UiO : Department of Physics
University of Oslo

Project 5: The diffusion equation

FYS3150 - Computational physics

Kristine Baluka Hein
Anders Johansson

Contents

Abstract

In this report we derive, analyse and implement three different numerical methods for solving the diffusion equation in one dimension. The three presented methods are the Forward Euler, Backward Euler and Crank-Nicolson schemes. As is shown analytically through von Neumann stability analysis and confirmed numerically, the Forward Euler scheme sets a criterion for the ratio of the spatial and temporal step lengths, while the other schemes always give stable results. The truncation errors are also analytically derived and numerically verified by comparison with the analytical solution, with the Crank-Nicolson scheme giving the best results.

We also derive the basic physical theory of diffusion, and develop a script for animating the process for easy visualisation. Finally, a simple, explicit method is derived and implemented for simulating two-dimensional diffusion, as well as an analytical solution for comparison.

1 Introduction

Partial differential equations are important in many fields of science. These equations can describe the behaviour of many natural phenomenas such as waves of sound or sea, the flow of a liquid, growth of a population or diffusion, which is the phenomenon studied in this report. Diffusion is an important process which we find everywhere around us, from water softening spaghetti by diffusing into it to the random motion of viruses on the hunt for fresh cells to destroy.

Partial differential equations are often solved numerically since an analytical solution is difficult, or even impossible to find. An example of the latter is the Navier-Stokes equation to model the flow of a viscous fluid when not certain assumptions to simplify the equations have been made.

In this report we will focus on how we can numerically solve the diffusion equation on a dimensionless form, which is written as

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

After deriving Fick's laws of diffusion, we derive and analyse methods for solving the equation in both one and two dimensions. Only the explicit Forward Euler scheme is used in two dimensions, while the one-dimensional equation is also solved with the implicit Backward Euler and Crank-Nicolson schemes. In one dimension, von Neumann stability analysis is used to find criteria which guarantee the stability of the methods. Analytical solutions are found in both cases for verification of the methods and error analysis.

2 Physical theory of diffusion

Diffusion is simply the net movement of particles as a result of a random walk, and can be modelled as each atom constantly making a jump in a random direction. It may seem a little counterintuitive that random motion can lead to a net flow, but consider that if there are 10 particles to the left, of which half move to the right, and 100 particles on the right, of which half move to the left, the net result is 45 particles moving to the left. The opposite of diffusion is drift, where the net motion is a result of an external driving force, for example an electric field.

The two most important equations in diffusion are Fick's laws, which, in one dimension, state that

$$J = -D\frac{\partial C}{\partial x} \qquad \frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial x^2}$$

where D is the diffusion coefficient, J the flux, i.e. the number of particles moving per area, and C the concentration, i.e. the number of particles per volume. Fick's second law, which is also simply called the diffusion equation, is the partial differential equation which is solved numerically on a dimensionless form in this project.

2.1 Derivation of Fick's first law in one dimension

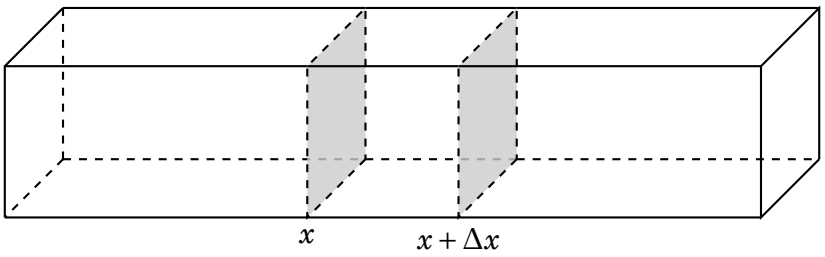

Figure 1: Sketch of the infinitesimal volume element used in the derivation of Fick's laws in one dimension. Drawn in TikZ.

As stated above, diffusion is the process of random walk. In one dimension, this means that each particle will have a 50 % probability of moving one step to the left, and a 50 % probability of moving one step to the right. As such, the net movement of particles to the right between x and $x + \Delta x$ equals half the particles at point x minus half the particles at point $x + \Delta x$. Since the flux is the movement of particles per cross-section area A (marked with grey in the figure) per time interval Δt, this can be formulated mathematically as

$$J = -\frac{N(x + \Delta x, t) - N(x, t)}{2A\Delta t}$$

Multiplying by $(\Delta x)^2$ in both the numerator and the denominator, and factoring out the Δt, we get

$$J = -\frac{(\Delta x)^2}{2\Delta t}\left(\frac{N(x + \Delta x, t) - N(x, t)}{A\Delta x \cdot \Delta x}\right)$$

Since $A\Delta x$ is a small volume element, $C = N/(A\Delta x)$:

$$J = -\frac{(\Delta x)^2}{2\Delta t}\left(\frac{C(x+\Delta x,t)-C(x,t)}{\Delta x}\right)$$

Defining $D = (\Delta x)^2/2\Delta t$ and letting $\Delta x \to 0$ gives Fick's first law:

$$J = -D\frac{\partial C}{\partial x}$$

2.2 Derivation of Fick's second law in one dimension

The derivation of Fick's second law is based on finding two different expressions for the change in the number of particles between x and $x+\Delta x$, setting them equal to each other and inserting Fick's first law into the result.

To find the change in number of particles between the two grey surfaces in figure 1 on the preceding page over a time interval Δt, remember that the flux is defined as the number of particles passing per cross-section A per time interval Δt. Therefore, the number of particles moving out of the volume between x and $x+\Delta x$ in a time interval Δt is $J(x+\Delta x,t)A\Delta t$, while the number of particles moving into the volume is $J(x,t)A\Delta t$. Thus, the net change in the number of particles between x and Δx can be written as

$$(J(x,t)-J(x+\Delta x,t))A\Delta t = -(J(x+\Delta x,t)-J(x,t))A\Delta t$$

On the other hand, this change in the number of particles must also equal the change in concentration, $C(x,t+\Delta t)-C(x,t)$, multiplied with the volume, $A\Delta x$. Since these two methods of calculating the change in the number of particles must give the same result, the following equation must hold:

$$-(J(x+\Delta x,t)-J(x,t))A\Delta t = (C(x,t+\Delta t)-C(x,t))A\Delta x$$
$$\frac{J(x+\Delta x,t)-J(x,t)}{\Delta x} = -\frac{C(x,t+\Delta t)-C(x,t)}{\Delta t}$$

Letting $\Delta x \to 0$ and $\Delta t \to 0$, we get the so-called continuity equation:

$$\frac{\partial J}{\partial x} = -\frac{\partial C}{\partial t}$$

Inserting Fick's first law for J on the left hand side, Fick's second law appears:

$$\frac{\partial}{\partial x}\left(-D\frac{\partial C}{\partial x}\right) = -\frac{\partial C}{\partial t}$$
$$D\frac{\partial^2 C}{\partial x^2} = \frac{\partial C}{\partial t}$$

2.3 Steady state

When $t \to \infty$, the system will reach a steady state where the concentration at each point does not change with time. A different formulation is that for each volume element, the number of particles entering must be the same as the number of particles leaving, which means that the flux must be the same at all points. Using Fick's first law, this can be formulated as

$$\text{constant} = J = -D\frac{\partial C}{\partial x}$$

which means that the concentration must be a linear function of x. This is a test which both the analytical and the numerical solutions must pass.

2.4 Initial and boundary conditions

We will study a situation with a constant source at $x = L$ and a drain at $x = 0$, with the system initially containing 0 particles everywhere but at the source. In the dimensionless form, this can be formulated as

$$\begin{aligned} u(x,0) &= 0 \qquad x \in [0,1) \\ u(0,t) &= 0 \qquad t \in \mathbb{R} \\ u(L,t) &= 1 \qquad t \in \mathbb{R} \end{aligned}$$

3 Mathematical theory in one dimension

Before we can begin our analysis on how it is possible to numerically solve the diffusion equation, we will first take a look at the analytical solution. This analytical solution can be used in von Neumann analysis to find requirements for the numerical parameters to ensure that a reasonable solution is found.

3.1 Analytical solution of the diffusion equation

Consider the diffusion equation with the given boundaries:

$$\begin{aligned} \frac{\partial^2 u(x,t)}{\partial x^2} &= \frac{\partial u(x,t)}{\partial t} \quad && t>0,\ x\in[0,L] \\ u(x,0) &= g(x) && 0<x<L \\ u(0,t) &= a && a\in\mathbb{R}, t>0 \\ u(L,t) &= b && b\in\mathbb{R}, t>0 \end{aligned} \tag{2}$$

As will become clear later, life is much easier when a and b are both zero. As such, we introduce a new function

$$v(x,t) = u(x,t) + w(x,t)$$

where u is a solution of equation (2) and the following also holds:

$$\begin{aligned} \frac{\partial^2 v(x,t)}{\partial x^2} &= \frac{\partial v(x,t)}{\partial t} \quad && t>0,\ x\in[0,L] \\ v(x,0) &= f(x) && 0<x<L \\ v(0,t) &= 0 && t>0 \\ v(L,t) &= 0 && t>0 \end{aligned} \tag{3}$$

When introducing $v(x,t)$, a new problem arises: What does $w(x,t)$ actually look like? Inserting the definition of v into equation (3) gives

$$\begin{aligned} \frac{\partial^2 v(x,t)}{\partial x^2} &= \frac{\partial v(x,t)}{\partial t} \\ \frac{\partial^2 u(x,t)}{\partial x^2} + \frac{\partial^2 w(x,t)}{\partial x^2} &= \frac{\partial u(x,t)}{\partial t} + \frac{\partial w(x,t)}{\partial t} \end{aligned}$$

The assumption that u is a solution to the original problem, which means that

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

As a result, equation (3) will hold if

$$\frac{\partial^2 w(x,t)}{\partial x^2} = 0 = \frac{\partial w(x,t)}{\partial t} \tag{4}$$

These two equalities are separately fulfilled if

$$\begin{aligned}\frac{\partial^2 w(x,t)}{\partial x^2} &= 0 \\ \frac{\partial w(x,t)}{\partial x} &= A \\ w(x,t) &= Ax + B + h_1(t) \end{aligned} \tag{5}$$

and

$$\begin{aligned}\frac{\partial w(x,t)}{\partial t} &= 0 \\ w(x,t) &= C + h_2(x) \end{aligned} \tag{6}$$

Combining these two requirements for w, we see that it must be on the form

$$w(x,t) = w(x) = Ax + D$$

because there is no dependence of t in equation (6).

