figure 22. Analytical solution (15) to the two dimensional diffusion equation (9), at time $t = 0$. Using $M = 10$, $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

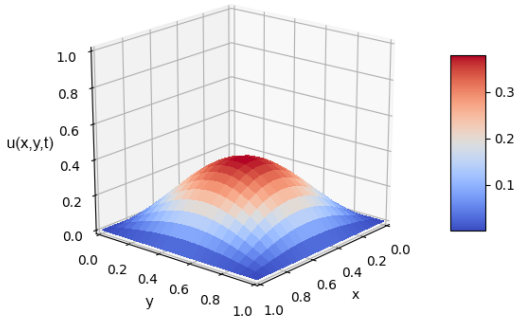

Figure 23. Analytical solution (15) to the two dimensional diffusion equation (9), at time $t = 0.073$. Using $M = 10$, $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

and $t = 0.15$ respectively. At time $t = 0$, we see that there are certain areas where the absolute difference is relatively large, which is expected when comparing figure (22) with figure (28). As we move forward in time, the differences are reduced on the edges of the xy-plane, but still remain in and around the center of the grid. There is a circular region around the center of the grid which represent the area where the difference is largest. As time increases, this area shrinks (the average error decreases). When we use $M = 100$, as shown in figure (25), (26) and (27), we see a clear improvement at the initial conditions when $t = 0$. This is what we expect when looking at figure (25) and (28). As time moves forward, there is no significant difference between the analytical solution using $M = 10$ or $M = 100$ when compared upon the numerical solution.

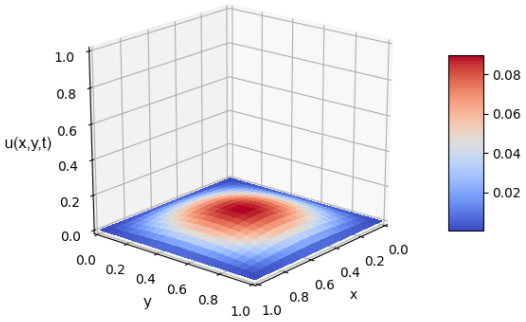

Figure 24. Analytical solution (15) to the two dimensional diffusion equation (9), at time $t = 0.15$. Using $M = 10$, $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

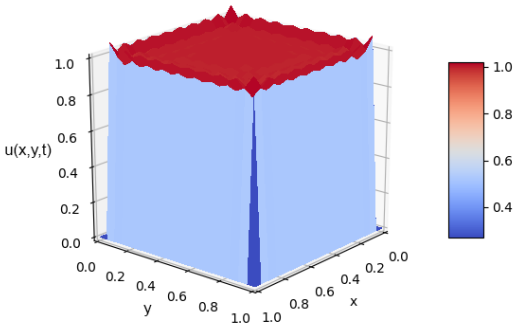

Figure 25. Analytical solution (15) to the two dimensional diffusion equation (9), at time $t = 0$. Using $M = 100$, $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

To calculate the average error in our numerical solution, we have used equation (17). This error calculation relies on having an analytic solution with high accuracy. Using $M = 10$, we know that our analytical solution fails to reproduce the initial conditions. As time moves forward, we have seen that it makes little difference using $M = 10$ or $M = 100$, and the solution seems to be quite good. Therefore we assume that our analytical solution is sufficiently accurate for these calculations. We expect the error to be larger at $t = 0$ than elsewhere when $M = 10$. Figure (37) and (38) shows the average error $\epsilon(t)$ using $M = 10$ and $M = 100$ respectively. The behaviour is as we expect, when $M = 10$ we have an average error $\epsilon(0) \approx 0.08$ which quickly drops to $\epsilon \approx 0.02$, before slowly moving towards zero. When $M = 100$, we have an average error $\epsilon(0) \approx 0.15$, which rises to a maximum value of $\epsilon \approx 0.02$ before moving towards zero. Since the numerical solution is the same

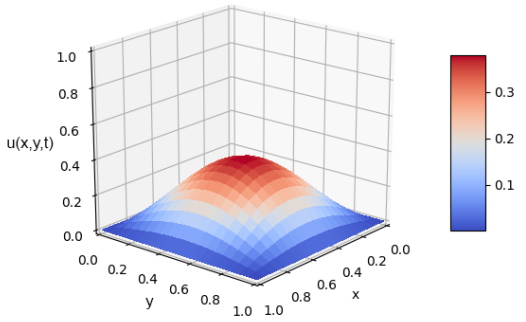

Figure 26. Analytical solution (15) to the two dimensional diffusion equation (9), at time $t = 0.073$. Using $M = 100$, $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

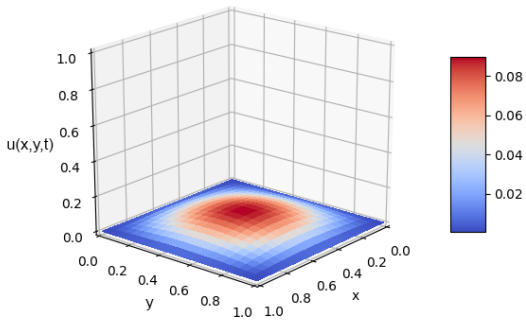

Figure 27. Analytical solution (15) to the two dimensional diffusion equation (9), at time $t = 0.15$. Using $M = 100$, $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

in both cases, this shows that the analytical solutions are quite similar expect at the initial time $t = 0$. The trend in the average error is also the same as it was in the one dimensional case, shown by figure (17) and (21). The average error $\epsilon(t) \to 0$ when $t \to \infty$.