We have now found a possible form for w. However, the values of A and D still need to be determined. This can be done by looking at the boundary conditions for v at $x = 0$ and $x = L$ and using the assumption that u fulfills the boundary conditions of equation (2), i.e. $u(0,t) = a$:

$$0 = v(0,t) = u(0,t) + w(x = 0) = a + D$$

which implies $D = -a$. Similarly, the boundary at $x = L$ gives

$$0 = v(L,t) = u(L,t) + w(x = L) = b + AL - a \implies A = \frac{a-b}{L}$$

In conclusion, we have

$$w(x) = \frac{a-b}{L}x - a$$

As the expression of w has been found, we have also found $f(x)$ in equation (3) on the preceding page:

$$\begin{aligned} v(x,0) &= u(x,0) + w(x) \\ &= g(x) + w(x) \\ &= f(x) \end{aligned}$$

Having set up the problem properly, we are now ready to solve for $v(x,t)$. To do so, we make the assumption that v is separable, i.e. it can be written as

$$v(x,t) = F(x)G(t)$$

which means that

$$\begin{aligned} \frac{\partial^2 v(x,t)}{\partial x^2} &= \frac{\partial v(x,t)}{\partial t} \\ F''(x)G(t) &= F(x)G'(t) \end{aligned}$$

$$\frac{F''(x)}{F(x)} = \frac{G'(t)}{G(t)}$$

As the two sides of the last equation depend on different variables which can be varied independently, both sides must equal some constant $-\lambda$:

$$\frac{F''(x)}{F(x)} = \frac{G'(t)}{G(t)} = -\lambda \tag{7}$$

Solving for F we get

$$F'' = -\lambda F$$

with a general solution

$$F(x) = c_1 \cos\left(x\sqrt{\lambda}\right) + c_2 \sin\left(x\sqrt{\lambda}\right)$$

To find the constants c_1 and c_2, we will use the boundaries for $v(x,t)$ at $x = 0$ and $x = L$. The advantage of having the boundary conditions equal to zero now becomes apparent, as this implies

$$0 = v(0,t) \ = F(0)G(t) \implies F(0) = 0$$

and

$$0 = v(L,t) = F(L)G(t) \implies F(L) = 0$$

Since $\sin(0) = 0 \neq \cos(0)$, we must have that

$$0 = c_1$$

and

$$0 = c_2 \sin\left(L\sqrt{\lambda}\right)$$

Of course, c_2 could also be set to 0, but this is the trivial solution which is of no interest. Therefore, we must have

$$0 = \sin\left(L\sqrt{\lambda}\right)$$

Since sine is zero for every integer multiple of π, we have that

$$L\sqrt{\lambda} = k\pi \quad k \in \mathbb{N}$$
$$\lambda = \left(\frac{k\pi}{L}\right)^2$$

In conclusion, the following is a solution for all $k \in \mathbb{N}$:

$$F_k(x) = c_k \sin\left(x\frac{k\pi}{L}\right)$$

Similarly, equation (7) implies that

$$G_k'(t) = -\lambda_k G_k(t) \implies G_k(t) = C_k \mathrm{e}^{-\left(\frac{k\pi}{L}\right)^2 t}$$

For simplicity, C_k can be absorbed into c_k.

We are now close to an expression for v:

$$v_k(x,t) = c_k \mathrm{e}^{-\left(\frac{k\pi}{L}\right)^2 t} \sin\left(x\frac{k\pi}{L}\right)$$

What remains is to find c_k. The final, unused piece of information is the initial condition, i.e. $v(x,0)$. Inserting $t=0$ in the equation above makes the exponential 1, i.e.

$$f(x) = c_k \sin\left(x\frac{k\pi}{L}\right)$$

Multiplying the equation above by $\sin\left(x\frac{k\pi}{L}\right)$, integrating both sides and using the orthogonality of sine (note that this sine is L-periodic), gives us

$$\int_0^L f(x)\sin\left(x\frac{k\pi}{L}\right)\mathrm{d}x = c_k\frac{L}{2}$$
$$\frac{2}{L}\int_0^L f(x)\sin\left(x\frac{k\pi}{L}\right)\mathrm{d}x = c_k$$

However, we are not looking for one particular solution of equation (3), but for all solutions. From the principle of superposition, every particular solution contributes to another solution. Specifically, this means that

$$v(x,t) = \sum_{k=1}^{\infty} v_k(x,t)$$

Remember now the original ansatz

$$v(x,t) = u(x,t) + w(x,t)$$

As v and w have now been determined, u can be found through a reorganisation:

$$\begin{aligned} u(x,t) &= v(x,t) - w(x,t) \\ &= \sum_{k=1}^{\infty} v_k(x,t) - w(x) \\ &= \sum_{k=1}^{\infty} c_k \mathrm{e}^{-\left(\frac{k\pi}{L}\right)^2 t} \sin\left(x\frac{k\pi}{L}\right) - \left(\frac{a-b}{L}x - a\right) \end{aligned}$$

with

$$c_k = \frac{2}{L}\int_0^L f(x)\sin\left(x\frac{k\pi}{L}\right)\mathrm{d}x$$

For our case, that is equation (1) on page 3, we have that $a=0$, $b=1$, $g(x)=0$ and $L=1$. Inserting this into the solution of equation (3) on page 7 gives us

$$u(x,t) = \sum_{k=1}^{\infty} c_k \mathrm{e}^{-(k\pi)^2 t} \sin(xk\pi) + x \tag{8}$$

with

$$c_k = 2\int_0^1 (0-x)\cdot\sin(xk\pi)\,\mathrm{d}x = -\frac{2}{k\pi}\left[\frac{\sin(k\pi x)}{k\pi} - x\cos(k\pi x)\right]_0^1 = \frac{2(-1)^{k+2}}{k\pi} = \frac{2(-1)^k}{k\pi}$$

3.2 Discretisation

For the diffusion equation to be numerically solvable, it must be discretised.

This is done in the standard way: Discretise x as $x_0, x_1, \ldots, x_n$ with $\Delta x = (x_n - x_0)/n$, and t as $t_0, t_1, \ldots, t_m$ with $\Delta t = (t_m - t_0)/m$. The function itself is also discretised, with the notation $u_{i,j} \approx u(x_i, t_j)$.

We will now take a look into how we can numerically approximate the given problem. As there exist many different approaches, we have chosen to look at three different methods, namely Forward Euler, Backward Euler and Crank-Nicolson. The difference between Forward Euler and Backward Euler is how the time derivative is approximated, as the Crank-Nicolson method is a combination of both.

3.3 Forward Euler

Forward Euler is the simplest and most intuitive algorithm implemented in this report. As we will see later, this simplicity in both idea and implementation comes at a cost, as the Forward Euler scheme is the only algorithm which sets requirements for the values of Δx and Δt to even give a reasonable solution.

3.3.1 Derivation and error analysis

The Forward Euler scheme is an explicit scheme based on Taylor polynomials. To find an approximation to the time derivative of u at point (x_i, t_j), a first order Taylor polynomial around x_i, t_j is used to calculate $u(x_i, t_{j+1})$:

$$u_{i,j+1} = u_{i,j} + h\frac{\partial u_{i,j}}{\partial t} + \frac{1}{2}(\Delta t)^2 \frac{\partial^2 u(x_i, \tilde{t})}{\partial t^2} \implies \frac{\partial u_{i,j}}{\partial t} = \frac{u_{i,j+1} - u_{i,j}}{\Delta t} + \frac{1}{2}\Delta t \frac{\partial^2 u(x_i, \tilde{t})}{\partial t^2} \tag{9}$$

where $\tilde{t} \in (t_j, t_{j+1})$ and the last term is the truncation error, which is proportional to Δt.

Similarly, the three point approximation to the second derivative (derived in [1]) with its error is used to approximate the second derivative of u at point (x_i, t_j) with respect to position:

$$\frac{\partial^2 u_{i,j}}{\partial x^2} = \frac{u_{i+1,j} + u_{i-1,j} - 2u_{i,j}}{(\Delta x)^2} - \frac{1}{12}(\Delta x)^2 \frac{\partial^4 u(\tilde{x}, t_j)}{\partial x^4} \tag{10}$$

where the last term is the truncation error, which for this approximation is proportional to $(\Delta x)^2$. As shown in [1], the error term actually consists of two terms, one with $\tilde{x}_1 \in (x_{i-1}, x_i)$ and one with $\tilde{x}_2 \in (x_i, x_{i+1})$, but this can be approximated with $\tilde{x} \in (x_{i-1}, x_{i+1})$.