C. Stability of numerical schemes

In this part we look at the stability of the explicit and implicit schemes in one dimension when the stability criterion for the explicit scheme is violated. As discussed in the theory of truncation errors and stability (II E), we know that the implicit schemes are stable for all Δx and Δt, but the explicit scheme have stability requirements given by equation (3) in one dimension. For these

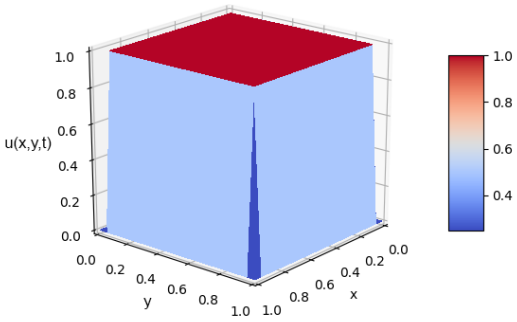

Figure 28. Numerical solution (10) to the two dimensional diffusion equation (9), at time $t = 0$. Forward Euler scheme. Using $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

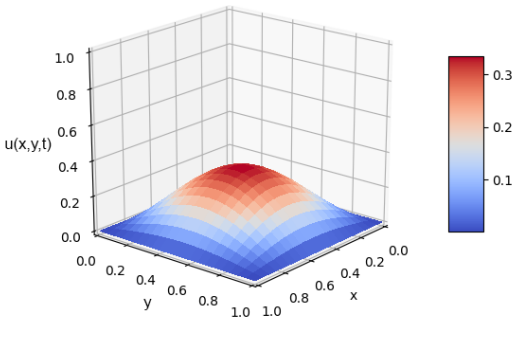

Figure 29. Numerical solution (10) to the two dimensional diffusion equation (9), at time $t = 0.073$. Forward Euler scheme. Using $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

tests we have used $\alpha = 0.6$, where $\alpha = \Delta t/\Delta x^2$ in one dimension. That is $\Delta x = 0.048$ and $\Delta t = 0.0014$. We have compared the numerical solutions with the analytical (13) using $M = 100$, at a time interval $t \in [0, 0.025]$. Figure (39) shows how the explicit forward Euler solution quickly becomes unstable and divergence to infinity when the stability requirements are not fulfilled. Figure (40) and (41) shows the implicit solutions, backward Euler and Crank-Nicolson respectively under the same conditions. The implicit solutions show no sign of instability at this time interval. The average error between the numerical solutions and the analytical solution in figure (42) shows that the error in the explicit scheme quickly rises under these conditions, a contrast to the situation in figure (21) where all schemes fare well.

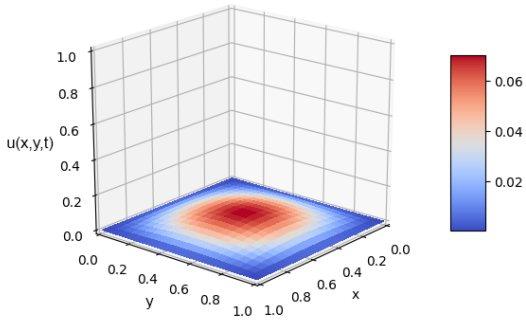

Figure 30. Numerical solution (10) to the two dimensional diffusion equation (9), at time $t = 0.15$. Forward Euler scheme. Using $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

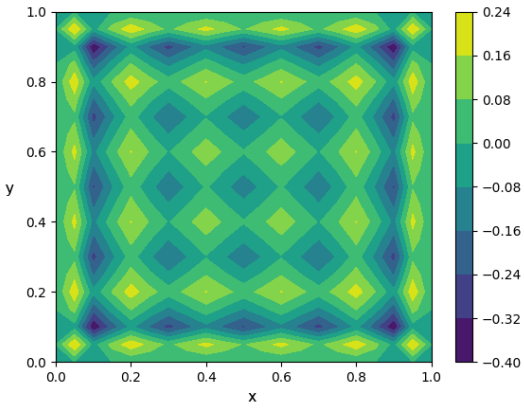

Figure 31. Difference between the numerical (10) and analytical (15) solutions of the two dimensional diffusion equation (9), at time $t = 0$. Using $M = 10$, $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

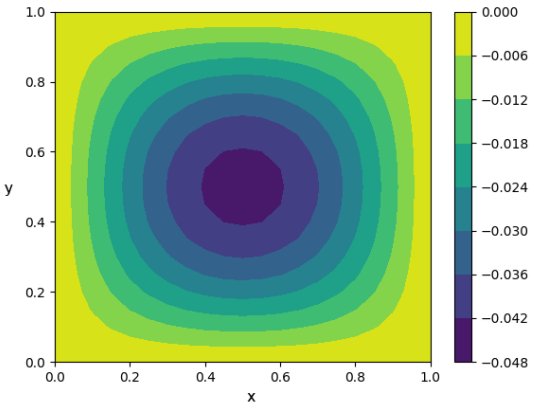

Figure 32. Difference between the numerical (10) and analytical (15) solutions of the two dimensional diffusion equation (9), at time $t = 0.073$. Using $M = 10$, $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

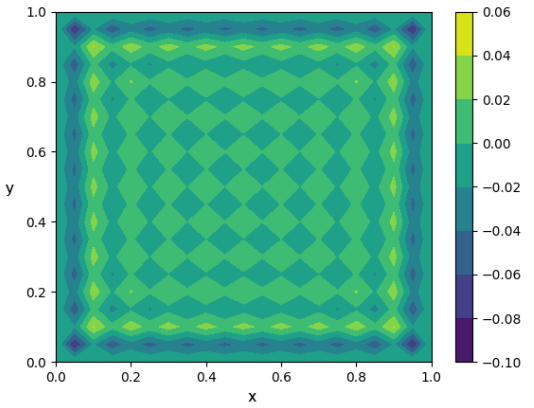

Figure 34. Difference between the numerical (10) and analytical (15) solutions of the two dimensional diffusion equation (9), at time $t = 0$. Using $M = 100$, $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

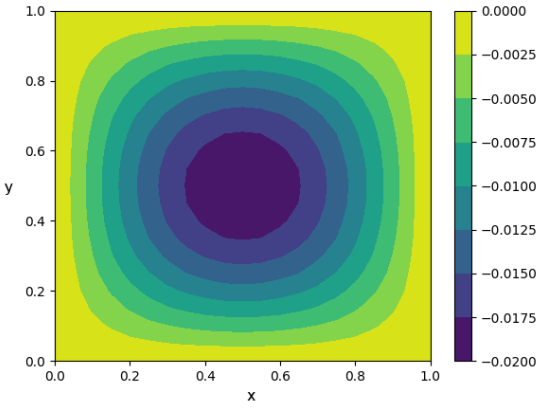

Figure 33. Difference between the numerical (10) and analytical (15) solutions of the two dimensional diffusion equation (9), at time $t = 0.15$. Using $M = 10$, $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

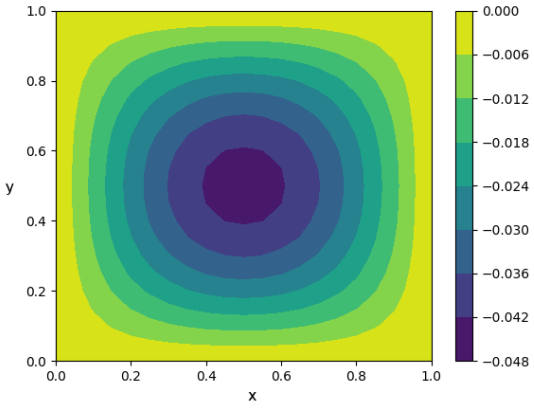

Figure 35. Comparing the numerical (10) and analytical (15) solutions of the two dimensional diffusion equation (9), at time $t = 0.073$. Using $M = 100$, $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

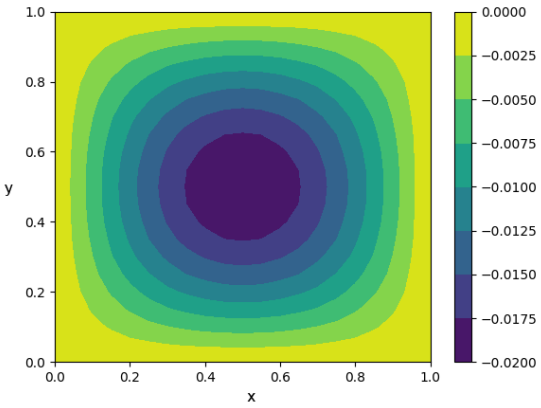

Figure 36. Comparing the numerical (10) and analytical (15) solutions of the two dimensional diffusion equation (9), at time $t = 0.15$. Using $M = 100$, $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