Inserting these two expressions into the diffusion equation gives

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} + \frac{1}{2}\Delta t \frac{\partial^2 u(x_i, \tilde{t})}{\partial t^2} = \frac{u_{i+1,j} + u_{i-1,j} - 2u_{i,j}}{(\Delta x)^2} - \frac{1}{12}(\Delta x)^2 \frac{\partial^4 u(\tilde{x}, t_j)}{\partial x^4}$$

The goal is to find the value of u at the next time step, i.e. $u_{i,j+1}$. Multiplying by Δt on both sides of the equation and moving one term to the right, we get

$$\begin{aligned} u_{i,j+1} =& u_{i,j} + \frac{\Delta t}{(\Delta x)^2}\left(u_{i+1,j} + u_{i-1,j} - 2u_{i,j}\right) \\ &+ \frac{1}{2}(\Delta t)^2 \frac{\partial^2 u(x_i,\tilde{t})}{\partial t^2} - \Delta t \cdot \frac{1}{12}(\Delta x)^2 \frac{\partial^4 u(\tilde{x},t_j)}{\partial x^4} \end{aligned}$$

The quantity $\Delta t/(\Delta x)^2$ can be defined as α, which, with a slight reorganisation yields the final expression

$$\begin{aligned} u_{i,j+1} =& (1-2\alpha)u_{i,j} + \alpha\left(u_{i+1,j} + u_{i-1,j}\right) \\ &+ \frac{1}{2}(\Delta t)^2 \frac{\partial^2 u(x_i,\tilde{t})}{\partial t^2} - \Delta t \cdot \frac{1}{12}(\Delta x)^2 \frac{\partial^4 u(\tilde{x},t_j)}{\partial x^4} \end{aligned}$$

The two error terms are proportional to $(\Delta t)^2$ and $\Delta t \cdot (\Delta x)^2$. As per usual, the global error is one order lower, as the error is accumulated. This gives one error term proportional to Δt and one proportional to $(\Delta x)^2$. The order of (Δx) is not reduced, as this error is not accumulated. If we truncate the errors in $u_{i,j}$, we get the numerical approximation of $u(x_i,t_{j+1})$ to be

$$u_{i,j+1} = (1-2\alpha)u_{i,j} + \alpha\left(u_{i+1,j} + u_{i-1,j}\right) \tag{11}$$

3.3.2 The idea behind von Neumann stability analysis

As we now have set up the algorithm using Forward Euler to approximate the derivatives, we might ask ourselves whether this method gives us stable results. This is an important question, since it may happen that the numerical errors propagate in such a manner that it will corrupt the final results. To see whether this is the case, we will use the method of von Neumann to study the stability of the Forward Euler scheme, as well as the Backward Euler and Crank-Nicolson schemes.

The problem of approximating $u(x,t)$ is to sufficiently approximate the infinite series given in equation (8) on page 10 and the exact value of the c_ks. Since x does not include infinity and therefore can be computed directly, we can assume that x will not contribute to the propagation of error. Therefore, we must look at the stability when calculating $\sum_{k=1}^{\infty} c_k e^{-(k\pi)^2 t}\sin(xk\pi)$, i.e the solution of $v(x,t)$ to make sure it will not blow up the solution of $u(x,t)$.

If we proceed in the similar manner as for finding the analytical solution and making the ansatz that

$$v_{k_{i,j}} = F_k(x_i)G_k(t_j) \tag{12}$$

and introducing D_x^2 and D_t as the numerical approximations of the spatial and time derivatives respectively, we get that

$$G(t_j)\left(D_x^2 F_k(x)\right)_i = F(x_i)\left(D_t G(t)\right)_j$$

where $\left(D_x^2 F_k(x)\right)_i$ and $\left(D_t G(t)\right)_j$ means that we are taking the derivative around the i-th and j-th step respectively. From what we found above, we have that

$$\frac{\left(D_x^2 F_k(x)\right)_i}{F(x_i)} = \frac{\left(D_t G_k(t)\right)_j}{G_k(t_j)} = -\mu_{k_{i,j}} \tag{13}$$

The second derivative of x is approximated using the second order central difference equation, i.e equation (10) on page 11. From the analytical solution, we can assume that $F(x_i) = \sin(k\pi x_i)$. This gives

$$\begin{aligned}\left(D_x^2 F_k(x)\right)_i &= -\mu_{k_{i,j}} F(x_i) \\ \frac{F(x_{i-1}) - 2F(x_i) + F(x_{i+1})}{\Delta t^2} &= -\mu_{k_{i,j}} F(x_i) \\ \sin(k\pi x_{i-1}) - 2\sin(k\pi x_i) + \sin(k\pi x_{i+1}) &= -\Delta t^2 \mu_{k_{i,j}} \sin(k\pi x_i)\end{aligned}$$

Using the trigonometric identities

$$\sin(k\pi x_{i-1}) + \sin(k\pi x_{i+1}) = 2\cos(k\pi\Delta t)\sin(k\pi x_i)$$

we get

$$\sin(k\pi x_i)(2\cos(k\pi\Delta t) - 2) = -\Delta t^2 \mu_{k_{i,j}} \sin(k\pi x_i)$$

The factor $\sin(k\pi x_i)$ can never be zero, as the derivative is taken at points in $(0,1)$ since the boundary values are known. The eigenvalue $\mu_{k_{i,j}}$ of the approximation of the second derivative with respect to x, D_x^2, is therefore

$$\frac{-2(\cos(k\pi\Delta t) - 1)}{\Delta t^2} = \mu_{k_{i,j}} \tag{14}$$

Furthermore, we have the identity

$$\begin{aligned}\cos(x) &= \cos^2\left(\frac{x}{2}\right) - \sin^2\left(\frac{x}{2}\right) \\ &= 1 - 2\sin^2\left(\frac{x}{2}\right) \\ 1 - \cos(x) &= 2\sin^2\left(\frac{x}{2}\right)\end{aligned}$$

and by inserting this into equation (14), we get the following eigenvalue

$$\frac{4\sin^2\left(\frac{k\pi\Delta t}{2}\right)}{\Delta t^2} = \mu_{k_{i,j}} = \mu_k \tag{15}$$

Now we have proven that assuming $F_k(x_i) = \sin(k\pi x_i)$ is not a such bad idea, as the guess is reasonable compared to the analytical solution and no other contradictions have arised.

We see from the analytical solution that the function of time, $G_k(t) = \mathrm{e}^{-\lambda_k t}$ is decreasing for increasing values of t, meaning that the discrete functions of time should behave in equivalent manner.

Therefore the computed function of time, $G_k(t_j)$, must hold the following constraints

$$G_k(t_0) = 1 \quad \text{and} \quad \left|\frac{G_k(t_{j+1})}{G_k(t_j)}\right| \leq 1 \ \text{ for } j = 0, \ldots, m-1$$

We include that the fraction could be equal to one as it may happen that we could be very close to the likely state, that is when $t \to \infty$, giving us $G_k(t_{j+1}) \approx G_k(t_j)$.

When the ratio above holds, we have that the solution does not grow for every time step, ensuring that the computed solution is stable for every time step.

Since the expression of $G_k(t_j)$ will depend on the choice of numerical scheme, we must do sufficient restrictions for every scheme such that $G_k(t_j)$ does not increase for an increasing time step.

3.3.3 Stability analysis of Forward Euler

To approximate the time derivative of Forward Euler, we get the ratios in equation (13) to be

$$\begin{aligned}
(D_t G_k(t))_j &= -G_k(t_j)\mu_k \\
\frac{G_k(t_{j+1}) - G_k(t_j)}{\Delta t} &= -G_k(t_j)\mu_k \\
G_k(t_{j+1}) &= -\Delta t G_k(t_j)\mu_k + G_k(t_j) \\
\frac{G_k(t_{j+1})}{G_k(t_j)} &= 1 - \Delta t \mu_k
\end{aligned}$$

As discussed in section 3.3.2, we must have that

$$\begin{aligned}
\left|\frac{G_k(t_{j+1})}{G_k(t_j)}\right| &\leq 1 \\
\left|1 - \Delta t \mu_k\right| &\leq 1 \\
\left|\Delta t \mu_k\right| &\leq 2 \\
\left|\frac{4\Delta t \sin^2\left(\frac{k\pi\Delta x}{2}\right)}{\Delta x^2}\right| &\leq 2 \\
\left|\frac{\Delta t \sin^2\left(\frac{k\pi\Delta x}{2}\right)}{\Delta x^2}\right| &\leq \frac{1}{2}
\end{aligned}$$

Since $\sin^2\left(\frac{k\pi\Delta x}{2}\right) \leq 1$ for any k and Δx, this simplifies to

$$\frac{\Delta t}{\Delta x^2} \leq \frac{1}{2} \tag{16}$$

which is the final criterion for stable results from the Forward Euler scheme.