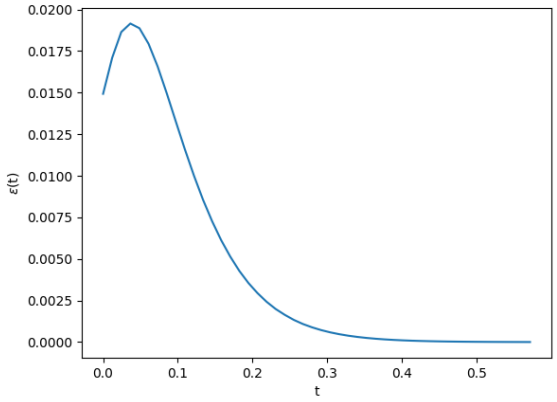

Figure 38. Average error (17) between the Forward Euler solution and the analytical solution. Two dimensional diffusion equation (9). Using $M = 100$, $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

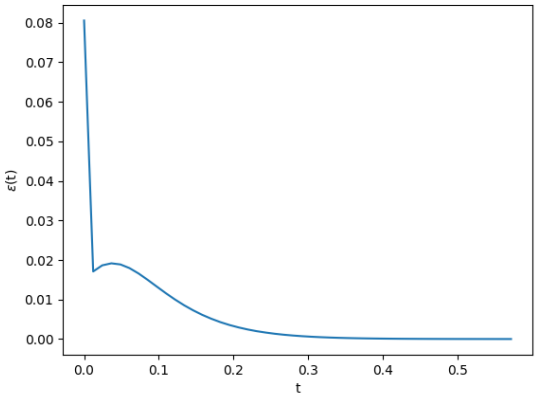

Figure 37. Average error (17) between the Forward Euler solution and the analytical solution. Two dimensional diffusion equation (9). Using $M = 10$, $h = 0.048$ and $\Delta t = 5.7 \cdot 10^{-4}$.

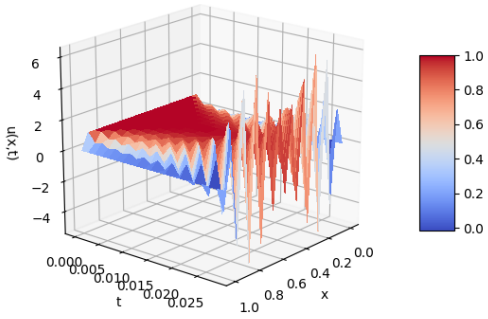

Figure 39. Forward Euler (explicit) solution of the one dimensional diffusion equation (2). Stability criterion (3) for explicit schemes violated. Using $\Delta x = 0.048$ and $\Delta t = 0.0014$.

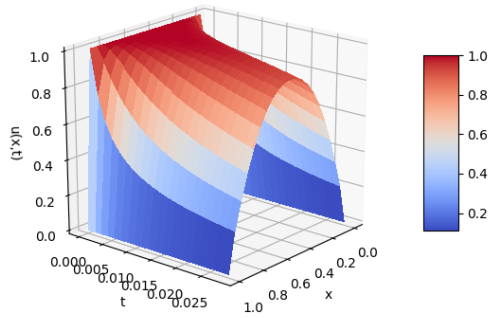

Figure 40. Backward Euler (implicit) solution of the one dimensional diffusion equation (2). Stability criterion (3) for explicit schemes violated. Using $\Delta x = 0.048$ and $\Delta t = 0.0014$

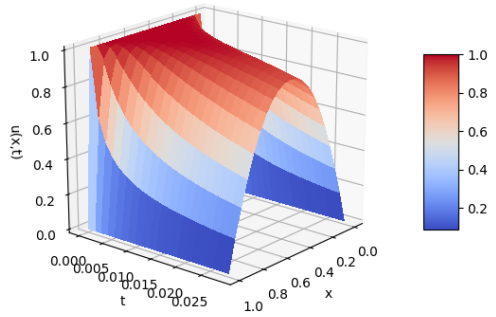

Figure 41. Crank-Nicolson (implicit) solution of the one dimensional diffusion equation (2). Stability criterion (3) for explicit schemes violated. Using $\Delta x = 0.048$ and $\Delta t = 0.0014$

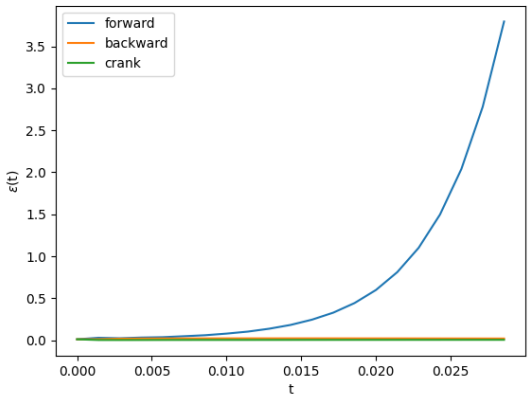

Figure 42. Average error (17) between our numerical solutions and the analytical solution. One dimensional diffusion equation (2). Using $M = 100$, $\Delta x = 0.048$ and $\Delta t = 0.0014$.

V. DISCUSSION

We have not tested a variety of different boundary and initial conditions, but rather chosen to limit ourselves to trivial conditions such as the Dirichlet boundary conditions and uniform initial conditions. These trivial conditions are often necessary for deriving closed-form solutions analytically, which we have used to compare with our numerical results. However, the numerical algorithms allows for much more complex situations than we have tested here.

When solving the implicit schemes we have used an algorithm based on Gaussian elimination. That is the forward and backward substitution introduced in appendix (A). This is an efficient method for solving linear differential equations that can be written as tridiagonal matrix equations (sparse matrices). Alternatively, one could solve these implicit schemes directly using the matrix equations (7) and (8) for the Backward Euler and Crank-Nicolson scheme respectively, by inverting matrices and then preforming matrix multiplications. However, this would require far more floating point operations (more CPU-time) and memory usage. When inverting a tridiagonal matrix the resulting matrix is no longer sparse. With the algorithm we have chosen, there is no need for defining any matrices, nor solving matrix equations directly, which saves memory and CPU-time when solving the implicit schemes. The number of floating point operations involved when using the forward and backward substitution on a $n \times n$ tridiagonal Toeplitz matrix is roughly $9n$.

Taking into consideration that the explicit schemes are easier to implement numerically and faster to solve (because the numerical algorithm involves less operations), it seems to be the better choice as long as the stability criterion is fulfilled. That is one of the main strengths behind explicit schemes. This is evident from looking at the average error $\epsilon(t)$ in figure (21) and (38) for one and two dimensions respectively. In this project, we have been free to choose the integration step sizes without any limitations as this is a pure numerical study. The experience from the one dimensional diffusion equation shows that all methods are roughly equal as long as the stability criterion was fulfilled. Therefore we chose the explicit scheme for solving the two dimensional diffusion equation. The implicit schemes fare much better if the task at hand is to study samples of data where the resolution is fixed. That way, using an implicit scheme grants more freedom to choose integration step sizes in position and time independently because these methods are stable for all step size combinations. The concern in terms of achieving an adequate accuracy of the numerical solutions will depend more on the truncation error.

Figure (42) shows how the average error $\epsilon(t)$ varies between the three different schemes in the one dimensional case, when the stability criterion is violated by a small amount, using $\alpha = \Delta t/\Delta x^2 = 0.6$. This shows how important the stability criteria is for explicit methods. The solution becomes unstable in a short amount of time, clearly visualized in figure (39). In contrast, there is no sign of instability during this time interval for the implicit methods.

The analytical solutions are Fourier series which ideally contains an infinite amount of terms. We have derived analytical expressions for the one and two-dimensional diffusion equation for the continuous problem in appendix (B). In the one-dimensional case, the analytical solution is given by equation (15) when we limit ourselves to some finite number M in the Fourier series. As seen in figure (18), the analytical solution is quite good even when M is low. We have chosen to use $M = 100$ to make sure that our average error calculations (16) are valid because they depend on having accurate closed-form solution as a reference to the numerical results. In the two dimensional case, the analytical solution is given by equation (15). It is more challenging to find good solutions in the two dimensional case because of the double sum in the Fourier series. When M and N are large, it takes a considerable amount of time to calculate the analytical expressions. However, we have seen that using $M = N = 10$ results in quite accurate solutions when compared with the explicit scheme, expect at the initial time $t = 0$. When $M = N = 10$, we see from figure (22) that the initial conditions are far from reproduced. It should look like the Forward Euler solution at $t = 0$ shown in figure (28). Figure (31) shows the difference between these solutions at $t = 0$. By increasing $M = N = 100$, initial conditions are reproduces with an acceptable precision as shown in figure (25) and (34). Using this analytical solution as a reference, we have been able to calculate an average error $\epsilon(t)$ (17) of the Forward Euler solution to the two dimensional diffusion equation, as shown in figure (38).

VI. CONCLUSION

We have seen that the explicit scheme and the implicit schemes are fairly equal in terms of accuracy and stability when using integration step sizes that fulfils the stability requirements for explicit schemes. This is the case when using both low and high resolution, as shown in figure (14) and (19). We have seen that the explicit scheme solution quickly becomes unstable when violating the stability criterion $\Delta t/\Delta x^2 > 1/2$. This is illustrated by figure (39). In contrast, the implicit schemes show no sign of instability under the same conditions, as shown by figure (40) and (41). This highlights the strengths and weaknesses of the different schemes. Explicit schemes are easier to implement, but have additional requirements that must be fulfilled, which limits the possible choices of step sizes. For implicit schemes, we are free to choose any combination of step sizes without having to consider stability criteria, only truncation errors.

When comparing numerical and analytical results, we have seen that the difference between them are small (low average error) in general. In the one dimensional case, the average error is largest at the beginning of the time interval. As time increases, so does the average error. This is illustrated well by figure (21). In the two dimensional case, the average error follows a similar pattern, shown by figure (38). The average error is largest at the beginning because the analytical solution struggles to reproduce accurate initial conditions, as shown in figure (22) and (25). Our interpretation is that we have made an accurate description of the diffusion equation based on these average error calculations, in both one and two dimensions.