3.4 Backward Euler

The idea of Backward Euler is similar to that of Forward Euler, as the only difference lies in how we approximate the time derivative of u. The result, however, is the freedom to choose Δt and Δx as we like and be guaranteed a resonable result.

3.4.1 Derivation and error analysis

The time derivative of u at (x_i, t_j) is once again found through a first order Taylor polynomial around (x_i, t_j), but the Taylor polynomial is now used to approximate $u_{i,j-1}$, rather than $u_{i,j+1}$ as in the derivation of the Forward Euler scheme:

$$u_{i,j-1} = u_{i,j} - \Delta t \frac{\partial u_{i,j}}{\partial t} + \frac{1}{2}(\Delta t)^2 \frac{\partial^2 u(x_i, \tilde{t})}{\partial t^2} \implies \frac{\partial u_{i,j}}{\partial t} = \frac{u_{i,j} - u_{i,j-1}}{\Delta t} + \frac{1}{2}\Delta t \frac{\partial^2 u(x_i, \tilde{t})}{\partial t^2}$$

where $\tilde{t}$ and the second term are the same as defined in section 3.3.1 on page 11. The same three-point approximation as in equation (10) on page 11 can be reused to approximate the second derivative with respect to x.

With these approximations, the diffusion equation can be written as

$$\begin{aligned}
\frac{u_{i,j} - u_{i,j-1}}{\Delta t} + \frac{1}{2}\Delta t \frac{\partial^2 u(x_j, \tilde{t})}{\partial t^2} &= \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + \frac{1}{12}(\Delta x)^2 \frac{\partial^4 u(\tilde{x}, t_j)}{\partial x^4} \\
u_{i,j} - u_{i,j-1} &= \alpha(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}) \\
&\quad + \Delta t \cdot \frac{1}{12}(\Delta x)^2 \frac{\partial^4 u(\tilde{x}, t_j)}{\partial x^4} - \frac{1}{2}(\Delta t)^2 \frac{\partial^2 u(x_j, \tilde{t})}{\partial t^2} \\
-\alpha u_{i-1,j} + (1+2\alpha)u_{i,j} - \alpha u_{i+1,j} &= u_{i,j-1} + \Delta t \cdot \frac{1}{12}(\Delta x)^2 \frac{\partial^4 u(\tilde{x}, t_j)}{\partial x^4} - \frac{1}{2}(\Delta t)^2 \frac{\partial^2 u(x_j, \tilde{t})}{\partial t^2}
\end{aligned}$$

In this equation, only the u on the right hand side is known, while the u's on the left side are the ones we are interested in. As it is not possible to find an explicit expression for the quantities of interest, this leads to an implicit scheme. Observe that the error is the same as for the Forward Euler scheme.

To solve the equation above, note that it must hold for all $i \in [1, n-1] \cap \mathbb{N}$. When ignoring the error terms, we get the following set of equations:

$$\begin{array}{ccccccc}
-\alpha\overbrace{u_{0,j}}^{0} & + & (1+2\alpha)u_{1,j} & - & \alpha u_{2,j} & = & u_{1,j-1} \\
-\alpha u_{1,j} & + & (1+2\alpha)u_{2,j} & - & \alpha u_{3,j} & = & u_{2,j-1} \\
-\alpha u_{2,j} & + & (1+2\alpha)u_{3,j} & - & \alpha u_{4,j} & = & u_{3,j-1} \\
\vdots & & \vdots & & \vdots & & \vdots \\
-\alpha u_{n-3,j} & + & (1+2\alpha)u_{n-2,j} & - & \alpha u_{n-1,j} & = & u_{n-2,j-1} \\
-\alpha u_{n-2,j} & + & (1+2\alpha)u_{n-1,j} & - & \alpha\underbrace{u_{n,j}}_{1} & = & u_{n-1,j-1}
\end{array}$$

This can then be written on a matrix form:

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

This is a simple, linear system with a tridiagonal matrix, for which an efficient solving algorithm was developed and implemented in project 1 [1].

3.4.2 Stability analysis

For this scheme we have that equation (13) on page 12 for $G_k(t_j)$ becomes

$$\begin{aligned}
(D_t G_k(t))_j &= -G_k(t_j)\mu_k \\
\frac{G_k(t_j) - G_k(t_{j-1})}{\Delta t} &= -G_k(t_j)\mu_k \\
-G_k(t_{j-1}) &= -\Delta t G_k(t_j)\mu_k - G_k(t_j) \\
\frac{G_k(t_j)}{G_k(t_{j-1})} &= \frac{1}{1+\Delta t\mu_k}
\end{aligned}$$

Since $1+\Delta t\mu_k \geq 1$ for all $k, \mu_k, \Delta t$, the fraction is less than one for any choice of $\mu_k, \Delta t$. This means that the Backward Euler scheme is stable for every choice of $\Delta x, \Delta t$, so there are no restrictions on the choice of Δx and Δt, as opposed to the case with Forward Euler.

3.5 Crank-Nicolson

The Crank-Nicolson scheme can be thought of as a combination of the Forward and Backward Euler schemes. We will see that like the Backward Euler scheme, Crank-Nicolson is stable for any choice of Δt and Δx.

3.5.1 Derivation and error analysis

The idea of the Crank-Nicolson algorithm is to use the derivatives at the times $t_i + \frac{1}{2}\Delta t$ rather than at t_i, as the asymmetric Newton quotient used in the derivation of the Forward Euler scheme will now be the symmetric Newton quotient, giving an error proportional to $(\Delta t)^2$ instead of Δt:

$$\frac{\partial u(x_i, t_j + \frac{1}{2}\Delta t)}{\partial t} = \frac{u_{i,j+1} - u_{i,j}}{\Delta t} + \frac{(\Delta t)^2}{12}\frac{\partial^3 u(x_i, \tilde{t})}{\partial t^3}$$

with $\tilde{t} \in (t_j, t_{j+1})$.

The three-point formula is again used to approximate the second derivative with respect to x, but as there is no known value of u at $t_j + \frac{1}{2}\Delta t$, the second derivative at this point is approximated by the mean of the second derivatives at t_{j+1} and t_j:

$$\begin{aligned}
\frac{\partial^2 u(x_i, t_j + \frac{1}{2}\Delta t)}{\partial x^2} =& \frac{1}{2}\left(\frac{u_{i+1,j+1} + u_{i-1,j+1} - 2u_{i,j+1})}{(\Delta x)^2} + \frac{u_{i+1,j} + u_{i-1,j} - 2u_{i,j}}{(\Delta x)^2}\right) \\
&+ \frac{(\Delta x)^2}{24}\left(\frac{\partial^4 u(\tilde{x}_1, t_{j+1})}{\partial x^4} + \frac{\partial^4 u(\tilde{x}_2, t_j)}{\partial x^4}\right)
\end{aligned}$$

with both $\tilde{x}_1$ and $\tilde{x}_2$ in (x_{i-1}, x_{i+1}).

Inserting these expressions into the diffusion equation gives

$$\begin{aligned}\frac{u_{i,j+1}-u_{i,j}}{\Delta t} =& \frac{1}{2}\left(\frac{u_{i+1,j+1}+u_{i-1,j+1}-2u_{i,j+1})}{(\Delta x)^2}+\frac{u_{i+1,j}+u_{i-1,j}-2u_{i,j}}{(\Delta x)^2}\right)\\ &+\frac{(\Delta x)^2}{24}\left(\frac{\partial^4 u(\tilde{x}_1,t_{j+1})}{\partial x^4}+\frac{\partial^4 u(\tilde{x}_2,t_j)}{\partial x^4}\right)+\frac{(\Delta t)^2}{12}\frac{\partial^3 u(x_i,\tilde{t})}{\partial t^3}\end{aligned}$$

We are interested in the solution at t_{j+1}. Multiplying by $2\Delta t$ on both sides and reusing $\alpha = \Delta t/(\Delta x)^2$ gives

$$\begin{aligned}-\alpha u_{i-1,j+1}+(2+2\alpha)u_{i,j+1}-\alpha u_{i+1,j+1} =& \alpha u_{i-1,j}+(2-2\alpha)u_{i,j}+\alpha u_{i+1,j}\\ &+\Delta t\cdot\frac{(\Delta x)^2}{12}\left(\frac{\partial^4 u(\tilde{x}_1,t_{j+1})}{\partial x^4}+\frac{\partial^4 u(\tilde{x}_2,t_j)}{\partial x^4}\right)+\frac{(\Delta t)^3}{6}\frac{\partial^3 u(x_i,\tilde{t})}{\partial t^3}\end{aligned}$$

Since the error is once again accumulated, the global error will have one term proportional to $(\Delta x)^2$ and one term proportional to $(\Delta t)^2$. The latter is one order better than both Euler schemes. With no possibility of solving for the next time step explicitly, this scheme is also implicit.