In future work it would be interesting to test more complex boundary and initial conditions in one and two dimensions. We have chosen to limit ourselves to cases where we can find analytical expressions to compare with our numerical results, thereby validating the numerical implementations and methods involved. The methods seems to be stable and accurate enough for testing more complex and non-trivial situations. The numerical implementation could be extended to three spacial dimensions as well.

REFERENCES

[1] Hjorth-Jensen, Morten. (2020). FYS3150 - Project 5. Fall semester 2020. University of Oslo. Web-address: http://compphysics.github.io/ComputationalPhysics/doc/Projects/2020/Project5/DiffusionEquation/html/DiffusionEquation.html (December 16, 2020).

[2] Hjorth-Jensen, Morten. (2015). Computational Physics: Lecture notes Fall 2015: "Chapter 10: Partial Differential Equations". Web-address: https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf (December 16, 2020).

[3] Hjorth-Jensen, Morten. (2015). Computational Physics: Lecture notes Fall 2015: "Chapter 6: Linear Algebra". Web-address: https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf (December 16, 2020).

[4] Lee, Jordan. (2017). Stability of Finite Difference Schemes on the Diffusion Equation with Discontinuous Coefficients. Massachusetts Institute of Technology. Web-address: https://math.mit.edu/research/highschool/rsi/documents/2017Lee.pdf (December 16, 2020).

[5] Armadillo C++ library for linear algebra & scientific computing. Web-address: http://arma.sourceforge.net/docs.html (December 16, 2020).

[6] The project's GitHub address. Web-address: https://github.com/Trrn13P/P5 (December 18, 2020)

Appendix A: Solving tridiagonal matrix equations

Introducing a numerical algorithm for solving linear matrix equations involving tridiagonal matrices. First we look at a general tridiagonal matrix. Once this problem is solved, it is easy to specialize to the Toeplitz case. Assuming that we have a real quadratic 4×4 matrix A, and we want to solve the matrix equation:

$$A\vec{x} = \vec{g},$$

where $\vec{x} = (x_1, x_2, x_3, x_4)$ is the unknown column vector and $\vec{g} = (g_1, g_2, g_3, g_4)$ is known. To do this, we use Gaussian elimination (Hjorth-Jensen, 2015, pp. 170-173)[3], on the matrix $B = (A, \vec{g})$. Starting off at

$$B = \begin{bmatrix} b_1 & a_1 & 0 & 0 & g_1 \\ a_1 & b_2 & a_2 & 0 & g_2 \\ 0 & a_2 & b_3 & a_3 & g_3 \\ 0 & 0 & a_3 & b_4 & g_4 \end{bmatrix}.$$

We multiply the second line by b_1/a_1, then subtract the first line from the second line. The first element in the second line becomes 0. Defining the new diagonal element to be $\tilde{b}_2$, given as

$$\tilde{b}_2 = b_2\frac{b_1}{a_1} - a_1.$$

Similarly

$$\tilde{a}_2 = a_2\frac{b_1}{a_1},$$

and

$$\tilde{g}_2 = g_2\frac{b_1}{a_1} - g_1,$$

such that our new matrix becomes

$$B \sim \begin{bmatrix} b_1 & a_1 & 0 & 0 & g_1 \\ 0 & \tilde{b}_2 & \tilde{a}_2 & 0 & \tilde{g}_2 \\ 0 & a_2 & b_3 & a_3 & g_3 \\ 0 & 0 & a_3 & b_4 & g_4 \end{bmatrix}.$$

Next step, we multiply the third line by $\tilde{b}_2/a_2$, and subtract the second line from the third line, resulting in

$$\tilde{b}_3 = b_3\frac{\tilde{b}_2}{a_2} - \tilde{a}_2,$$

and

$$\tilde{a}_3 = \frac{a_3}{a_2}\tilde{b}_2,$$

and lastly

$$\tilde{g}_3 = g_3\frac{\tilde{b}_2}{a_2} - \tilde{g}_2.$$

The new matrix then becomes

$$B \sim \begin{bmatrix} b_1 & a_1 & 0 & 0 & g_1 \\ 0 & \tilde{b}_2 & \tilde{a}_2 & 0 & \tilde{g}_2 \\ 0 & 0 & \tilde{b}_3 & \tilde{a}_3 & \tilde{g}_3 \\ 0 & 0 & a_3 & b_4 & g_4 \end{bmatrix},$$

and the same process is repeated on the last line. Multiplying with $\tilde{b}_3/a_3$, then subtracting the third line from the fourth. Finally we end up with the row-equivalent matrix on reduced form:

$$B \sim \begin{bmatrix} b_1 & a_1 & 0 & 0 & g_1 \\ 0 & \tilde{b}_2 & \tilde{a}_2 & 0 & \tilde{g}_2 \\ 0 & 0 & \tilde{b}_3 & \tilde{a}_3 & \tilde{g}_3 \\ 0 & 0 & 0 & \tilde{b}_4 & \tilde{g}_4 \end{bmatrix}.$$