We have therefore the following approximation

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

Solving the above equation is done exactly the same way as for the Backward Euler scheme: As it must hold for all $i \in [1, n-1]\cap\mathbb{N}$, a linear set of equations arises, which, with $\beta = 2+2\alpha$ and $\beta' = 2-2\alpha$, can be written on a matrix form as

$$\begin{bmatrix}\beta & -\alpha & 0 & 0 & 0 & \dots & 0 & 0\\ -\alpha & \beta & -\alpha & 0 & 0 & \dots & 0 & 0\\ 0 & -\alpha & \beta & -\alpha & 0 & \dots & 0 & 0\\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots\\ 0 & 0 & 0 & \dots & -\alpha & \beta & -\alpha & 0\\ 0 & 0 & 0 & 0 & \dots & -\alpha & \beta & -\alpha\\ 0 & 0 & 0 & 0 & 0 & \dots & -\alpha & \beta\end{bmatrix}\begin{bmatrix}u_{1,j+1}\\ u_{2,j+1}\\ u_{3,j+1}\\ \vdots\\ u_{n-3,j+1}\\ u_{n-2,j+1}\\ u_{n-1,j+1}\end{bmatrix} = \begin{bmatrix}\beta' u_{1,j}+\alpha u_{2,j}\\ \alpha u_{n-1,j}+\beta' u_{2,j}+\alpha u_{2,j}\\ \alpha u_{n-2,j}+\beta' u_{3,j}+\alpha u_{3,j}\\ \vdots\\ \alpha u_{n-4,j}+\beta' u_{n-3,j}+\alpha u_{n-2,j}\\ \alpha u_{n-3,j}+\beta' u_{n-2,j}+\alpha u_{n-1,j}\\ \alpha u_{n-2,j}+\beta' u_{n-1,j}+2\alpha\end{bmatrix}$$

The matrix is once again tridiagonal, so the solver from project 1 can be reused.

3.5.2 Stability analysis

Since the Crank-Nicolson scheme is derived from the approximation of the second derivative with respect to x at $v_k(x_i,t_j)$ and $v_k(x_i,t_{j+1})$, we can use the ansatz in equation (12) on page 12 and rewrite the scheme as follows:

$$\frac{1}{2}\big(G_k(t_{j+1})\big(D_x^2F_k(x)\big)_i + G_k(t_j)\big(D_x^2F_k(x)\big)_i\big) = \frac{(F_k(x))_i}{\Delta t}\big(G_k(t_{j+1})-G_k(t_j)\big)$$

We found in section 3.3.2 that $D_x^2 F_k(x))_i = -\mu_k F_k(x_i)$, meaning that we can rewrite the equation above into

$$\frac{1}{2}\big(G_k(t_{j+1})\big(D_x^2 F_k(x)\big)_i + G_k(t_j)\big(D_x^2 F_k(x)\big)_i\big) = \frac{F_k(x_i)}{\Delta t}\big(G_k(t_{j+1}) - G_k(t_j)\big)$$

$$\frac{1}{2}\big(-\mu_k F_k(x_i) G_k(t_{j+1}) - \mu_k F_k(x_i) G_k(t_j)\big) = \frac{F_k(x_i)}{\Delta t}\big(G_k(t_{j+1}) - G_k(t_j)\big)$$

$$\frac{-\mu_k}{2}\big(G_k(t_{j+1}) + G_k(t_j)\big) = \frac{1}{\Delta t}\big(G_k(t_{j+1}) - G_k(t_j)\big)$$

$$-\frac{\Delta t \mu_k}{2} G_k(t_{j+1}) - \frac{\Delta t \mu_k}{2} G_k(t_j) = G_k(t_{j+1}) - G_k(t_j)$$

$$G_k(t_j)\left(1 - \frac{\Delta t \mu_k}{2}\right) = G_k(t_{j+1})\left(1 + \frac{\Delta t \mu_k}{2}\right)$$

$$\frac{1 - \frac{\Delta t \mu_k}{2}}{1 + \frac{\Delta t \mu_k}{2}} = \frac{G_k(t_{j+1})}{G_k(t_j)}$$

Since the numerator is always smaller than the denominator, the fraction which we found will always satisfy

$$\left|\frac{1 - \frac{\Delta t \mu_k}{2}}{1 + \frac{\Delta t \mu_k}{2}}\right| \leq 1$$

independent of the choice of Δt and Δx. This means that the Crank-Nicolson scheme is stable for any choice of step lengths for t and x.

4 Mathematical theory in two dimensions

The derivation of Fick's laws can be generalised to higher dimensions, giving

$$\vec{J} = -D\nabla C \qquad \frac{\partial C}{\partial t} = D\nabla^2 C$$

Our differential equation can therefore be written in two dimensions as

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{\partial u}{\partial t} \tag{17}$$

Time and position are again discretised, with the notation $u_{i,j,k} = u(x_i, y_j, t_k)$. For all simulations, Δy will be set equal to Δx.

4.1 Analytical solution in two dimensions

For the two-dimensional case, we have chosen to look at a specific set of initial and boundary conditions which make the analytical solution much easier. This analytical solution will later be used to verify that the numerical solver gives acceptably accurate results.

So, consider equation (17) with the following boundaries over a spatial domain of size $L \times L$:

$$\begin{aligned} u(0,y,t) = u(L,y,t) &= 0, && y \in (0,L), \quad t > 0 \\ u(x,0,t) = u(x,L,t) &= 0, && x \in (0,L), \quad t > 0 \\ u(x,y,0) &= \sin\left(\frac{\pi}{L}x\right)\sin\left(\frac{\pi}{L}y\right), && \text{for } x,y \in (0,L) \end{aligned} \tag{18}$$

This choice of boundary conditions has no physical interpretation, and is made purely because it greatly simplifies the analytical solution. These boundary conditions physically correspond to a predisposition of matter in the middle of the grid, with drains at all borders. In the steady state, it is therefore expected that all matter has left the grid, i.e. $u(x,y) = 0$ everywhere.

Solving the two-dimensional case is done in almost exactly the same way as for the one-dimensional case. Because of the similarity to what we did in section 3.1 on page 7, the calculation has been placed in appendix A.

The analytical result is

$$u(x,y,t) = \sin\left(\frac{\pi}{L}x\right)\sin\left(\frac{\pi}{L}y\right)\mathrm{e}^{\frac{-2\pi^2}{L^2}t} \tag{19}$$

4.2 An explicit numerical algorithm

To derive an explicit scheme for solving the diffusion equation in two dimensions, the same approximations to derivatives as in the derivation on the Forward Euler scheme is used. The

error terms are also the same.

$$\frac{\partial u}{\partial t} \approx \frac{u_{i,j,k+1} - u_{i,j,k}}{\Delta t}$$
$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1,j,k} + u_{i-1,j,k} - 2u_{i,j,k}}{(\Delta x)^2}$$
$$\frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i,j+1,k} + u_{i,j-1,k} - 2u_{i,j,k}}{(\Delta y)^2}$$

When inserting these into the two-dimensional diffusion equation, we get

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

We are once again interested in the value at t_{k+1}. Multiplying by Δt, defining $\alpha = \Delta t/(\Delta x)^2 = \Delta t/(\Delta y)^2$ and solving for the interesting value gives

$$u_{i,j,k+1} = (1 - 4\alpha)u_{i,j,k} + \alpha\left(u_{i+1,j,k} + u_{i-1,j,k} + u_{i,j+1,k} + u_{i,j-1,k}\right)$$

Since everything on the right-hand side is known, this is an explicit scheme. Parallel to the one-dimensional Forward Euler scheme, the global error consists of three terms, which are proportional to Δt, $(\Delta x)^2$ and $(\Delta y)^2$ respectively.

5 Implementation and code overview

The code can be found at GitHub[1]. See makefile for usage.

- onedimlib.cpp contains the class `OneDimSolver`, with methods for all the one-dimensional schemes discussed.
- onedim_ui.cpp is a command-line interface to the solver above, which uses its arguments to run a simulation.
- transpose.py swaps the rows and columns in a datafile from onedim_ui.cpp for plotting.
- tridiagonalsolver.cpp contains the tridiagonal solver from project 1.
- deltaxtest.sh and deltaxtest.gpi run and plot the one-dimensional simulations.
- analysis1d.py and analysis1d.gpi run and plot the stability analysis.
- animate1d.py takes a data file from onedim_ui.cpp as input argument and makes an animated gif.
- simple_diffusion.py implements an alternative, much easier way of simulating diffusion, as discussed in appendix B.
- twodim.cpp contains the explicit solver for the two-dimensional diffusion equation, and runs the simulation based on command-line arguments.
- plottwodim.gpi plots datafiles from twodim.cpp.

To further reduce the already nearly non-existant runtime, the explicit schemes in both one and two dimensions have been parallelised using `OpenMP`.

5.1 Visualisation

For easy visualisation of how the concentration evolves with time, we have made the script animate1d.py, which creates an animated gif based on the data from a file generated by onedim_ui.cpp.

See README.md (also shown at the bottom of the GitHub repository) for an example.

[1] https://github.com/anjohan/Project5

6 Results in one dimension

6.1 Qualitative overview for different Δx

As derived in section 3.3.3, the Forward Euler scheme requires that $\Delta t \leq \frac{1}{2}(\Delta x)^2$. Simulations with $\Delta x = 1/10$ and $\Delta x = 1/100$ and $\Delta t = \frac{1}{2}(\Delta x)^2$ gives the figure below.

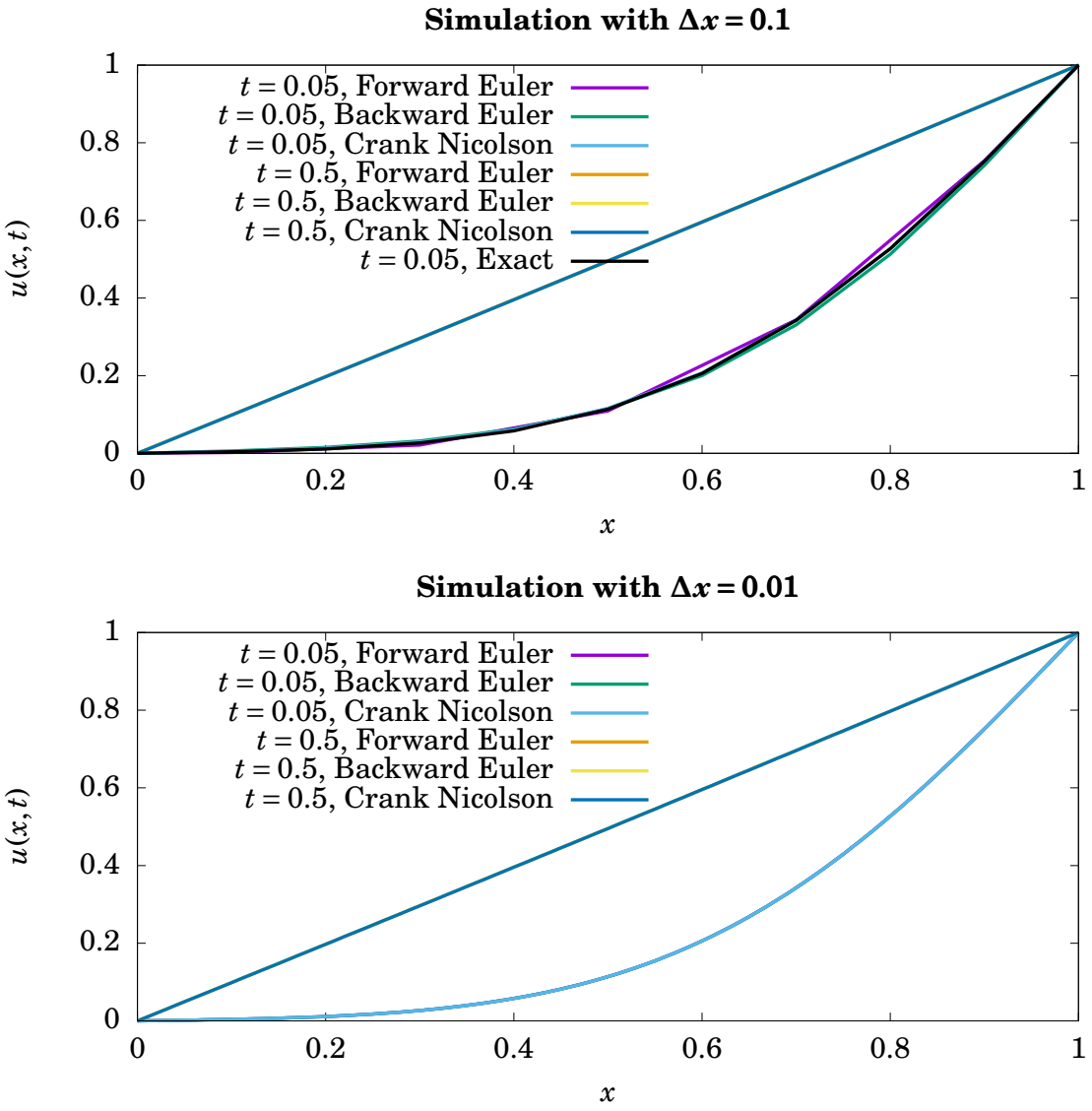

Figure 2: Results for two different values of Δx. Generated by deltaxtest.sh.

It is clear from the figure that all methods give acceptable results for the tested values of Δx. As expected from the error terms, the Crank Nicolson scheme slightly outperforms the others. For $\Delta x = 1/100$, there is essentially no difference. We see that the steady state requirement of linearity from section 2.3 is fulfilled.

6.2 Quantitative error analysis

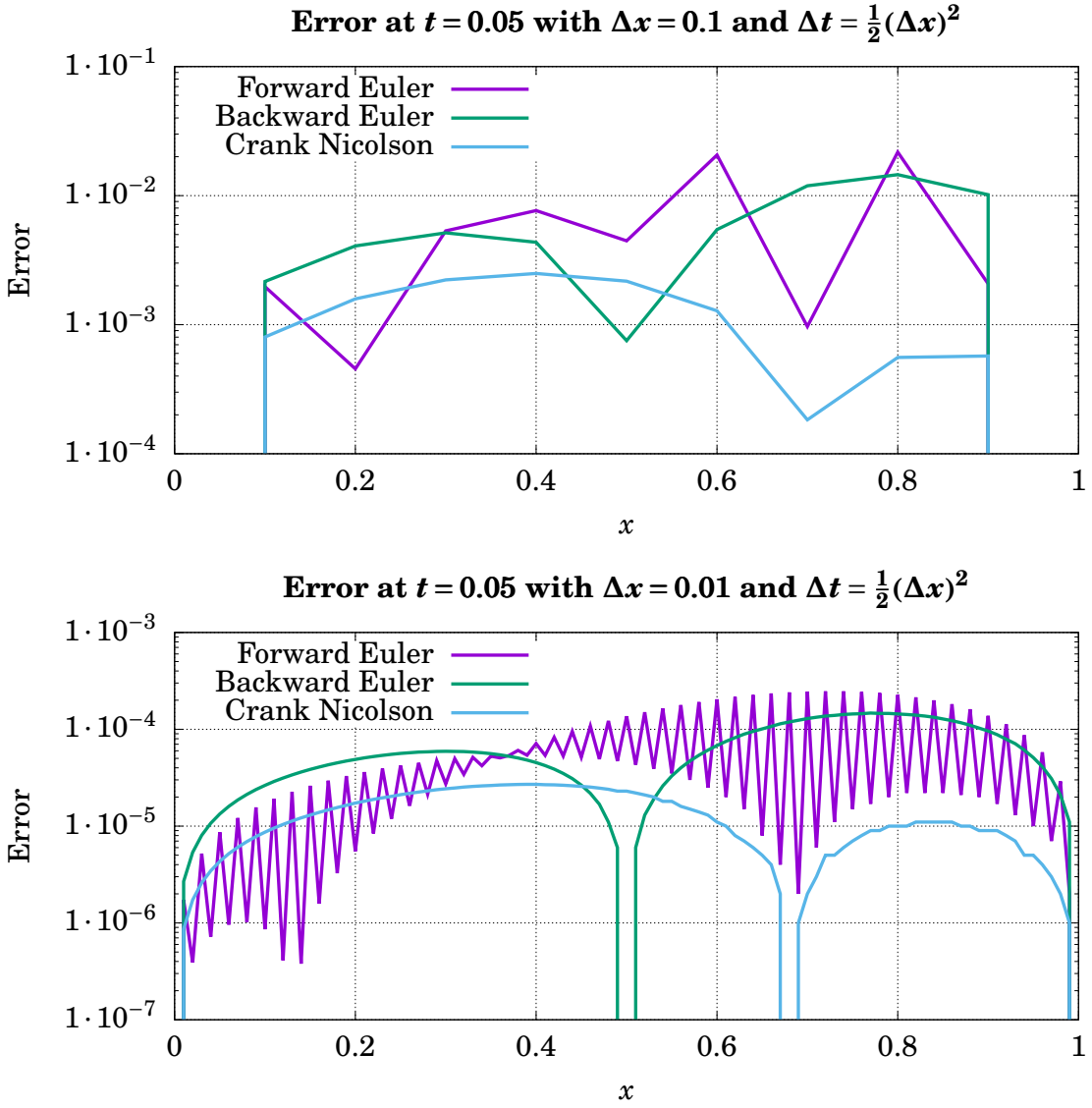

Figure 3: A closer look at the errors in the previous figure.

From the derivation of the methods and their error terms, we know that the Euler schemes have one error term proportional to Δt and one proportional to $(\Delta x)^2$, while the Crank-Nicolson scheme has error terms proportional to $(\Delta t)^2$ and $(\Delta x)^2$. Since Δt has been set to $\frac{1}{2}(\Delta x)^2$, the error in the Euler schemes should decrease by two orders of magnitude when Δx is reduced from 0.1 to 0.01, which fits well by the computed errors. As expected, the Crank-Nicolson error decreases more rapidly, but an exact prediction can not be made since the error term proportional to $(\Delta x)^2$ should decrease by two orders of magnitude and the term proportional to $(\Delta t)^2$ by four.