We generalize this to an $n \times n$ matrix. The algorithm for the new matrix elements is a **forward substitution** and goes as follows: The first values are the same as the original matrix

$$\begin{aligned} \tilde{b}_1 &= b_1, \\ \tilde{a}_1 &= a_1, \\ \tilde{g}_1 &= g_1, \end{aligned}$$

then we update for $j = 2, ..., n$:

$$\tilde{b}_i = b_i\frac{\tilde{b}_{i-1}}{a_{i-1}} - \tilde{a}_{i-1},$$

$$\tilde{a}_i = \frac{a_i}{a_{i-1}}\tilde{b}_{i-1},$$

$$\tilde{g}_i = g_i\frac{\tilde{b}_{i-1}}{a_{i-1}} - \tilde{g}_{i-1}.$$

Solving these equations, we obtain a set of equations for $x_1, x_2, ..., x_n$ which can be solved by **backward substitution**, meaning we solve for x_n first:

$$x_n = \frac{\tilde{g}_n}{\tilde{b}_n},$$

then for $j = n-1, ..., 1$:

$$x_i = \frac{\tilde{g}_i - \tilde{a}_i x_{i+1}}{\tilde{b}_i}.$$

When we have a **Toeplitz tridiagonal matrix**, all diagonal elements are constant, $b_i = b$. In addition, using the matrices involved when solving the diffusion equation (2) as a matrix equation (II C), we have that all $a_i = a$. This simplifies the **forward substitution:**

$$\tilde{a}_i = \tilde{b}_{i-1},$$

$$\tilde{b}_i = \frac{b}{a}\tilde{b}_{i-1} - \tilde{a}_{i-1},$$

$$\tilde{g}_i = \frac{g_i}{a}\tilde{b}_{i-1} - \tilde{g}_{i-1}.$$

Appendix B: Analytic solutions to the diffusion equation

Deriving the closed-form solution to continuous problem for the one dimensional diffusion equation (2) and two dimensional diffusion equation (9) analytically.

First we look at the **one dimensional** case (Hjorth-Jensen, 2015, pp. 313-314)[2]. We want to find a solution $u(x,t)$ to the following equation:

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

with initial conditions $u(x,0) = g(x)$ for $0 < x < L$. The boundary conditions are given by Dirichlet conditions: $u(0,t) = u(L,t) = 0$. Our first step is to assume that the solution can be separated in terms of x and t (separation of variables), which means that

$$u(x,t) = F(x)G(t).$$

Inserting this for $u(x,t)$ in the diffusion equation, we get

$$\frac{F^{(2)}(x)}{F(x)} = \frac{G^{(1)}(t)}{G(t)}.$$

Where $F^{(2)}$ is the second derivative of F with respect to x, and $G^{(1)}$ is the first derivative of G with respect to t. Since the left side only depends on x and the right side only depends on t, both sides must be equal to a constant. We define this constant to be $-\lambda^2$, giving two differential equations. The first is

$$\frac{F^{(2)}(x)}{F(x)} = -\lambda^2 \implies F^{(2)}(x) = -\lambda^2 F(x),$$

which has the general solution:

$$F(x) = A\sin(\lambda x) + B\cos(\lambda x),$$

where A, B are constants. The second differential equation we obtain is

$$\frac{G^{(1)}(t)}{G(t)} = -\lambda^2 \implies G^{(1)}(t) = -\lambda^2 G(t),$$

which has the general solution:

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

where C is a constant. We can determine the coefficients B and λ by using the boundary conditions. At $x = 0$, we have

$$F(0) = A\sin(0) + B\cos(0) = A \cdot 0 + B \cdot 1 = B = 0.$$

Knowing that $B = 0$, we can determine λ by using the boundary condition at $x = L$:

$$F(L) = A\sin(\lambda L) = 0,$$

which is true when $\lambda = m\pi/L$ (we assume $A \neq 0$).

Our solution is then

$$u(x,t) = AC\sin\left(\frac{m\pi}{L}x\right)e^{-(m\pi/L)^2 t},$$

where AC is a constant, but this is only for one particular m value. There are infinitely many solutions because there are infinitely many m values. Each m value also has a corresponding coefficient A_m. The general solution is a superposition of such particular solutions.The general form of the solution is therefore

$$u(x,t) = \sum_{m=1}^{\infty} A_m \sin\left(\frac{m\pi}{L}x\right)e^{-(m\pi/L)^2 t},$$

with initial conditions for $0 < x < L$:

$$u(x,0) = g(x) = \sum_{m=1}^{\infty} A_m \sin\left(\frac{m\pi}{L}x\right).$$

The coefficients A_m is the Fourier coefficients of $g(x)$, and it is therefore determined by the Fourier transformation:

$$A_m = \frac{2}{L}\int_0^L g(x)\sin\left(\frac{m\pi}{L}x\right)dx.$$

Because of this, the analytical solution will vary depending on what kind of initial conditions we are using, determined by $g(x)$.