6.3 Stability analysis

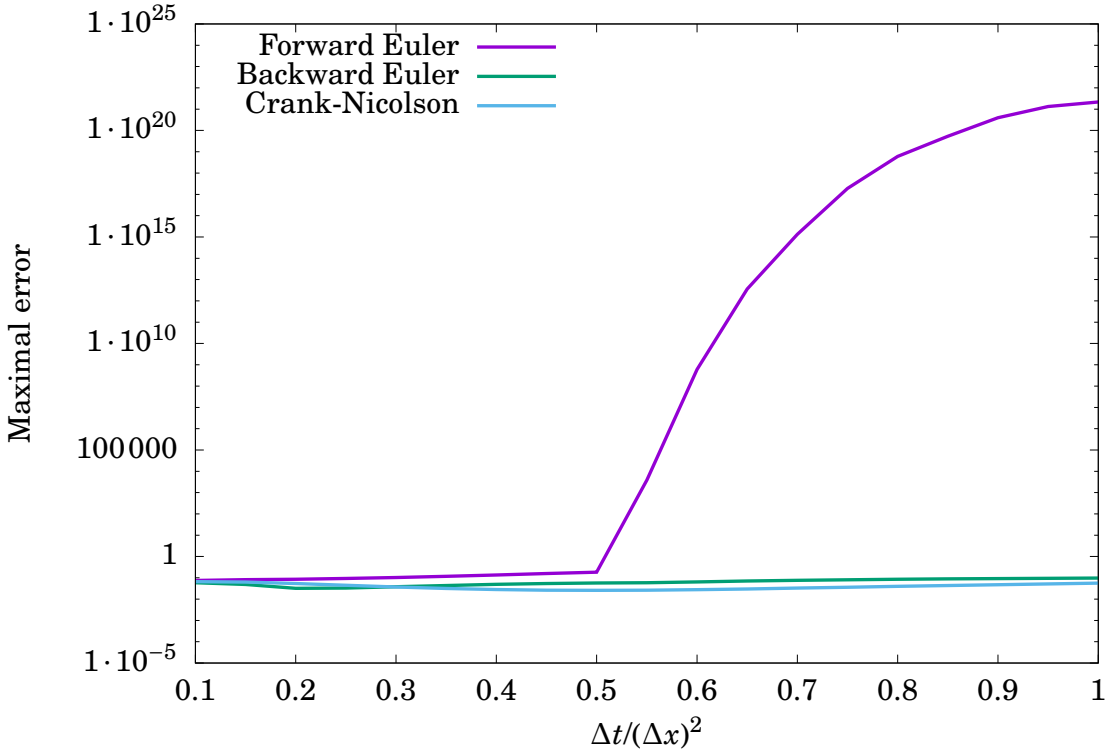

Figure 4: Visual verification of the criterion for the ratio Δt and Δx found through von Neumann analysis, with $\Delta x = 0.1$. The analytically derived $\Delta t/(\Delta x)^2 \leq 1/2$ fits well with the result for the Forward Euler scheme, while the other schemes do not rely on this ratio, as expected. Generated by analysis1d.py.

7 Results in two dimensions

7.1 Comparison of different Δx values

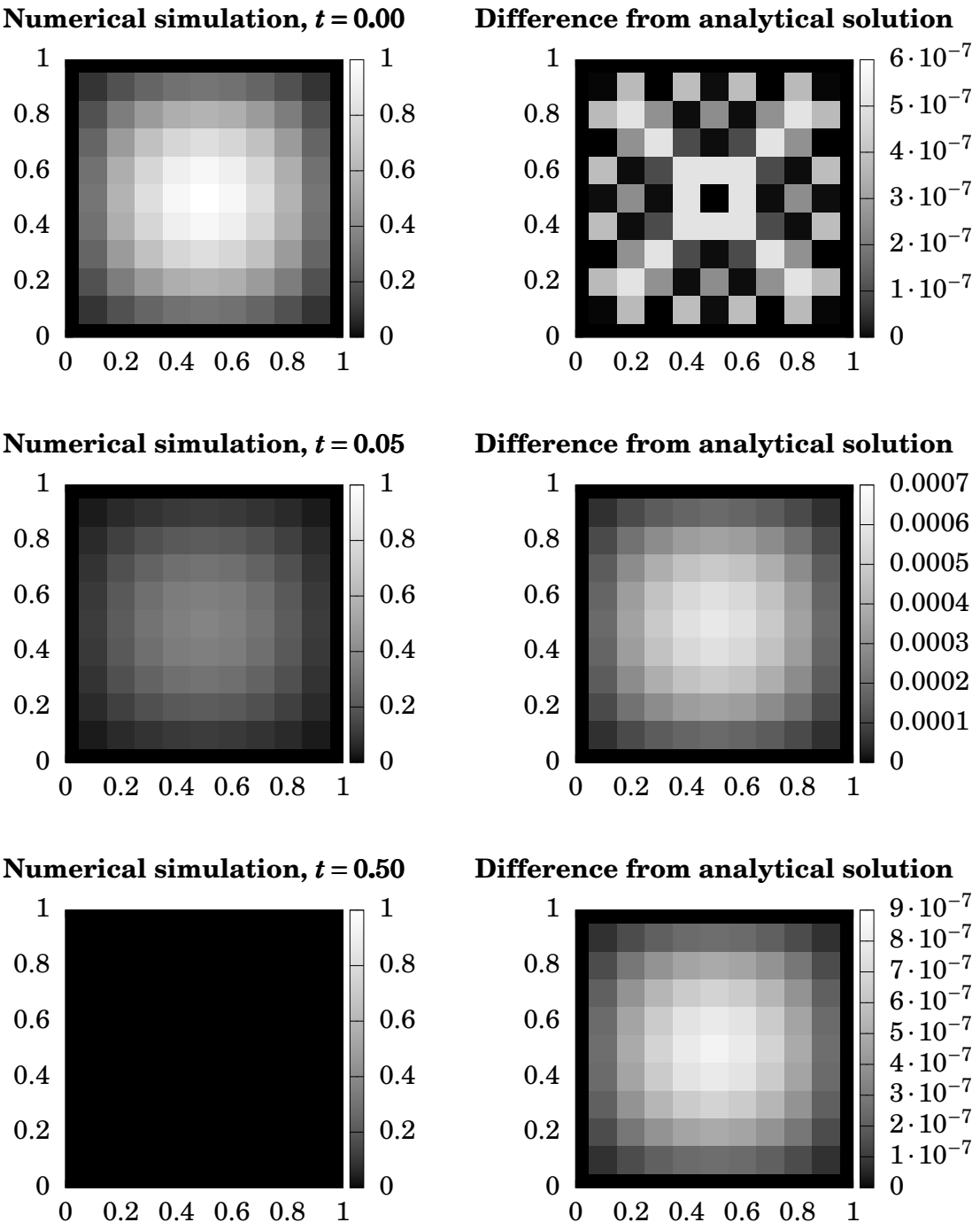

Figure 5: Simulation with $\Delta t = 10^{-3}$ and $\Delta x = \Delta y = 10^{-1}$. Note the difference in color scales between the figures.

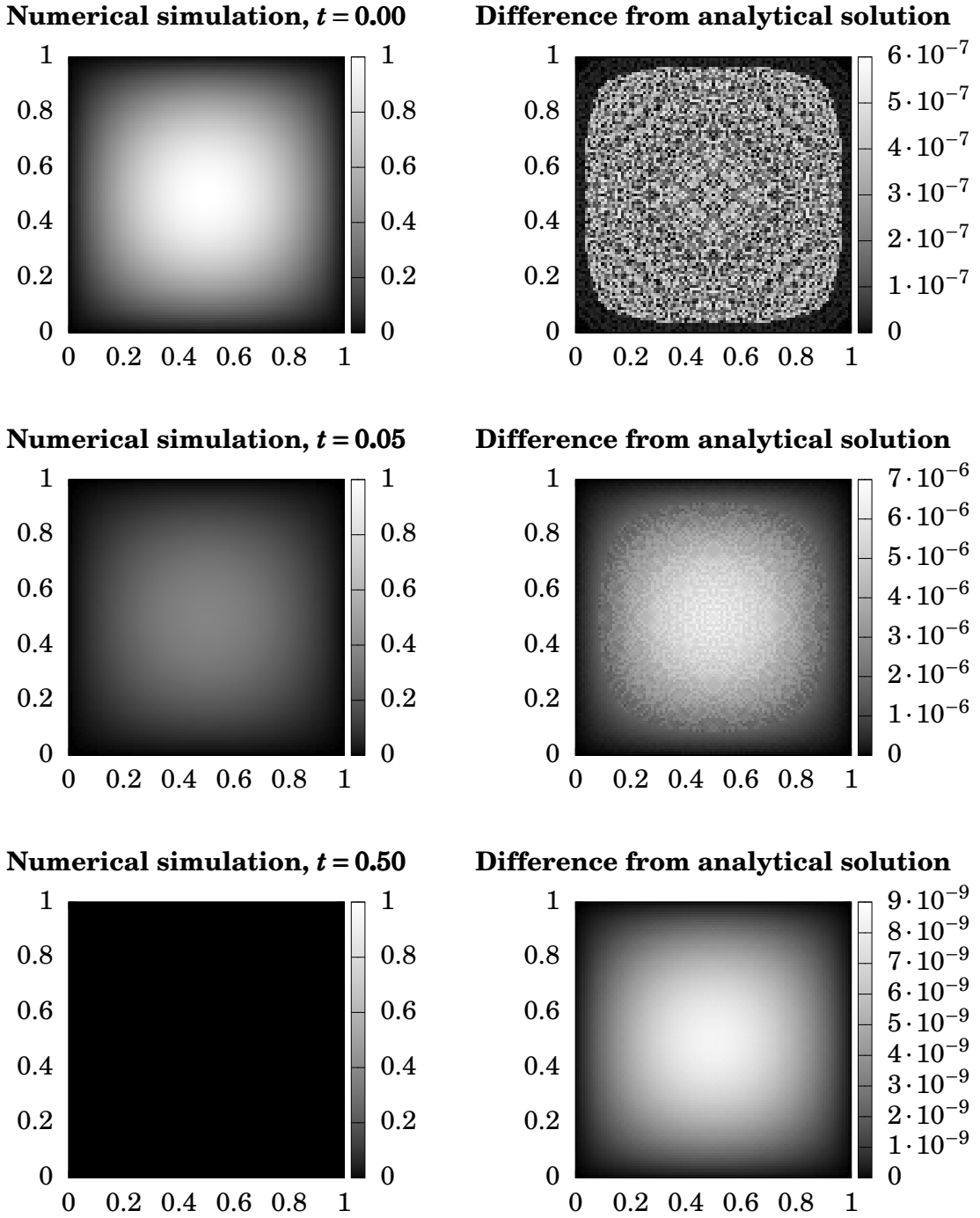

Figure 6: Simulation with $\Delta t = 10^{-5}$ and $\Delta x = \Delta y = 10^{-2}$.