Now we look at the **two dimensional** case (Hjorth-Jensen, 2015, pp. 324-325)[2]. The equation is given by (9), which is

$$\frac{\partial^2}{\partial x^2}u(x,y,t) + \frac{\partial^2}{\partial y^2}u(x,y,t) = \frac{\partial}{\partial t}u(x,y,t).$$

Assuming that $x, y \in [0, L]$. We use boundary conditions:

$$\begin{aligned} u(0,y,t) &= 0,\\ u(x,0,t) &= 0,\\ u(L,y,t) &= 0,\\ u(x,L,t) &= 0, \end{aligned}$$

and initial conditions for $0 < x, y < L$ as $u(x, y, 0) = 1$.

We assume that variables x, y, t and be separated such that

$$u(x,y,t) = F(x,y)G(t).$$

Inserting this into the diffusion equation we get

$$GF^{xx} + GF^{yy} = FG^t,$$

where F^{xx} and F^{yy} are the double derivatives of F with respect to x and y, respectively. G^t is the first derivative of G with respect to t. We rearrange the terms to end up with

$$\frac{G^t}{G} = \frac{F^{xx} + F^{yy}}{F},$$

where both sides has to be equal to a constant, because of the separable variables. We define this constant to be $lambda^2$. The equation above yields two differential equations:

$$\begin{aligned} F^{xx} + F^{yy} &= -\lambda^2 F,\\ G^t &= -\lambda^2 G. \end{aligned}$$

The second equation has a general solution on the form

$$G(t) = Re^{-\lambda^2 t},$$

where R is a constant. For the first equation, we assume that x and y can be separated such that

$$F(x,y) = H(x)Q(y).$$

The first equation then becomes:

$$QH^{xx} + HQ^{yy} = -\lambda^2 HQ,$$

which can be written as

$$\frac{H^{xx}}{H} + \frac{Q^{yy}}{Q} = -\lambda^2.$$

Then we define $\rho^2 = \lambda^2 - k^2$. This yields two new equations:

$$\begin{aligned} H(x) &= A\cos(kx) + B\sin(kx),\\ Q(y) &= C\cos(\rho y) + D\sin(\rho y). \end{aligned}$$

Where A, B, C, D are constants. Using the boundary conditions, we can determine some of these constants:

$$\begin{aligned} H(0) &= A = 0,\\ H(L) &= B\sin(kL) = 0,\\ Q(0) &= C = 0,\\ Q(L) &= D\sin(\rho L) = 0. \end{aligned}$$

From the first and third equation we have that $A = C = 0$. The second and fourth equations yields $k = m\pi/L$ and $\rho = n\pi/L$, such that

$$\lambda^2 = \rho^2 + k^2 = \left(\frac{\pi}{L}\right)^2 (m^2 + n^2).$$

Putting everything together, we see that

$$F(x,y) = AD\sin\left(\frac{m\pi x}{L}\right)\sin\left(\frac{n\pi y}{L}\right),$$

where AD is just a constant. To obtain the full solution, we must include the time dependency $G(t)$. Since

the general solution is a superposition of particular solutions, we combine the constants into one A_{mn}. The general solution is then

$$u = \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} A_{mn} \sin\left(\frac{m\pi x}{L}\right) \sin\left(\frac{n\pi y}{L}\right) e^{-\pi^2(m^2+n^2)t/L^2}, \tag{B1}$$

where $u = u(x, y, t)$ and initial conditions

$$u(x, y, 0) = f(x, y),$$

given as

$$f(x, y) = \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} A_{mn} \sin\left(\frac{m\pi x}{L}\right) \sin\left(\frac{n\pi y}{L}\right).$$

Therefore, the coefficients A_{mn} are given through the Fourier transformation:

$$A_{mn} = \frac{4}{L^2}\int_0^L \int_0^L f(x, y) \sin\left(\frac{m\pi x}{L}\right) \sin\left(\frac{n\pi y}{L}\right) dxdy. \tag{B2}$$

These coefficients vary depending on the initial conditions $f(x, y)$.

Appendix C: Crank-Nicolson scheme

We can combine the implicit and explicit methods in a general approach (Hjorth-Jensen, 2015, pp. 310-313)[2] to derive the Crank-Nicolson scheme. Introducing the parameter θ, we can define the equation

$$\frac{\theta}{\Delta x^2}(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) + \frac{1-\theta}{\Delta x^2}(u_{i-1,j-1} - 2u_{i,j-1} + u_{i+1,j-1}) = \frac{1}{\Delta t}(u_{i,j} - u_{i,j-1}). \tag{C1}$$

Here θ is a parameter that specifies how much of the combination of the two methods favour one over the other. We can see this by investigating the limits: If $\theta = 0$, we obtain from equation (C1) the explicit forward method. If $\theta = 1$, we get the implicit backward method. Using $\theta = 1/2$, we obtain the Crank-Nicolson scheme.