From the figures above, it is clear that a reduction in the values of Δt and Δx reduces the error. Quantitatively, we see that the error is reduced by approximately two orders of magnitude when Δt is reduced from 10^{-3} to 10^{-5} and $\Delta x = \Delta y$ from 10^{-1} to 10^{-2}, which fits well with the numerical scheme having error terms proportional to Δt, $(\Delta x)^2$ and $(\Delta y)^2$.

8 Conclusion

We have now seen how the diffusion equation can be solved numerically in one and two dimensions. In both situations, it is clear that an explicit algorithm is easy to derive and implement and will give reasonable results when the spatial and temporal step lengths are chosen wisely. On the other hand, it has also become apparent that an unwise choice can lead to spectacularly wrong results, while the implicit methods discussed give stable results no matter what step lengths are chosen, with the Crank-Nicolson scheme being the most accurate.

Since we were able to reuse an efficient tridiagonal solver from a previous project, the added difficulties in derivation and implementation of the implicit methods are clearly worth the extra effort when simulating diffusion in a finer mesh of spatial points for which the criteria of the explicit scheme would require a lot of CPU time.

As such, future work should include the development of an implicit scheme for the two-dimensional diffusion equation. Additionally, it would be beneficial to either analytically derive or numerically search for the stability requirement of the explicit scheme as done for one-dimensional equation. For practical applications it might also be beneficial to determine analytical solutions for initial and boundary conditions corresponding to other physical situations.

Considering that the 25 or so pages and about 400 lines of code we have written about one-dimensional diffusion could be replaced by the two paragraphs and 17 lines of Python code in appendix B, it might also be wise to look at other ways of studying diffusion, for example Monte Carlo simulations.

References

[1] Anders Johansson. “Project 1”. In: *FYS3150, Computational Physics* (Sept. 2016). URL: https://github.com/anjohan/Offentlig/blob/master/FYS3150/Oblig1/Johansson_Anders_FYS3150_Oblig1.pdf.

[2] Aslak Tveito and Ragnar Winther. *Introduction to partial differential equations: a computational approach*. Vol. 29. Springer Science & Business Media, 2004.

[3] Morten Hjorth-Jensen. *Computational Physics*. Lecture notes. 2015. URL: https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf.

Appendix

A Solving for the two dimensional case

Here, we will explain how we found the solution equation (19) on page 19 with the given problem equation (17) on page 19 with the boundaries equation (18) on page 19.

We make an ansatz that

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

Equation (17) on page 19 gives us that

$$\begin{aligned} G(t)\left(\frac{\partial^2 F(x,y)}{\partial x^2} + \frac{\partial^2 F(x,y)}{\partial y^2}\right) &= F(x,y)\frac{\partial G(t)}{\partial t} \\ \frac{1}{F(x,y)}\left(\frac{\partial^2 F(x,y)}{\partial x^2} + \frac{\partial^2 F(x,y)}{\partial y^2}\right) &= \frac{G'(t)}{G(t)} \end{aligned} \tag{20}$$

Since the ratios of two functions with different variables are equal, we must have that they are all equal to a constant $-\lambda$. We will first solve for $F(x,y)$.

Assume that we can again separate $F(x,y)$ into

$$F(x,y) = X(x)Y(y)$$

Inserting this into equation (20) gives us

$$\begin{aligned} \frac{Y(y)X''(x) + Y''(y)X(x)}{X(x)Y(y)} &= -\lambda \\ \frac{X''}{X} + \frac{Y''}{Y} &= -\lambda \end{aligned}$$

Since an addition of the ratios of two functions with different variables gives a constant, each of the ratios must therefore be a constant.We will first solve for X:

$$\begin{aligned} \frac{X''}{X} &= -\left(\frac{Y''}{Y} + \lambda\right) \\ \frac{X''}{X} &= -\mu \end{aligned} \tag{21}$$

This gives a general solution

$$X(x) = c_1\cos\left(\sqrt{\mu}x\right) + c_2\sin\left(\sqrt{\mu}x\right)$$

Since $u(0,y,t) = u(L,y,t) = 0$ from equation (18) on page 19, we must have that $c_1 = 0$ and

$$\begin{aligned} \sqrt{\mu_k}L &= k\pi, \quad k \in \mathbb{N} \\ \mu_k L^2 &= (k\pi)^2 \\ \mu_k &= \left(\frac{k\pi}{L}\right)^2 \end{aligned} \tag{22}$$

which gives that $c_2 = c_k$, and

$$X(x) = c_k \sin\left(\frac{k\pi}{L}x\right)$$

Now, we have to solve for Y:

From equation (21) on the preceding page and equation (22) on the previous page, we have that

$$\begin{aligned}\frac{Y''}{Y} + \lambda &= \left(\frac{k\pi}{L}\right)^2 \\ \frac{Y''}{Y} &= -\left(-\left(\frac{k\pi}{L}\right)^2 + \lambda\right) \\ \frac{Y''}{Y} &= -\nu\end{aligned}$$

This gives a general solution of Y:

$$Y(y) = d_1 \cos\left(\sqrt{\nu}y\right) + d_2 \sin\left(\sqrt{\nu}y\right)$$

By using the same procedure of finding X, we get that $d_1 = 0$ and

$$\nu = \left(\frac{m\pi}{L}\right)^2$$

for $m \in \mathbb{N}$. Setting $d_2 = d_m$, we get the solution of Y to be

$$Y(y) = d_m \sin\left(\frac{m\pi}{L}y\right)$$

with

$$\nu_m = \left(\frac{m\pi}{L}\right)^2 \tag{23}$$

Therefore, we have the particular solution of F for any value of k, m.

$$F_{k,m}(x,y) = c_k d_m \sin\left(\frac{k\pi}{L}x\right)\sin\left(\frac{m\pi}{L}y\right)$$

In equation (20) on the preceding page, we can solve for $G(t)$:

$$\frac{G'(t)}{G(t)} = -\lambda$$

which gives the general solution

$$G(t) = C\mathrm{e}^{-\lambda t}$$

where we set the constant $C = 1$. To find λ, we just use the expression for ν_m which we found in equation (23):

$$\begin{aligned}\nu_m &= \left(\frac{m\pi}{L}\right)^2 \\ -\left(\frac{k\pi}{L}\right)^2 + \lambda_{k,m} &= \left(\frac{m\pi}{L}\right)^2\end{aligned}$$

$$\lambda_{k,m} = \left(\frac{m\pi}{L}\right)^2 + \left(\frac{k\pi}{L}\right)^2$$

we see that the value of λ depends on the value of k and m, thus giving that $\lambda = \lambda_{k,m}$.

In total, we have the following particular solution of u:

$$u_{k,m}(x,y,t) = c_k d_m \sin\left(\frac{k\pi}{L}x\right)\sin\left(\frac{m\pi}{L}y\right)e^{-\lambda_{k,m}t}$$

To decide c_k and d_m, we have to look at the boundary of u at $t = 0$, which gives

$$\sin\left(\frac{\pi}{L}x\right)\sin\left(\frac{\pi}{L}y\right) = c_k d_m \sin\left(\frac{k\pi}{L}x\right)\sin\left(\frac{m\pi}{L}y\right)$$

meaning that $c_k = d_m = k = m = 1$ and therefore

$$\lambda_{k,m} = 2\left(\frac{\pi}{L}\right)^2$$

This gives the final solution of our problem described in equation (17) on page 19 and equation (18) on page 19 to be

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

B Simple simulation of diffusion in Python

As discussed in the derivation of Fick's laws in section 2, diffusion is simply the random walk of many particles, resulting in a net flux if there's a concentration gradient. This can be modelled as particles jumping one step length either to the left or to the right, with a 50 % probability for each direction. When the number of particles goes to infinity, the law of large numbers says that exactly half the particles will jump to the left while the other half will jump to the right.

This means that at the time t_{j+1}, the concentration at a point x_i will equal the sum of half the concentration at x_{i-1} and half the concentration at x_{i+1} at the time t_j. For fun, we have implemented this way of thinking about diffusion in the short script simple_diffusion.py, and the results look very reasonable!

Code snippet 1: simple_diffusion.py

```
from matplotlib.pylab import *

nt = 1000001
nx = 101

x = linspace(0,1,nx)
u = zeros((nt,nx))

u[:,-1] = 1 #Boundary conditions

for j in range(nt-1):
    u[j+1,1:-1] = 0.5*(u[j,:-2] + u[j,2:])

plot(x,u[1000],label="1 000 time steps")
plot(x,u[-1],label="1 000 000 time steps")
legend(loc="best")
show()
```